The complexity of scheduling typed task systems with and without communication delays*

Jacques Verriet

Department of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
E-mail: jacques@cs.uu.nl

Abstract

We consider the problem of scheduling typed task systems subject to precedence constraints and communication delays. For such task systems, the set of tasks is divided into sets of tasks of the same type and there are different types of processors. In a schedule for a typed task system, the type of a task must correspond to the type of the processor on which it is executed.

It is proved that if there is only one processor of each type, then, without communication delays, a schedule of length at most three can be constructed in polynomial time and that determining the existence of a schedule of length at most four is an NP-complete decision problem. Moreover, we show that with non-zero communication delays, determining the existence of a schedule of length at most four can be done in polynomial time and that determining the existence of schedules of length at most five is an NP-complete decision problem. The problem of constructing minimum-length schedules for typed task systems on two processors is shown to be an NP-hard optimisation problem.

We prove that if there are no restrictions on the number of processors of one type, then, without communication delays, the existence of a schedule of length at most two can be determined in polynomial time and that determining the existence of a schedule of length at most three is an NP-complete decision problem. In addition, it is proved that with non-zero communication delays, the existence of schedules of length at most three can be done in polynomial time and that determining the existence of a schedule of length at most four is a NP-complete decision problem.

For the special case of interval-ordered tasks, it is proved that minimum-length schedules for interval orders with unit-length tasks on m processors can be constructed in polynomial time and that constructing minimum-length schedules for preallocated interval-ordered tasks with arbitrary task lengths is an NP-hard optimisation problem.

1 Introduction

Many parallel computer systems contain components that can execute only one type of tasks. For instance, the evaluation of arithmetic expressions is often done by special processors. In this report, we will study the complexity of scheduling problems in which every processor and every task has a type. If a task is to be executed on a processor, then their types must coincide.

The execution of a set of precedence-constrained tasks on a number of typed processors gives rise to a lot of data transfers. Data-dependent tasks of different types have to be executed on different processors. So the result of a task needs to be transferred to other processor in order to execute data-dependent tasks on this processors. We will consider two kinds of scheduling problems, problems in which the duration of these data transfers is neglected and problems in which

*This research was partially supported by ESPRIT Long Term Research Project 20244 (project ALCOM IT: *Algorithms and Complexity in Information Technology*).

these durations are not neglected.

Little is known about the general problem of scheduling typed task systems: Jaffe [16] and Jansen [18] studied the problem of scheduling typed task systems without any restrictions on the number of processors of one type. However, there are two special cases that have been studied extensively. The first is the problem of scheduling a set of precedence-constrained tasks on a number of identical processors. In this special case, all tasks and all processors have the same type. Overviews of the results for this special case are given by Cheng and Sin [6] for scheduling without communication delays, and by Chrétienne and Picouleau [7] for scheduling with communication delays.

The second well-studied special case is the problem of scheduling preallocated tasks subject to precedence constraints. In this special case, there is exactly one processor of each type. Consequently, it is known on which processor a task has to be executed. Job shops form collections of chains of preallocated tasks in which no immediate successors has to be executed on the same processor [11], a flow shop is a job shop in which all chains have the same length and the i^{th} tasks of all chains must be executed on the same processor [11]. More general precedence constraints are considered by Bernstein et al. [2, 3, 4], who studied the problem of scheduling preallocated tasks on two processors of different types, and by Goyal [13], who studied the complexity of scheduling preallocated tasks on m processors.

In this report, we study the complexity of scheduling preallocated tasks and the more general problem of scheduling arbitrary typed task systems. For both types of problems, we will consider problems in which the duration of the data transfers is negligible and ones in which these durations are not negligible.

This report is divided into two parts. The first part consists of Sections 3, 4 and 5. In these sections, we consider the problem of scheduling preallocated task systems. In Section 3, it is proved that constructing minimum-length schedules on two processors is an NP-hard optimisation problem even if the tasks form a collection of chains of unit-length tasks and the communication delays are of unit length.

Section 4 deals with the problem of scheduling preallocated tasks without communication delays. We present an algorithm that determines the existence of a schedule of length at most three in polynomial time. Determining the existence of a schedule of length at most four is proved to be an NP-complete decision problem even if the precedence constraints form a collection of fork graphs with unit-length tasks.

In Section 5, we study the problem of scheduling preallocated tasks subject to non-zero communication delays. It is shown that scheduling preallocated tasks with non-zero communication delays is a special case of scheduling without communication delays. It will be proved that this special case is less complicated than the general case: the existence of a schedule of length at most four can be determined in polynomial time. Determining the existence of schedules of length at most five is shown to be an NP-complete decision problem.

The second part of this report consists of Sections 6, 7 and 8. In these sections, we study the problem of scheduling arbitrary typed task systems. In Section 6, a simple polynomial-time algorithm is presented that determines the existence of a schedule of length at most two. The problem of determining the existence of schedules of length at most three is shown to be an NP-complete decision problem even if the tasks form a collection of fork graphs with unit-length tasks.

Section 7 deals with the problem of scheduling typed task systems subject to non-zero communication delays. Although this is not a special case of scheduling typed task systems without communication delays, it is shown that it is less complicated than scheduling without communication delays. It is proved that the existence of schedules of length at most three can be determined in polynomial time, whereas determining the existence of schedules of length at most four is shown to be an NP-complete decision problem.

Section 8 considers typed task systems for which the precedence constraints form an interval

order. We present a polynomial-time algorithm that constructs minimum-length schedules for interval-ordered tasks of unit length that must be scheduled subject to unit-length communication delays. In addition, it is proved that if the task lengths are unrestricted, then constructing minimum-length schedules for interval orders is an NP-hard optimisation problem even for preallocated interval-ordered tasks on two processors.

2 Preliminaries

In this report, I will present several algorithms that construct schedules for typed precedence-constrained tasks on m typed processors. The precedence-constrained tasks will be represented by a directed acyclic graph. An instance of such a scheduling problem is represented by a tuple (G, m, τ, μ, c), where $G = (V, E)$ is a directed acyclic graph or precedence graph, $m \in \{2, 3, \ldots, \infty\}$ is the number of processors, $\tau : V \cup \{1, \ldots, m\} \to \{1, \ldots, k\}$ assigns a *type* to every task of G and every processor, $\mu : V \to \mathbb{Z}^+$ an *execution length* to every node of G, and $c : E \to \mathbb{N}$ a *communication duration* to every arc of G. The number of processors of type i is denoted by m_i. Clearly, $\sum_{i=1}^{k} m_i = m$.

A node of G corresponds to a task of the typed task system and will be called a *task* of G. The execution of a task u of G takes $\mu(u)$ time units on a processor of type $\tau(u)$ during which this processor cannot executed another task. u cannot be executed on a processor of another type than $\tau(u)$. Consider two tasks u_1 and u_2 of G, such that E contains the arc from u_1 to u_2. If u_1 is executed on processor p_1 and u_2 on processor $p_2 \neq p_1$, then the result of u_1 has to be transferred from processor p_1 to processor p_2 after u_1 is completed. This takes $c(u_1, u_2)$ time. During this delay, processors p_1 and p_2 can execute other tasks. If u_1 and u_2 are executed on the same processor, then no data transfer is required.

Let G be a precedence graph. $V(G)$ denotes the set of nodes of G and $E(G)$ the set of arcs. Throughout this report, we will assume that $V(G)$ contains n tasks and that $E(G)$ contains e arcs. Let u_1 and u_2 be two tasks of G. If $E(G)$ contains the arc (u_1, u_2), than u_2 is called a *child* of u_1 and u_1 a *parent* of u_2. This is denoted by $u_1 \prec_0 u_2$. The number of children of a task u is the *outdegree* of u; the *indegree* of u is the number of parents of u. If there is a directed path in G from u_1 to u_2, then u_1 is said to be a *predecessor* of u_2 and u_2 a *successor* of u_1. This is denoted by $u_1 \prec u_2$. If u_1 is neither a predecessor nor a successor of u_2, then u_1 and u_2 will be called *incomparable*. Otherwise, they are called *comparable*.

Let u be a task of G. If u has no successors, then it is called a *sink* of G; if u has no predecessors, then u is said to be a *source* of G. An *independent task* is a task of G that has no predecessors and no successors.

To define special instances (G, m, τ, μ, c), we will use two functions. For any set S, $\mathbf{1}_S$ is the function that assigns 1 to every element of S. Similarly, $\mathbf{0}_S$ is the function that assigns 0 each element of S. These functions are used to represent special instances. For example, the instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ corresponds to the problem of scheduling typed unit-length tasks on m typed processors without communication delays.

A *schedule* for an instance (G, m, τ, μ, c) is a pair of functions (σ, π), where $\sigma : V(G) \to \mathbb{N}$ assigns a *starting time* to every task of G and $\pi : V(G) \to \{1, \ldots, m\}$ a processor. A schedule (σ, π) is called a *feasible schedule* for (G, m, τ, μ, c) if, for all tasks $u_1 \neq u_2$ of G,

1. $\tau(\pi(u_1)) = \tau(u_1)$;
2. if $\pi(u_1) = \pi(u_2)$, then $\sigma(u_1) \geq \sigma(u_2) + \mu(u_2)$ or $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1)$;
3. if u_2 is a child of u_1, then $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1)$; and
4. if u_2 is a child of u_1 and $\pi(u_1) \neq \pi(u_2)$, then $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1) + c(u_1, u_2)$.

The first constraint ensures that each task is scheduled on a processor of the correct type. The second constraint states that no processor can execute two tasks at the same time. The third constraint ensures that a task is scheduled after its predecessors. The fourth constraint states that there is a communication delay between two comparable tasks on different processors.

Consider a feasible schedule (σ, π) for an instance (G, m, τ, μ, c). Let u be a task of G. Then u is said to be scheduled at time $\sigma(u)$ on processor $\pi(u)$. In addition, u is executed at times $\sigma(u), \ldots, \sigma(u) + \mu(u) - 1$. $\sigma(u) + \mu(u)$ is the *completion time* of u. The *length* or *makespan* of schedule (σ, π) is the maximum completion time of a task of G: the length of (σ, π) equals $\max_{u \in V(G)} \sigma(u) + \mu(u)$.

As stated in the previous section, scheduling without processor requirements is a special case of scheduling typed task systems. For such problems, all processors have the same type. An instance of a untyped problem (an untyped instance) will be denoted by a tuple (G, m, μ, c).

Scheduling preallocated tasks is another special case of scheduling typed task systems. This problem corresponds to scheduling typed task systems in which all processors have a different type. Consider an instance (G, m, τ, μ, c), such that $\tau(i) \neq \tau(j)$ for all processors $i \neq j$. Since each processor has a different type and the type of each task is known, the processor on which a task must be scheduled is known is advance. Hence such an instance will be denoted by (G, m, π, μ, c), where $\pi : V(G) \to \{1, \ldots, m\}$ assigns a processor to every task of G. A *schedule* for such an instance (G, m, π, μ, c) can be represented by a function $\sigma : V(G) \to \mathbb{N}$ assigning a starting time to every task of G. A schedule σ for (G, m, π, μ, c) is called *feasible* if (σ, π) is feasible for the corresponding instance (G, m, τ, μ, c). In other words, σ is a feasible schedule for (G, m, π, μ, c) if, for all tasks $u_1 \neq u_2$ of G,

1. if $\pi(u_1) = \pi(u_2)$, then $\sigma(u_1) \geq \sigma(u_2) + \mu(u_2)$ or $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1)$;
2. if u_2 is a child of u_1, then $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1)$; and
3. if u_2 is a child of u_1 and $\pi(u_1) \neq \pi(u_2)$, then $\sigma(u_2) \geq \sigma(u_1) + \mu(u_1) + c(u_1, u_2)$.

3 Preallocated tasks on two processors

In this section, we will consider the complexity of scheduling preallocated tasks on two processors subject to precedence constraints and communication delays. Goyal [13] proved that this problem without communication delays is NP-hard if all tasks have length one. Bernstein, et al [4] showed that this result remains true if the precedence constraints form a outforest with unit-length tasks. Jansen [18] proved that constructing minimum-length schedules for chains of unit-length tasks without communication delays is a strongly NP-hard optimisation problem. Picouleau [23] generalised the result of Goyal [13] for scheduling subject to unit-length communication delays: he showed that constructing minimum-length schedules for arbitrary precedence graphs with unit-length communication delays is an NP-hard optimisation problem.

In this section, the result of Jansen [18] will be generalised for scheduling subject to non-zero communication delays: it will be proved that the constructing a minimum-length schedule for an instance $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ is a strongly NP-hard optimisation even if G is a collection of chains. This is proved using a pseudo-polynomial reduction from 3PARTITION which is known to be a strongly NP-complete decision problem [10]. 3PARTITION is the following decision problem.

Problem. 3PARTITION
Instance. A set of $3k$ integers $A = \{a_1, \ldots a_{3k}\}$ and a positive integer B, such that $\frac{1}{4}B < a_i < \frac{1}{2}B$ for all $i \in \{1, \ldots, 3k\}$ and $\sum_{i=1}^{3k} a_i = kB$.
Question. Are there k pairwise disjoint subsets $A_1, \ldots, A_k$ of A, such that $\bigcup_{j=1}^{k} A_j = A$ and $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, k\}$?

TWO-PROCESSOR PREALLOCATION is the following decision problem.

Problem. TWO-PROCESSOR PREALLOCATION
Instance. An instance $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is a collection of chains and a positive integer D.
Question. Is there a feasible schedule for $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most D?

Using a pseudo-polynomial reduction from 3PARTITION similar to the one presented by Jansen [18] for chains without communication delays and the one presented by Bernstein, et al [4] for inforests without communication delays, I will show that TWO-PROCESSOR PREALLOCATION is a strongly NP-complete decision problem.

Lemma 3.1. *There is a pseudo-polynomial reduction from* 3PARTITION *to* TWO-PROCESSOR PREALLOCATION.

Proof. Consider an instance $A = \{a_1, \ldots, a_{3k}\}$, B of 3PARTITION. Construct an instance $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of TWO-PROCESSOR PREALLOCATION as follows. For each element a_i of A, construct a chain C_i consisting of $2a_i$ tasks $u_{i,1}, \ldots, u_{i,a_i}, v_{i,1}, \ldots, v_{i,a_i}$, such that $\pi(u_{i,j}) = 1$ and $\pi(v_{i,j}) = 2$ for all $j \in \{1, \ldots, a_i\}$. In addition, construct a chain C_0 consisting of $2m(B-2)+2$ tasks. C_0 is divided in $2m$ subchains $C_{0,1}, \ldots, C_{0,2m}$, such that the last task of $C_{0,j}$ is a predecessor of the first task of $C_{0,j+1}$. Subchain $C_{0,1}$ consists of $B-1$ tasks $c_{1,1}, \ldots, c_{1,B-1}$ and subchain $C_{0,2m}$ of $B-1$ tasks $c_{2m,1}, \ldots, c_{2m,B-1}$. For each $j \in \{2, \ldots, 2m-1\}$, subchain $C_{0,j}$ consists of $B-2$ tasks $c_{j,1}, \ldots, c_{j,B-2}$. For all $j \in \{1, \ldots, m\}$, the tasks of subchains $C_{0,2j-1}$ are to be scheduled on processor 2; those of subchains $C_{0,2j}$ on processor 1. Furthermore, let $D = 2mB - 2m + 1$. Now it will be proved that there are pairwise disjoint subsets $A_1, \ldots, A_k$ of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, k\}$ if and only if there is a feasible schedule for $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most D.

($\Rightarrow$) Let $A_1, \ldots, A_k$ be pairwise disjoint subsets of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, k\}$. Construct a schedule σ for $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ as follows. For all $j \in \{1, \ldots, B-1\}$, let $\sigma(c_{1,j}) = j - 1$. For each $i \in \{1, \ldots, m-1\}$ and $j \in \{1, \ldots, B-2\}$, let

$$\sigma(c_{i,2j}) = B + (i-1)(2B-2) + j - 1$$

and

$$\sigma(c_{i,2j+1}) = i(2B-2) + j.$$

In addition, let

$$\sigma(c_{2m,j}) = (m-1)(2B-2) + B + j - 1$$

for all $j \in \{1, \ldots, B-1\}$. Each set A_j contains exactly three elements. Assume $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$ for all $j \in \{1, \ldots, k\}$. Schedule the tasks of the chains C_{j_1}, C_{j_2} and C_{j_3} as follows. For all $i \in \{1, \ldots, a_{j_1}\}$, let

$$\sigma(u_{j_1,i}) = (j-1)(2B-2) + i - 1$$

and

$$\sigma(v_{j_1,i}) = (j-1)(2B-2) + B + i - 2.$$

For all $i \in \{1, \ldots, a_{j_2}\}$, let

$$\sigma(u_{j_2,i}) = (j-1)(2B-2) + a_{j_1} + i - 1$$

and

$$\sigma(v_{j_2,i}) = (j-1)(2B-2) + B + a_{j_1} + i - 2.$$

In addition, for all $i \in \{1, \ldots, a_{j_3}\}$, let

$$\sigma(u_{j_3,i}) \;=\; (j-1)(2B-2) + a_{j_1} + a_{j_2} + i - 1$$

and

$$\sigma(v_{j_3,i}) \;=\; (j-1)(2B-2) + B + a_{j_1} + a_{j_2} + i - 2.$$

The last task of subchain $C_{0,1}$ finishes at time $B-1$ on processor 2; the first of subchain $C_{0,2}$ starts at time B on processor 1. For each $j \in \{1, \ldots, m-1\}$, the last task of subchain $C_{0,2j}$ is completed at time $(j-1)(2B-2)+2B-2 = j(2B-2)$ on processor 2 and the first of subchain $C_{0,2j+1}$ at time $j(2B-2)+1$ on processor 1. Moreover, all tasks of subchain $C_{0,2j+1}$ are finished at time $j(2B-2)+B-1$ and at time $j(2B-2)+B$, the first task of subchain $C_{0,2j+2}$ starts. So chain C_0 is scheduled without violation of the precedence constraints and the communication delays.

Consider the set $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$. The tasks of chains C_{j_1}, C_{j_2} and C_{j_3} are scheduled at times $(j-1)(2B-2), \ldots, (j-1)(2B-2)+B-1$ on processor 1 and at times $(j-1)(2B-2)+B-1, \ldots, j(2B-2)$ on processor 2. This does not interfere with the execution of the tasks of chain C_0. Task $u_{j_i,a_{j_i}}$ is completed at time

$$(j-1)(2B-2) + \sum_{i' \le i} a_{j_{i'}}$$

on processor 1; its child, $v_{j_i,1}$, starts at time

$$(j-1)(2B-2) + B - 1 + \sum_{i' \le i} a_{j_{i'}}$$

on processor 2. Since $B \ge 3$, the execution of these tasks does not violate the precedence constraints and the communication delays. So σ is a feasible schedule for $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$.

The last task of chain C_0 (that is task $c_{2m,B-1}$) is completed at time $(m-1)(2B-2)+2B-1 = 2mB - 2m + 1 = D$. The last task of a chain C_j finishes at time $m(2B-2)+1 = 2mB - 2m + 1 = D$. So σ is a schedule of length D.

($\Leftarrow$) Let σ be a feasible schedule for $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most D. Because of the unit-length communication delays, there is only one way in which the tasks of chain C_0 can be executed. The tasks of C_0 are scheduled on processor 2 at times $0, \ldots, B-1$ and at times $j(2B-2)+1, \ldots, j(2B-2)+B-2$ for all $j \in \{1, \ldots, m-1\}$. Tasks of C_0 are scheduled on processor 1 at times $(j-1)(2B-2)+B, \ldots, (j-1)(2B-2)+2B-3$ for all $j \in \{1, \ldots, m-1\}$ and at times $(m-1)(2B-2)+B, \ldots, (m-1)(2B-2)+2B-2$.

The number of tasks of G equals $2\sum_{i=1}^{3k} a_i + 2m(B-2) + 2 = 2D$. Hence both processors execute a task at all times $0, \ldots, D-1$. Consider the tasks scheduled at times $B-1, \ldots, 2B-2$ on processor 2. This time interval contains exactly B tasks $v_{i,j}$. Since $\frac{1}{4}B < a_i < \frac{1}{2}B$ for all $i \in \{1, \ldots, 3k\}$, these tasks are contained in at least three chains C_i. The tasks $u_{i,j}$ of these chains are executed at times $0, \ldots, B-1$ on processor 1. The tasks $u_{i,j}$ of at most three chains C_i can be completed at time B. So the tasks $u_{i,j}$ and $v_{i,j}$ executed before time $2B-2$ form three chains C_{i_1}, C_{i_2} and C_{i_3}. Since σ does not have any idle time slots, $a_{i_1} + a_{i_2} + a_{i_3} = B$. Choose $A_1 = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. The same argument can be repeated for the remaining intervals containing tasks of chains C_i. This way one constructs disjoint subsets $A_1, \ldots, A_k$ of A, such that $\sum_{a \in A_j} a = B$ for all $j \in \{1, \ldots, k\}$.

□

The reduction shown in Lemma 3.1 proves that TWO-PROCESSOR PREALLOCATION is a strongly NP-complete decision problem and that constructing minimum-length schedules for a collection of chains on two processors is a strongly NP-hard optimisation problem.

Theorem 3.2. *Constructing minimum-length schedules for instances* $(G, 2, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$*, such that* G *is a collection of chains, is a strongly* NP*-hard optimisation problem.*

Proof. Obvious from Lemma 3.1. □

Although constructing minimum-length schedules for preallocated tasks on two processors is unlikely to be tractable, the two-processor job shop scheduling problem with unit-length tasks is solvable in polynomial time. Hefetz and Adiri [14] presented a level algorithm that constructs minimum-length schedules for the two-processor job shop with tasks of unit length in polynomial time. Their algorithm depends on the constraint that two subsequent tasks in one job (chain of tasks) have to be executed on different processors. Lemma 3.1 shows that if this constraint is removed, then the constructing minimum-length schedules becomes a strongly NP-hard optimisation problem. If another constraint of this problem is relaxed, then the problem becomes NP-hard as well: Lenstra and Rinnooy Kan [21] showed that constructing minimum-length schedules for the three-processor job shop with unit-length tasks and for the two-processor job shop with tasks of length 1 and 2 are strongly NP-hard optimisation problems.

For special classes of precedence graphs, a minimum-length schedule for an instance $(G, m, \pi, \mathbf{1}_{V(G)}, c)$ can be constructed in polynomial time. The *width* of a precedence graph G is the maximum number of pairwise incomparable tasks of G. If the width of a precedence graph G equals a constant w and the communication delays are bounded by a constant, then a minimum-length schedule for an instance $(G, m, \pi, \mathbf{1}_{V(G)}, c)$ can be constructed in $O(n^w)$ time using a dynamic-programming algorithm [29]. This dynamic-programming approach can also be used to construct schedules that are optimal with respect to other objective functions [29, 30].

The dynamic-programming approach cannot be generalised for scheduling preallocated tasks with arbitrary execution lengths. It is unlikely that a minimum-length schedule for an instance (G, m, π, μ, c), such that G is a precedence graph of constant width w, can be constructed in polynomial time: Sotskov and Shakhlevich [28] proved that constructing a minimum-length schedule for a job shop with three jobs and three processors is an NP-hard optimisation problem. However, the job shop scheduling problem with two jobs and m processors is solvable in polynomial time [5, 27].

4 Preallocated tasks without communication delays

In this section, we consider the problem of scheduling arbitrary precedence graphs with preallocated tasks without communication delays. We will study the the complexity of constructing schedules of small length. In Section 4.1, it will be proved that determining the existence of a schedule of length at most three can be done in polynomial time. If there is a schedule of length at most three, then such a schedule can be constructed in polynomial time as well.

In Section 4.2, it is proved that determining whether there is a schedule of length at most four is an NP-complete decision problem, even if the precedence constraints form a collection of fork graphs and all tasks have unit length.

4.1 Schedules of length at most three

Williamson et al. [31] proved that for the job shop scheduling problem, a schedule of length at most three can be constructed in polynomial time if such a schedule exists. For an instance of the job shop scheduling problem, they construct an instance of 2SAT that is satisfiable if and only if there is a schedule of length at most three. A truth assignment for the instance of 2SAT coincides with a schedule of length at most three. In this section, a generalisation of their algorithm for scheduling preallocated tasks without communication delays is presented.

Consider an instance $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$ for which we want to construct a schedule of length at most three. We will assume that, for all processors $p \in \{1, \ldots, m\}$,

$$\sum_{\pi(u)=p} \mu(u) \;\leq\; 3$$

and there are no tasks of length at least four. If this is not true, then there is no schedule of length at most three.

Two height functions are used to determine the possible starting times of a task in a schedule of length at most three. $h^+(u)$ is a lower bound on the schedule length for the instance $(G^+(u), m, \pi, \mu, \mathbf{0}_{E_{G^+(u)}})$, where $G^+(u)$ is the subgraph of G induced by u and its successors.

$$h^+(u) \;=\; \begin{cases} \mu(u) & \text{if } u \text{ is a sink} \\ \max\limits_{u \prec v}(h^+(v) + \mu(u)) & \text{otherwise} \end{cases}$$

Similarly, $h^-(u)$ a lower bound on the length of a feasible schedule for $(G^-(u), m, \pi, \mu, \mathbf{0}_{E_{G^-(u)}})$, where $G^-(u)$ is the subgraph of G induced by u and its predecessors.

$$h^-(u) \;=\; \begin{cases} \mu(u) & \text{if } u \text{ is a source} \\ \max\limits_{v \prec u}(h^-(v) + \mu(u)) & \text{otherwise} \end{cases}$$

Suppose there exists a schedule σ for $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$ of length three. Then for all tasks u of G, $1 \leq h^+(u), h^-(u) \leq 3$ and

$$h^-(u) - \mu(u) \;\leq\; \sigma(u) \;\leq\; 3 - h^+(u).$$

Hence we may assume that $h^-(u) - 1 \leq 3 - h^+(u)$ and that $1 \leq h^+(u), h^-(u) \leq 3$ for all tasks u of G.

Let u be a task of G, such that $h^-(u) - \mu(u) = 0$ and $3 - h^+(u) = 2$. Then u has three possible starting times in a schedule of length three. Clearly, $h^+(u) = h^-(u) = \mu(u) = 1$. Hence u is an independent task of length one. Let G^* be the subgraph of G containing the tasks v with $h^+(v) \geq 2$ or $h^-(v) \geq 2$. It is easy to see that a schedule for $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$ can be constructed from a schedule for $(G^*, m, \pi, \mu, \mathbf{0}_{E(G)})$ by scheduling the remaining tasks in idle time slots. This is possible, since we assumed that $\sum_{\pi(u)=p} \mu(u) \leq 3$ for all processors p.

Let u be a task of G^*. Let $S(u) = \{h^-(u) - 1, 3 - h^+(u)\}$ be the set of potential starting times of u is a schedule of length at most three. Note that if $S(u)$ contains two elements, then these are consecutive.

Using the sets $S(u)$, we will construct an instance of 2SAT. 2SAT is the following decision problem.

Problem. 2SAT
Instance. A set of truth variables $\{x_1, \ldots, x_n\}$ and a collection $\mathcal{C}$ of clauses containing two literals (disjunctions of two variables and negations of variables).
Question. Is there a truth assignment that satisfies every clause of $\mathcal{C}$?

The instance of 2SAT constructed using the sets $S(u)$ has variables $x_{u,t}$ for all tasks u of G^* and all times $t \in \{0, 1, 2\}$. The intended meaning is as follows: if u starts at time t, then $x_{u,t}$ is true, and if u does not start at time t, then $x_{u,t}$ is false.

A collection of clauses $\mathcal{C}$ is constructed as follows.

1. For all tasks u of G^*, add clauses

$$\bigvee_{t \in S(u)} x_{u,t} \quad \text{and} \quad \bar{x}_{u,t'}$$

for all times t', such that $t' \notin S(u)$.

2. For all tasks u of G^*, add clauses

$$\bar{x}_{u,t_1} \vee \bar{x}_{u,t_2}$$

for all times t_1 and t_2, such that $t_1 \neq t_2$.

3. For all tasks u_1 and u_2 of G^*, such that $u_1 \prec u_2$, add clauses

$$\bar{x}_{u_1,t_1} \vee \bar{x}_{u_2,t_2}$$

for all times t_1 and t_2, such that $t_2 \leq t_1 + \mu(u_1) - 1$.

4. For all tasks u_1 and u_2 of G^*, such that $\pi(u_1) = \pi(u_2)$, add clauses

$$\bar{x}_{u_1,t_1} \vee \bar{x}_{u_2,t_2}$$

for all times t_1 and t_2, such that $t_1 \leq t_2 \leq t_1 + \mu(u_1) - 1$ or $t_2 \leq t_1 \leq t_2 + \mu(u_2) - 1$.

The clauses of the first type demand that a task starts at one of its potential starting times. The clauses of the second type ensure that each task has exactly one starting time. Those of the third type state that a task must be executed after its predecessors and those of the fourth type ensure that two tasks on the same processor are not executed at the same time.

We can prove that the instance of 2SAT is satisfiable if and only if there is a feasible schedule for $(G^*, m, \pi, \mu, \mathbf{0}_{E(G)})$ of length at most three.

Lemma 4.1. *There is a feasible schedule for $(G^*, m, \pi, \mu, \mathbf{0}_{E(G)})$ of length at most three if and only if there is a truth assignment that satisfies every clause in $\mathcal{C}$.*

Proof. ($\Rightarrow$) Suppose σ is a feasible schedule for $(G^*, m, \pi, \mu, \mathbf{0}_{E(G)})$ of length at most three. Construct a truth assignment as follows. Let $x_{u,\sigma(u)}$ be true and let $x_{u,t}$ be false for all times $t \neq \sigma(u)$. It is not difficult to see that this truth assignment satisfies all clauses of $\mathcal{C}$.

($\Leftarrow$) Suppose there is a truth assignment that satisfies all clauses in $\mathcal{C}$. Construct a schedule σ for $(G^*, m, \pi, \mu, \mathbf{0}_{E(G)})$ as follows. Let u be a task of G^*. There is exactly one time $t \in \{0, 1, 2\}$, such that $x_{u,t}$ is true; let $\sigma(u) = t$ for that t. If $\sigma(u) = t$, then $t \in S(u)$. Hence the length of σ is at most three. Since the clauses of the fourth type are satisfied, no two tasks are executed at the same time on the same processor. In addition, because the clauses of the third type are satisfied, a task is scheduled after its predecessors. So σ is a feasible schedule of length at most three.

□

Determining the existence of a truth assignment that satisfies all clauses of an instance of 2SAT in a polynomial problem; and if such an assignment exists, then it can be constructed in polynomial time [9]. Hence the existence of a schedule of length at most three can be determined in polynomial time.

Theorem 4.2. *There is a polynomial-time algorithm that determines the existence of a feasible schedule of length at most three for all instances $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$; if such a schedule exists, then this algorithm constructs a feasible schedule for $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$ of length at most three.*

4.2 Schedules of length at most four

Williamson et al. [31] proved that determining the existence of a schedule of length at most four for flow shops and job shops is an NP-hard problem. Their proof also shows that verifying whether there exists a schedule of length at most four for instances $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ is NP-complete, even if G forms a collection of chains of length at most three.

In this section, I will prove the same for instances $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$, where G is a collection of fork graphs. A *fork graph* is an outtree of height one; it consist of a task, its *source*, and the children of the source, its *sinks*. A *join graph* is an fork graph in which the arcs have been reversed, an intree of height one; a join graph consists of a task, its *sink*, and the parents of the sinks, its *sources*. The NP-completeness of determining the existence of schedules of length at most four is proved using a polynomial reduction from 3SAT, which is the following decision problem.

Problem. 3SAT
Instance. A set of truth variables $\{x_1, \ldots, x_n\}$ and a collection $\mathcal{C}$ of clauses containing three literals.
Question. Is there a truth assignment that satisfies every clause of $\mathcal{C}$?

3SAT is a strongly NP-complete decision problem [10]. Define the decision problem PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS as follows.

Problem. PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS
Instance. An instance $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$, where G is a collection of fork graphs.
Question. Is there a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most four?

Using a polynomial reduction from 3SAT, we will prove that PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Lemma 4.3. *There is a polynomial reduction from* 3SAT *to* PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS.

Proof. Let $\{x_1, \ldots, x_n\}$ and $\mathcal{C}$ be an instance of 3SAT. Then $\mathcal{C}$ is a collection of clauses with three literals. Suppose $\mathcal{C} = \{C_1, \ldots, C_k\}$, such that $C_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$ for all $i \in \{1, \ldots, k\}$. Note that a literal $y_{i,j}$ is either a truth variable or the negation of a truth variable. Construct an instance $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS as follows. Let $m = n + k$. For every literal $y_{i,j}$, G contains a task $v_{i,j}$, such that $\pi(v_{i,j}) = n + i$. For each variable x_ℓ, G contains two tasks u_ℓ and $\bar{u}_\ell$, such that $\pi(u_\ell) = \pi(\bar{u}_\ell) = \ell$. If literal $y_{i,j}$ equals variable x_ℓ, then G contains an arc from u_ℓ to $v_{i,j}$. Similarly, if $y_{i,j}$ corresponds to $\bar{x}_\ell$, then G contains an arc from $\bar{u}_\ell$ to $v_{i,j}$. Now we will prove that there is a truth assignment that satisfies all clauses of $\mathcal{C}$ if and only if there is a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most four.

($\Rightarrow$) Suppose there is a truth assignment that satisfies all clauses of $\mathcal{C}$. Consider such an assignment. A schedule σ for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ can be constructed as follows. If x_ℓ is true, then let $\sigma(u_\ell) = 0$ and $\sigma(\bar{u}_\ell) = 1$. Otherwise, let $\sigma(\bar{u}_\ell) = 0$ and $\sigma(u_\ell) = 1$. Every clause C_i contains at least one literal that is satisfied. Let y_{i,j_0} be such a literal. Let $\sigma(v_{i,j_0}) = 1$ and schedule the remaining tasks $v_{i,j}$ at times 2 and 3. Obviously, σ is a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$. The last task is completed at time 4, so σ is a schedule of length at most four.

($\Leftarrow$) Let σ be a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most four. Every task $v_{i,j}$ has a predecessor. So such a task is executed at time 1, 2 or 3. For each clause C_i, there is a task $v_{i,j}$, such that $\sigma(v_{i,j}) = 1$. The predecessor of $v_{i,j}$ must be executed at time 0. Construct a truth assignment as follows. For all $\ell \in \{1, \ldots, n\}$, if $\sigma(u_\ell) = 0$, then let x_ℓ be true. Otherwise, let x_ℓ be false. Since the tasks $v_{i,j}$ that start at time 1 are successors of a task scheduled at time 0, each clause contains at least one literal that is satisfied. So there is a truth assignment that satisfies all clauses of $\mathcal{C}$.

□

Lemma 4.3 shows that PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Theorem 4.4. PREALLOCATED TASKS WITHOUT COMMUNICATION DELAYS *is a strongly* NP*-complete decision problem.*

Proof. Obvious from Lemma 4.3. □

Theorem 4.4 shows the following result.

Corollary 4.5. *Unless* P *equals* NP*, there is no polynomial-time algorithm that constructs schedules for instances* $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$*, such that* G *is a collection of fork graphs of length at most* $\frac{5}{4}$ *times the length of a minimum-length schedule for* $(G, m, \pi, \mu, \mathbf{0}_{E(G)})$*.*

Proof. Obvious from Theorem 4.4. □

5 Preallocated tasks with non-zero communication delays

In this section, we will consider the problem of scheduling preallocated tasks subject to non-zero communication delays. Consider an instance (G, m, π, μ, c), such that $c(u_1, u_2) \geq 1$ for all arcs (u_1, u_2) of G. Such an instance will be called an *instance with non-zero communication delays.* Since for each task, it is known on which processor it will be scheduled, the communication delays that occur in a feasible schedule for (G, m, π, μ, c) are known. Such a delay does not interfere with the execution of the tasks of G. So a communication delay can be viewed as a task that has to be scheduled on a separate processor.

This observation can be used to construct an instance $(G', m', \pi', \mu', \mathbf{0}_{E_{G'}})$, such that there is a one-to-one correspondence between feasible schedules for (G, m, π, μ, c) and feasible schedules for $(G', m', \pi', \mu', \mathbf{0}_{E_{G'}})$. Such an instance can be constructed as follows. For each arc (u, v) of G, such that $\pi(u) \neq \pi(v)$, a new task $c_{u,v}$ is introduced. Hence

$$V(G') \;=\; V(G) \cup \{c_{u,v} \mid (u, v) \in E(G) \wedge \pi(u) \neq \pi(v)\}.$$

The tasks $c_{u,v}$ correspond to the communication delays: a task $c_{u,v}$ has to be scheduled after task u and before task v. So

$$E_{G'} \;=\; \{(u, v) \in E(G) \mid \pi(u) = \pi(v)\} \cup \{(u, c_{u,v}), (c_{u,v}, v) \mid (u, v) \in E(G) \wedge \pi(u) \neq \pi(v)\}.$$

Let $\mu'(u) = \mu(u)$ and $\pi'(u) = \pi(u)$ for all tasks of G. Moreover, $\mu'(c_{u,v}) = c(u, v)$ and $\pi'(c_{u,v}) = m + i_{u,v}$, where $i_{u,v}$ is the index of arc (u, v) in some enumeration of $E(G) \setminus \{(u, v) \in E(G) \mid \pi(u) = \pi(v)\}$. Each communication task is executed on a separate processor, so

$$m' \;=\; m + |\{(u, v) \in E(G) \mid \pi(u) \neq \pi(v)\}|.$$

It is not difficult to see that a feasible schedule σ for (G, m, π, μ, c) can be transformed into a feasible schedule σ' for $(G', m', \pi', \mu', \mathbf{0}_{E_{G'}})$: let $\sigma'(u) = \sigma(u)$ for all tasks u of G, and $\sigma'(c_{u,v}) = \sigma(u) + \mu(u)$ for all communication tasks $c_{u,v}$. Furthermore, let σ' be a feasible schedule for $(G', m', \pi', \mu', \mathbf{0}_{E_{G'}})$. Then the restriction of σ' to $V(G)$ is a feasible schedule for (G, m, π, μ, c). Note that in both transformations, the length of the schedules remains unchanged.

Hence scheduling with non-zero communication delays is a special case of scheduling without communication delays. From Theorem 4.2, the existence of a schedule of length at most three can be determined in polynomial time. In Section 5.1, it will be proved that scheduling with non-zero communication delays is less complicated than scheduling without them: the existence of a schedule of length at most four can be determined in polynomial time. In Section 5.2, the NP-completeness of determining the existence of a schedule of length at most five is shown even if the precedence graph is a collection of fork graphs.

5.1 Schedules of length at most four

In this section, we will present a polynomial-time algorithm that checks whether there is a schedule of length at most four for instances (G, m, π, μ, c) with non-zero communication delays. If such a schedule exists, then a schedule of length at most four is constructed in polynomial time. Like in Section 4, an instance of 2SAT is constructed. This instance is proved to be satisfiable if and only if there is a feasible schedule of length at most four.

Consider an instance (G, m, π, μ, c) with non-zero communication delays. We want to construct a schedule for (G, m, π, μ, c) of length four. Hence we may assume that, for all processors $p \in \{1, \ldots, m\}$,

$$\sum_{\pi(u)=p} \mu(u) \;\leq\; 4.$$

Like in Section 4.1, we will use two height functions that are lower bounds on the length of a schedule for (G, m, π, μ, c). $h^+(u)$ is a lower bound on the schedule length for the instance $(G^+(u), m, \pi, \mu, c)$.

$$h^+(u) \;=\; \begin{cases} \mu(u) & \text{if } u \text{ is a sink} \\ \max\{\max\limits_{u \prec_0 v}(h^+(v)+\mu(u)), \max\limits_{u \prec_0 v:\pi(u)\neq\pi(v)}(h^+(v)+\mu(u)+c(u,v)), & \\ \quad \max\limits_{1\leq p\leq m}(\mu(u)+\sum\limits_{u\prec v:\pi(v)=p}\mu(v))\} & \text{otherwise} \end{cases}$$

Similarly, $h^-(u)$ is a lower bound on the length of a feasible schedule for $(G^-(u), m, \pi, \mu, c)$.

$$h^-(u) \;=\; \begin{cases} \mu(u) & \text{if } u \text{ is a source} \\ \max\{\max\limits_{v \prec_0 u}(h^-(v)+\mu(u)), \max\limits_{v \prec_0 u:\pi(u)\neq\pi(v)}(h^-(v)+\mu(u)+c(v,u)), & \\ \quad \max\limits_{1\leq p\leq m}(\mu(u)+\sum\limits_{v\prec u:\pi(v)=p}\mu(v))\} & \text{otherwise} \end{cases}$$

If for some task u of G, $h^+(u) \geq 5$ or $h^-(u) \geq 5$, then there is no schedule for (G, m, π, μ, c) of length at most four. Hence we may assume that $h^+(u), h^-(u) \leq 4$ for all tasks u of G.

If there is a feasible schedule for (G, m, π, μ, c) of length at most four, then task u starts at a time in $S(u) = \{h^-(u) - \mu(u), \ldots, 4 - h^+(u)\}$. It is not difficult to see that

$$|S(u)| \geq 3 \quad \text{if and only if} \quad h^+(u) + h^-(u) \leq 3 \;\text{ or }\; h^+(u) = h^-(u) = \mu(u) = 2.$$

Let u be a task of G. If u is an independent task of length one, then this task can be scheduled at any time t that processor $\pi(u)$ is idle. Suppose $\mu(u) = 1$ and u has a child v with $\mu(v) = 1$, $\pi(u) = \pi(v)$ and u and v do not have other children or parents. Such a chain will be called an *independent chain*. Then tasks u and v can be scheduled in any two idle time slots on processor $\pi(u)$. Hence we need not consider such tasks. Let G^* be the precedence graph obtained from G by removing all independent tasks of length one and all independent chains consisting of two tasks of length one. Then every feasible schedule for (G^*, m, π, μ, c) of length four can be extended to a feasible schedule for (G, m, π, μ, c) with the same length.

Let u be a task of G^*, such that $|S(u)| \geq 3$. Then $h^+(u) = h^-(u) = \mu(u) = 2$ or $\mu(u) = 1$ and u has a child or parent v with $\mu(v) = 1$, such that the chain containing u and v is not independent. u cannot be executed at any idle time slot on processor $\pi(u)$, but it does restrict the possible starting times of the other tasks on processor $\pi(u)$. This is expressed in the collection of clauses constructed as follows.

For all tasks u of G^* and all times $t \in \{0, 1, 2, 3\}$, a truth variable $x_{u,t}$ is introduced. The intended meaning is as follows: if $x_{u,t}$ is true, then u starts at time t; if $x_{u,t}$ is false, then u does not start at time t. Construct a collection of clauses $\mathcal{C}$ using the following rules.

1. For all tasks u of G^*, add the clauses

$$\bar{x}_{u,t}$$

for all times t, such that $t \notin S(u)$.

2. For all tasks u of G^*, add the clauses

$$\bar{x}_{u,t_1} \vee \bar{x}_{u,t_2}$$

for all times t_1 and t_2, such that $t_1 \neq t_2$.

3. For all tasks u of G^*, such that $|S(u)| \leq 2$, add the clause

$$\bigvee_{t \in S(u)} x_{u,t}.$$

4. For all tasks u_1 and u_2 of G^*, such that $u_1 \prec_0 u_2$, add the clauses

$$\bar{x}_{u_1,t_1} \vee \bar{x}_{u_2,t_2}$$

for all times t_1 and t_2, such that $t_2 \leq t_1 + \mu(u_1) - 1$. For all tasks u_1 and u_2 of G^*, such that $u_1 \prec_0 u_2$ and $\pi(u_1) \neq \pi(u_2)$, add the clauses

$$\bar{x}_{u_1,t_1} \vee \bar{x}_{u_2,t_2}$$

for all times t_1 and t_2, such that $t_2 \leq t_1 + \mu(u_1) + c(u_1, u_2) - 1$.

5. For all tasks u_1 and u_2 of G^*, such that $\pi(u_1) = \pi(u_2)$, add the clauses

$$\bar{x}_{u_1,t_1} \vee \bar{x}_{u_2,t_2}$$

for all times t_1 and t_2, such that $t_1 \leq t_2 \leq t_1 + \mu(u_1) - 1$ or $t_2 \leq t_1 \leq t_2 + \mu(u_2) - 1$.

6. For all tasks u_1 and u_2 of G^*, such that $\pi(u_1) = \pi(u_2)$ and $\mu(u_1) = \mu(u_2) = 2$, add the clauses

$$x_{u_1,0} \vee x_{u_1,2} \quad \text{and} \quad x_{u_2,0} \vee x_{u_2,2}.$$

7. For all tasks u, u_1 and u_2 of G^*, such that $\pi(u) = \pi(u_1) = \pi(u_2)$, $\mu(u) = 2$ and $\mu(u_1) = \mu(u_2) = 1$, add the clauses

$$\bar{x}_{u_1,1} \vee \bar{x}_{u_2,3}, \qquad \bar{x}_{u_1,3} \vee \bar{x}_{u_2,1}, \qquad \bar{x}_{u_1,0} \vee \bar{x}_{u_2,2},$$

$$\bar{x}_{u_1,2} \vee \bar{x}_{u_2,0}, \qquad \bar{x}_{u_1,1} \vee \bar{x}_{u_2,2} \quad \text{and} \quad \bar{x}_{u_1,2} \vee \bar{x}_{u_2,1}.$$

8. For all tasks u_1, u_2, v_1 and v_2 of G^*, such that $\pi(u_1) = \pi(u_2) = \pi(v_1) = \pi(v_2)$, $\mu(u_1) = \mu(u_2) = \mu(v_1) = \mu(v_2) = 1$ and $u_1 \prec v_1$ and $u_2 \prec v_2$, add the clauses

$$x_{u_1,0} \vee x_{u_2,0} \quad \text{and} \quad x_{v_1,3} \vee x_{v_2,3}.$$

9. For all tasks u_1, u_2 and v of G^*, such that $\pi(u_1) = \pi(u_2) = \pi(v)$, $\mu(u_1) = \mu(u_2) = 1$, $\mu(v) \leq 2$ and $|S(u_1) \cup S(u_2)| = 2$, add the clause

$$\bigvee_{t \notin S(u_1) \cup S(u_2)} x_{v,t}.$$

10. For all tasks u_1, u_2, u_3 and v of G^*, such that $\mu(u_1) = \mu(u_2) = \mu(u_3) = \mu(v) = 1$, $\pi(u_1) = \pi(u_2) = \pi(u_3) = \pi(v)$ and $|S(u_1) \cup S(u_2) \cup S(u_3)| = 3$, add the clause

$$x_{v,t},$$

for all times $t \notin S(u_1) \cup S(u_2) \cup S(u_3)$.

11. For all tasks u_1, u_2 and v of G^*, such that $\pi(u_1) = \pi(u_2) = \pi(v)$, $\mu(u_1) = \mu(u_2) = 1$, $\mu(v) = 2$ and $u_1 \prec u_2$, add the clauses

$$\bar{x_{u_1,1}}, \qquad \bar{x_{u_2,2}}, \qquad x_{u_1,0} \vee x_{v,0} \quad \text{and} \quad x_{u_2,3} \vee x_{v,2}.$$

12. For all $t \in \{0,1,2,3\}$ and all tasks u_1, u_2, v_1 and v_2 of G^*, such that $\mu(u_1) = \mu(u_2) = \mu(v_1) = \mu(v_2) = 1$, $\pi(u_1) = \pi(u_2) = \pi(v_1) = \pi(v_2)$ and $t \notin S(v_1), S(v_2)$, add the clause

$$x_{u_1,t} \vee x_{u_2,t}.$$

13. For all tasks u, v_1, v_2 and w of G^*, such that $\pi(u) = \pi(v_1) = \pi(v_2) = \pi(w)$, $\mu(u) = \mu(v_1) = \mu(v_2) = \mu(w) = 1$ and $u \prec v_1, v_2$ and $0 \notin S(w)$, add the clause

$$x_{u,0}.$$

14. For all tasks u, v_1, v_2 and w of G^*, such that $\pi(u) = \pi(v_1) = \pi(v_2) = \pi(w)$, $\mu(u) = \mu(v_1) = \mu(v_2) = \mu(w) = 1$ and $v_1, v_2 \prec u$ and $3 \notin S(w)$, add the clause

$$x_{u,3}.$$

The clauses of Rules 1, 2 and 3 state that a task u must start at a time in $S(u)$ and that each task has exactly one starting time. The clauses of Rule 4 demand that tasks are scheduled before their children and those of Rule 5 that no two tasks are executed at the same time on one processor. The clauses added using Rule 6 state that if two tasks of length two are scheduled on the same processor, then they must start at time 0 or 2. The clauses of Rule 7 ensure that a task of length two can be scheduled in two consecutive time slots. Rule 8 ensures that, if two chains of two unit-length tasks are executed at the same processor, then a source must start at time 0 and a sink at time 3. The clauses of Rule 9 state that, if two unit-length tasks have a set of two potential starting times in common, then the other tasks cannot be executed at those two times. The clauses added using Rule 10 are similar to those of Rule 9: if three tasks have three possible starting times, then a fourth task must be scheduled on the remaining starting time. The clauses added by Rule 11 demand that, if two comparable unit-length tasks and a task of length two are to be scheduled on the same processor, then either the task of length two or the source task starts at time 0, and, similarly, either the task of length two or the sink task finishes at time 4. The clauses of Rule 12 state that if there are four tasks with the same processor assignment and two of these cannot start at a time t, then one of the other two tasks must be scheduled at time t. The clauses added by Rules 13 ensure that, if there are two unit-length tasks with the same processor

assignment and one task has two successors that must be scheduled on the same processor, then this task must start at time 0 if the second task cannot be executed at that time. Rule 14 is the same as Rule 13 for the inverted schedule.

The next lemma shows that a satisfying truth assignment for $\mathcal{C}$ can be used to construct a feasible schedule for (G^*, m, π, μ, c) of length at most four.

Lemma 5.1. *There is a feasible schedule for (G^*, m, π, μ, c) of length at most four if and only if there is a truth assignment that satisfies all clauses of $\mathcal{C}$.*

Proof. ($\Rightarrow$) Let σ be a feasible schedule for (G^*, m, π, μ, c) of length at most four. Then construct a truth assignment for $\mathcal{C}$ as follows: $x_{u,\sigma(u)}$ is true and $x_{u,t}$ is false for all times $t \neq \sigma(u)$. It is not difficult to verify that this truth assignment satisfies all clauses of $\mathcal{C}$.

($\Leftarrow$) Consider a truth assignment that satisfies every clause in $\mathcal{C}$. Construct a schedule σ for (G^*, m, π, μ, c) as follows. For all tasks u and $t \in \{0, 1, 2, 3\}$, if the variable $x_{u,t}$ is true, then let $\sigma(u) = t$. These starting times do not interfere. However, not all tasks of G^* have a starting time. Let u be a task without a starting time. Then, with Rule 3, $|S(u)| \geq 3$. In that case, u is either an independent task of length two or a task of length one with a child (or parent) v with $\mu(v) = 1$ and $\pi(u) = \pi(v)$ and u has no other parents or children. These tasks can be assigned a starting time by considering every processor separately. Define $V_p = \{v \in V(G^*) \mid \pi(v) = p\}$. Let V_p' be the set of tasks of V_p without a starting time. Then $0 \leq \sum_{v \in V_p'} \mu(v) \leq 4$ and the total amount of idle time on processor p is at least $\sum_{v \in V_p'} \mu(v)$. A starting time is assigned to every task of V_p' using the following case analysis.

Case 1. $\sum_{v \in V_p'} \mu(v) = 0$.

Trivial, because $V_p' = \varnothing$.

Case 2. $\sum_{v \in V_p'} \mu(v) = 1$.

V_p' contains exactly one task u of length one. u has either a parent or a child of length one that has to be scheduled on processor p and u has no other parents or children. We will assume that u has a child v, such that $\mu(v) = 1$ and $\pi(v) = p$. Suppose u cannot be scheduled on processor p without violating the feasibility of (partial schedule) σ. Then processor p is busy at times 0, 1 and 2. v has a predecessor, so $0 \notin S(v)$. Hence v starts at time 1 or 2. Let v' be the task with $\pi(v') = p$ and $\sigma(v') + \mu(v') = 3$. Then $\mu(v') = 1$, because v is scheduled at time 1 or 2. If the tasks executed at times 0, 1 and 2 form a chain, then the total execution length of the predecessors of v' equals three. In that case, $h^-(v') = 4$ and $S(v') \subseteq \{3\}$. Contradiction. So the tasks scheduled at times 0, 1 and 2 do not form a chain.

Case 2.1. Every task in V_p has length one.

Assume $V_p = \{u, v_0, v_1, v_2\}$, such that $\sigma(v_i) = i$ for all $i \in \{0, 1, 2\}$. We know that $S(u) = \{0, 1, 2\}$ and that $S(v_0) \subseteq \{0, 1, 2\}$. So, if $S(v_1) \subseteq \{0, 1, 2\}$, then, with Rule 10, the clause $x_{v_2,3}$ is satisfied and $\sigma(v_2) = 3$. Hence we may assume that $3 \in S(v_1)$. Both u and v_0 can start at time 0. Each set $S(w)$ consists of consecutive times. No independent task is to be scheduled on processor p, so $3 \notin S(u)$ and $3 \notin S(v_0)$. Hence, with Rule 12, the clause $x_{v_1,3} \vee x_{v_2,3}$ is satisfied. So either v_1 or v_2 is scheduled at time 3. Contradiction.

Case 2.2. V_p contains a task of length at least two.

Then V_p contains one task of length two and two of length one. Assume $V_p = \{u, v, v_0\}$, such that $\mu(v_0) = 2$. From Rule 11, the clauses $x_{u,1}^-$, $x_{v,2}^-$, $x_{u,0} \vee x_{v_0,0}$ and $x_{v_0,2} \vee x_{v,3}$ are satisfied. Since v does not start at time 3, it must start at time 1. Then v_0 starts at time 2 and u can be scheduled at time 0. Contradiction.

Case 3. $\sum_{v \in V_p'} \mu(v) = 2$.

Then V_p' contains either one task of length two or two of length one.

Case 3.1. V'_p contains one task of length two.

Let u be the task in V'_p. u is an independent task of length two. From Rules 6 and 7, it is easy to see that there are at least two consecutive idle times on processor p. So u can be scheduled without violating the feasibility of σ.

Case 3.2. V'_p contains two tasks of length one.

Assume u_1 and u_2 are the tasks in V'_p. Both have either a parent or child of length one that has to be scheduled on processor p. If the chains in which u_1 and u_2 are contained are disjoint, then, with Rule 8, it is easy to see that u_1 and u_2 can be scheduled on processor p without violating the feasibility of σ.

Therefore we may assume that u_1 and u_2 have a common parent v, such that $\pi(v) = p$ and $\mu(v) = 1$. $h^+(v) \geq 3$, so $S(v) \subseteq \{0, 1\}$. Suppose u_1 and u_2 cannot be scheduled without violating the feasibility of σ. Then there is a task w of length one, such that $\pi(w) = p$ and $\sigma(w) \in \{2, 3\}$ and, moreover, $\sigma(v) = 1$. w is not an independent task, so $S(w) \subseteq \{1, 2, 3\}$. Using Rule 13, the clause $x_{v,0}$ is satisfied. In that case, $\sigma(v) = 0$. Contradiction.

In a similar way, one can prove that u_1 and u_2 cannot have a common child v, such that $\pi(v) = p$ and $\mu(v) = 1$.

Case 4. $\sum_{v \in V'_p} \mu(v) = 3$.

V'_p does not contain a task of length three, so V'_p contains either one task of length two and one of length one or three tasks of length one.

Case 4.1. V'_p contains one task of length two and one of length one.

Assume V'_p contains tasks u_1 and u_2, such that $\mu(u_1) = 1$ and $\mu(u_2) = 2$. u_1 has either a child or a parent v with $\pi(v) = p$ and $\mu(v) = 1$. We will assume that v is a parent of u_1. Using Rule 11, the clauses $x^-_{v,1}$, $x^-_{u_1}$, $x_{v,0} \vee x_{u_2,0}$ and $x_{u_2,2} \vee x_{u_1,3}$ are satisfied. So v is not scheduled at time 1. Then u_1 and u_2 can be scheduled without violating the feasibility of σ. Similarly, if v is a child of u_1, then u_1 and u_2 can be scheduled without violating the feasibility of σ.

Case 4.2. V'_p contains three tasks of length one.

Assume u_1, u_2 and u_3 are the tasks in V'_p. Each of these tasks has a child or parent v, such that $\mu(v) = 1$ and $\pi(v) = p$. Since the total execution time of the tasks of V_p is at most four, such a task v is either a common child or a common parent of u_1, u_2 and u_3. Assume v is a parent of u_1, u_2 and u_3. In that case, $h^+(v) = 4$ and $S(v) = \{0\}$. With Rule 3, $\sigma(v) = 0$. Hence u_1, u_2 and u_3 can be scheduled without violating the feasibility of σ. The case that v is a common child of u_1, u_2 and u_3 is similar.

Case 5. $\sum_{v \in V'_p} \mu(v) = 4$.

Since V'_p does not contain the tasks of independent chain of two unit-length tasks, V'_p contains two independent tasks u_1 and u_2 of length two. With Rule 6, we find that the clauses $x_{u_1,0} \vee x_{u_1,2}$ and $x_{u_2,0} \vee x_{u_2,2}$ are satisfied. From Rule 1, a task has at most one starting time. As a result, u_1 and u_2 have been assigned a starting time. Contradiction.

Hence there is a schedule for (G^*, m, π, μ, c) of length at most four. □

If there is a satisfying truth assignment for an instance of 2SAT, then such an assignment can be constructed in polynomial time [9]. So if there is a feasible schedule for (G^*, m, π, μ, c) of length at most four, then such a schedule σ can be constructed in polynomial time. The tasks of G that have no starting time in σ can be scheduled without violating the feasibility of σ or increasing its length. Hence if there is a feasible schedule for (G, m, π, μ, c) of length at most four, then such a schedule can be constructed in polynomial time.

Theorem 5.2. *There is a polynomial-time algorithm that determines the existence of a feasible schedule of length at most four for all instances (G, m, π, μ, c) with non-zero communication delays; if such a schedule exists, then this algorithm constructs a feasible schedule for (G, m, π, μ, c) of length at most four.*

5.2 Schedules of length at most five

In this section, I will prove that determining the existence of a feasible schedule of length at most five for instances (G, m, π, μ, c) with non-zero communication delays is a strongly NP-complete decision problem. This is done using the same reduction as the one presented in Section 4.2. Let PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS be the following decision problem.

Problem. PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS
Instance. An instance $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, where G is a collection of fork graphs.
Question. Is there a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most five?

Using a polynomial reduction from 3SAT, we will prove that PREALLOCATED TASKS WITH COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Lemma 5.3. *There is a polynomial reduction from* 3SAT *to* PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS.

Proof. Let $\{x_1, \ldots, x_n\}$ and $\mathcal{C}$ be an instance of 3SAT. Then $\mathcal{C}$ is a collection of clauses with three literals. Suppose $\mathcal{C} = \{C_1, \ldots, C_k\}$. Assume $C_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$ for all $i \in \{1, \ldots, k\}$. Construct an instance $(G, m, \pi, \mathbf{1}_{V(G)}, c)$ of PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS as follows. Let $m = n + k$. For every literal $y_{i,j}$, G contains a task $v_{i,j}$, such that $\pi(v_{i,j}) = n + i$. For each variable x_ℓ, G contains two tasks u_ℓ and $\bar{u}_\ell$, such that $\pi(u_\ell) = \pi(\bar{u}_\ell) = \ell$. If literal $y_{i,j}$ equals variable x_ℓ, then G contains an arc from u_ℓ to $v_{i,j}$. Similarly, if $y_{i,j}$ corresponds to $\bar{x}_\ell$, then G contains an arc from $\bar{u}_\ell$ to $v_{i,j}$. The communication requirement of each arc of G equals one. Now we will prove that there is a truth assignment that satisfies all clauses of $\mathcal{C}$ if and only if there is a schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most five.

($\Rightarrow$) Suppose there is a truth assignment that satisfies all clauses of $\mathcal{C}$. Consider such an assignment. A schedule σ for $(G, m, \pi, \mathbf{1}_{V(G)}, c)$ can be constructed as follows. If x_ℓ is true, then let $\sigma(u_\ell) = 0$ and $\sigma(\bar{u}_\ell) = 1$. Otherwise, let $\sigma(\bar{u}_\ell) = 0$ and $\sigma(u_\ell) = 1$. Every clause C_i contains at least one literal that is satisfied. Let $y_{i,j}$ be such a literal. Let $\sigma(v_{i,j}) = 2$ and schedule the remaining literal tasks $v_{i,j}$ at times 3 and 4. Clearly, σ is a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ and the length of σ is five.

($\Leftarrow$) Let σ be a feasible schedule for $(G, m, \pi, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most five. Every literal task $v_{i,j}$ has a predecessor. This predecessor is not scheduled on the same processor as $v_{i,j}$. So the literal tasks are executed at time 2, 3 or 4. For each clause C_i, there is a task $v_{i,j}$, such that $\sigma(v_{i,j}) = 2$. The predecessor of $v_{i,j}$ is executed at time 0. Construct a truth assignment as follows. If $\sigma(u_\ell) = 0$, then let x_ℓ be true. Otherwise, let x_ℓ be false. Since the tasks $v_{i,j}$ scheduled at time 2 are successors of tasks scheduled at time 0, each clause contains a literal that is satisfied. Consequently, there is a truth assignment that satisfies all clauses of $\mathcal{C}$.

□

Lemma 5.3 shows that PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS is strongly NP-complete.

Theorem 5.4. PREALLOCATED TASKS WITH UNIT COMMUNICATION DELAYS *is a strongly* NP*-complete decision problem.*

Proof. Obvious from Lemma 5.3. □

Theorem 5.4 shows the following result.

Corollary 5.5. *Unless* P *equals* NP*, there is no polynomial-time algorithm that constructs schedules for instances (G, m, π, μ, c) with non-zero communication delays, such that G is a collection of fork graphs, of length at most $\frac{6}{5}$ times the length of a minimum-length schedule for (G, m, π, μ, c).*

Proof. Obvious from Theorem 5.4. □

Theorems 4.4 and 5.2 show that scheduling preallocated tasks subject to non-zero communication delays is less complicated than the more general problem of scheduling without communication delays. This is due to the fact that, in a short schedule, the number of possible starting times of a task decreases when communication delays are introduced.

The same situation occurs in scheduling untyped task systems. Without communication delays, the existence of a schedule of length at most two for an untyped task system can be determined in polynomial time, whereas determining the existence of a schedule of length at most three is an NP-complete decision problem [20]. With communication delays, one can determine in polynomial time whether there exists a schedule of length at most three; determining the existence of a schedule of length at most four is an NP-complete decision problem [15]. In Sections 6 and 7, the same is shown for scheduling arbitrary typed task systems.

6 Typed task systems without communication delays

In this section and the next, we will study the complexity of the problem of constructing short schedules for typed instances (G, m, τ, μ, c). In this section, we will assume that the communication delays are negligible. It will be shown that scheduling typed instances is more complicated than the special case of scheduling preallocated tasks. In Section 4.1, we proved that that the existence of schedules of length at most three can be determined in polynomial time for preallocated tasks. This problem is proved to be strongly NP-complete in Section 6.2. In Section 6.1, it will be shown that the existence of a schedule of length at most two for an arbitrary typed task system can be determined in polynomial time.

6.1 Schedules of length at most two

In this section, it will be shown that determining the existence of a schedule of length at most two for typed instances $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$ is a polynomial problem. Consider an instance $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$. We may assume that there are no tasks of length three and there are no paths in G with a total task length more than two. Furthermore, we may assume that the sum of the task lengths of the tasks of type i does not exceed $2m_i$, where m_i equals the number of processors of type i. If G contains tasks of length two, then these are independent. Such a task must be scheduled on one processor of the same type as the task. If there are not enough processor of these types, then there is no schedule of length at most two.

So we may assume that G does not contain tasks of length two. We need not consider independent tasks of length one, since these can be added to a feasible schedule for the instance $(G^*, m, \tau, \mu, \mathbf{0}_{E_{G^*}})$, where G^* is the subgraph of G induced by the tasks that are not independent. So the other tasks are either sources or sinks. The sources must be scheduled at time 0, the sinks at time 1. It is easy to see that there is a schedule of length at most two if and only if, for each type i, the number of sources of type i and the number of sinks of type i does not exceed the number of processors of type i.

Theorem 6.1. *There is a polynomial-time algorithm that determines the existence of a feasible schedule of length at most two for all instances $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$; if such a schedule exists, then this algorithm constructs a feasible schedule for $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$ of length at most two.*

6.2 Schedules of length at most three

Using a polynomial reduction from CLIQUE, Lenstra and Rinnooy Kan [20] proved that determining the existence of a schedule of length at most three for an untyped instance $(G, m, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ is an NP-complete decision problem. Since scheduling untyped task systems is a special case of scheduling typed task systems, this is also true for scheduling typed instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$.

In this section, I will present a proof that determining the existence of a schedule of length at most three for an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ is an NP-complete decision problem even if G is a collection of fork graphs. This is proved using a reduction from 3SAT that is similar to the ones presented in Sections 4 and 5. Let TYPED TASKS WITHOUT COMMUNICATION DELAYS be the following decision problem.

Problem. TYPED TASKS WITHOUT COMMUNICATION DELAYS
Instance. An instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$, such that G is a collection of fork graphs.
Question. Is there a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most three?

Using a polynomial reduction from 3SAT, we will prove that TYPED TASKS WITHOUT COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Lemma 6.2. *There is a polynomial reduction from* 3SAT *to* TYPED TASKS WITHOUT COMMUNICATION DELAYS.

Proof. Let $\{x_1, \ldots, x_n\}$ and $\mathcal{C}$ be an instance of 3SAT. Then $\mathcal{C}$ is a collection of clauses with three literals. Suppose $\mathcal{C} = \{C_1, \ldots, C_k\}$. Assume $C_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$ for all $i \in \{1, \ldots, k\}$. Construct an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of TYPED TASKS WITHOUT COMMUNICATION DELAYS as follows. Let $m = n + 2k$. There are $n + k$ different types of tasks (and processors). There is one processor (processor ℓ) of type ℓ for all $\ell \in \{1, \ldots, n\}$ and two processors (processors $n + 2i - 1$ and $n + 2i$) of type $n + i$ for all $i \in \{1, \ldots, k\}$. For every literal $y_{i,j}$, G contains a task $v_{i,j}$, such that $\tau(v_{i,j}) = n + i$. For each variable x_ℓ, G contains two tasks u_ℓ and $\bar{u}_\ell$, such that $\tau(u_\ell) = \tau(\bar{u}_\ell) = \ell$. If literal $y_{i,j}$ equals variable x_ℓ, then G contains an arc from u_ℓ to $v_{i,j}$. Similarly, if $y_{i,j}$ corresponds to $\bar{x}_\ell$, then G contains an arc from $\bar{u}_\ell$ to $v_{i,j}$. The communication requirement of each arc of G equals zero. Now we will prove that there is a truth assignment that satisfies all clauses of $\mathcal{C}$ if and only if there is a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most three.

($\Rightarrow$) Suppose there is a truth assignment that satisfies all clauses of $\mathcal{C}$. Consider such an assignment. A schedule σ for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ can be constructed as follows. If x_ℓ is true, then let $\sigma(u_\ell) = 0$ and $\sigma(\bar{u}_\ell) = 1$. Otherwise, let $\sigma(\bar{u}_\ell) = 0$ and $\sigma(u_\ell) = 1$. Let $\pi(u_\ell) = \pi(\bar{u}_\ell) = \ell$. Every clause C_i contains a literal that is satisfied. Let $y_{i,j}$ be such a literal. Let $\sigma(v_{i,j}) = 1$ and $\pi(v_{i,j}) = n + 2i - 1$. Schedule the remaining literal tasks $v_{i,j}$ at time 2 on processors $n + 2i - 1$ and $n + 2i$. Clearly, (σ, π) is a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$. The length of (σ, π) is three.

($\Leftarrow$) Let (σ, π) be a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{0}_{E(G)})$ of length at most three. Every literal task $v_{i,j}$ has a predecessor that must be scheduled on a processor of a different type. So the tasks $v_{i,j}$ are executed at time 1 or 2. For each clause C_i, there is at least one task $v_{i,j}$, such that $\sigma(v_{i,j}) = 1$. The predecessor of $v_{i,j}$ must be executed at time 0. Construct a truth assignment as follows. For all variables x_ℓ, if $\sigma(u_\ell) = 0$, then let x_ℓ be true. Otherwise, let x_ℓ be false. Since the tasks $v_{i,j}$ scheduled at time 1 are successors of task scheduled at time 0, each clause contains a literal that is satisfied. So there is a truth assignment that satisfies all clauses of $\mathcal{C}$.

□

Lemma 6.2 shows that TYPED TASKS WITHOUT COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Theorem 6.3. TYPED TASKS WITHOUT COMMUNICATION DELAYS *is a strongly* NP*-complete decision problem.*

Proof. Obvious from Lemma 6.2. □

Theorem 6.3 shows the following result.

Corollary 6.4. *Unless* P *equals* NP*, there is no polynomial-time algorithm that constructs schedules for instances* $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$*, such that* G *is a collection of fork graphs, whose lengths are less than* $\frac{4}{3}$ *times the length of a minimum-length schedule for* $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$*.*

7 Typed task systems with non-zero communication delays

Similar to the construction presented in Section 5, one can show that scheduling typed tasks with non-zero communication delays is less complicated than scheduling typed task systems without communication requirements. It is, however, not a special case. The results that are proved are the same as for scheduling untyped unit-length tasks [15]: determining the existence of a schedule of length at most three is a polynomial problem, whereas determining the existence of a schedule of length at most four is an NP-complete decision problem. These results are proved in Sections 7.1 and 7.2, respectively.

7.1 Schedules of length at most three

In this section, it will be shown that the existence of a schedule of length at most three for an instance (G, m, τ, μ, c) with non-zero communication delays can be determined in polynomial time. The proof consists of two parts. First, the problem instance is divided into a number of untyped scheduling problems. For these untyped problems, the existence of a schedule of length at most three is determined.

Consider an instance (G, m, τ, μ, c) with non-zero communication delays. Suppose G contains two tasks u_1 and u_2, such that $u_1 \prec_0 u_2$ and $\tau(u_1) \neq \tau(u_2)$. If $\mu(u_1) \geq 2$, $\mu(u_2) \geq 2$ or $c(u_1, u_2) \geq 2$, then there is no schedule of length at most three. Hence we may assume that $\mu(u_1) = \mu(u_2) = c(u_1, u_2) = 1$. In a feasible schedule for (G, m, τ, μ, c) of length (at most) three, u_1 is executed at time 0 and u_2 at time 2. This knowledge can be used to remove the arc from u_1 to u_2.

A new instance $(G', m', \tau', \mu', c')$ is constructed as follows.

$$V(G') \;=\; V(G) \cup \{v_1, v_2, v_3, w_1, w_2, w_3\}$$

and

$$E(G') \;=\; (E(G) \setminus \{(u_1, u_2)\}) \cup \{(v_1, v_2), (v_2, v_3), (w_1, w_2), (w_2, w_3), (u_1, v_3), (w_1, u_2)\}.$$

The types and task lengths of the tasks in $V(G)$ remain unchanged. The same holds for the communication requirements of the arcs in $E(G) \setminus \{(u_1, u_2)\}$. The new tasks v_1, v_2 and v_3 have length one and type $\tau(u_1)$. Similarly, tasks w_1, w_2 and w_3 are of unit length and have the same type as u_2. Two processors are added, one of type $\tau(u_1)$ and one of type $\tau(u_2)$. The communication requirements of the new arcs equal one.

It is not difficult to see that a schedule of length three for (G, m, τ, μ, c) can be transformed into a schedule for $(G', m', \tau', \mu', c')$ by executing tasks v_1, v_2 and v_3 at times 0, 1 and 2 on the new processor of type $\tau(u_1)$ and tasks w_1, w_2 and w_3 at times 0, 1 and 2 on the new processor of type $\tau(u_2)$. Furthermore, if (σ, π) is a feasible schedule of length three for $(G', m', \tau', \mu', c')$, then the restriction of (σ, π) to $V(G)$ is a feasible schedule for (G, m, τ, μ, c) of the same length.

Hence we may assume that there are no arcs between tasks of different types. In that case, there is a feasible schedule of length at most three for (G, m, τ, μ, c) if and only if there are such schedules for the untyped instances (G_i, m_i, μ_i, c_i), where $V(G_i) = \{u \in V(G) \mid \tau(u) = i\}$, $E(G_i) = E(G) \cap (V(G_i) \times V(G_i))$, m_i equals the number of processors of type i, and μ_i and c_i are the restrictions of μ and c to $V(G_i)$ and $E(G_i)$, respectively. Hence we have proved the following result.

Theorem 7.1. *The existence of a schedule of length at most three for all instances* (G, m, τ, μ, c) *with non-zero communication delays can be determined in polynomial time if and only if the*

existence of a schedule of length at most three for an untyped instances (G', m', μ', c') with non-zero communication delays can be determined in polynomial time.

From now on, we will consider untyped instances. Consider an untyped instance (G, m, μ, c) with non-zero communication delays. Since we want to determine whether there exists a schedule for (G, m, μ, c) of length at most three, we may assume that the sum of the task lengths of all tasks does not exceed $3m$ and that there is no path in G with total task length at least four. Hence every task of length three is an independent task. Such a task must be scheduled at time 0 on a processor of its own. The other tasks of G have to be scheduled on the remaining $m-1$ processors. So we may assume that G does not contain any tasks of length three.

Independent tasks of length one can be scheduled in every idle time on any processor. Hence these tasks can be scheduled without violating the feasibility of a feasible partial schedule. Therefore we will assume that there are no independent tasks of length one.

Let u_1 and u_2 be two tasks of G, such that $u_1 \prec_0 u_2$, $\mu(u_1) = \mu(u_2) = 1$, $c(u_1, u_2) \geq 2$ and u_1 and u_2 do not have other children or parents. Let (σ, π) be a feasible schedule for (G, m, μ, c) of length at most three. Then $\pi(u_1) = \pi(u_2)$. Suppose $\sigma(u_1) = 0$ and $\sigma(u_2) = 2$. Due to non-zero communication delays, the idle time on processor $\pi(u_1)$ can only be filled by an independent task of length one. Since such tasks can be executed in any empty time slot, we can reschedule u_1 and u_2, such that $\sigma(u_1) = 1$ or $\sigma(u_2) = 1$. This does not violate the feasibility of (σ, π). In that case, u_1 and u_2 are executed without interruption. Hence we may replace them by a single task of length two.

Now we will assign a starting time to every task. Consider tasks u_1, u_2 and u_3 of G, such that $\mu(u_1) = \mu(u_2) = \mu(u_3) = 1$ and $u_1 \prec_0 u_2 \prec_0 u_3$. Then let $\sigma(u_1) = 0$, $\sigma(u_2) = 1$ and $\sigma(u_3) = 2$. In a feasible schedule for (G, m, μ, c) of length at most three, u_1, u_2 and u_2 must be scheduled on the same processor. If u_1 has other children $v_1, \ldots, v_k$, then these must be scheduled at time 2. Similarly, if u_3 has other parents $w_i, \ldots, w_\ell$, then these are to start at time 0. Note that there is no schedule of length at most three if $\mu(v_i) \geq 2$ or $\mu(w_i) \geq 2$ for some $i \in \{1, \ldots, k\}$ or $c(u_1, v_i) \geq 2$ or $c(w_i, u_3) \geq 2$ for some $i \in \{1, \ldots, \ell\}$. Moreover, the tasks v_i must be sinks of G and the tasks w_i sources of G. The other children of u_1 and the other parents of u_3 may be scheduled on any idle processor.

Next consider tasks u_1 and u_2 of G, such that $\mu(u_1) = 1$, $\mu(u_2) = 2$ and $u_1 \prec_0 u_2$. Then let $\sigma(u_1) = 0$ and $\sigma(u_2) = 1$. These tasks must be scheduled on the same processor. If u_1 has other children, then these must start at time 2. Such a child w of u_1 must be a sink of G of length one and $c(u_1, w)$ may not exceed 1. u_2 may not have other parents or children. In a schedule of length three, the children of u_1 can be scheduled on any processor.

A similar construction can be used to assign starting times to tasks v_1 and v_2 of G, such that $\mu(v_1) = 2$, $\mu(v_2) = 1$ and $v_1 \prec_0 v_2$: v_1 starts at time 0 and v_2 at time 2. A parent $w \neq v_1$ of v_2 must be a source of unit length, such that $c(w, v_2) = 1$. w must start at time 0 and can be executed on any idle processor.

Let u_1 and u_2 be two unit-length of G tasks that have not been assigned a starting time. Suppose u_2 is a child of u_1 and $c(u_1, u_2) \geq 2$. We may assume that u_1 has other children or u_2 has other parents. Suppose the tasks $v_1, \ldots, v_k$ are the other children of u_1 and $w_1, \ldots, w_\ell$ the other parents of u_2. If $\sum_{i=1}^{k} c(u_1, v_i) + \sum_{i=1}^{\ell} c(w_i, u_2) \geq k + \ell + 2$, then there is no schedule for (G, m, μ, c) of length at most three. Hence we may assume that $\sum_{i=1}^{k} c(u_1, v_i) + \sum_{i=1}^{\ell} c(w_i, u_2)$ is at most $k + \ell + 1$. If $k \geq 1$, then let $\sigma(u_1) = 0$. If $\ell \geq 1$, then u_2 must start at time 2.

Assume $k, \ell \geq 1$. If $c(u_1, v_i) \geq 2$, then v_i starts at time 1 and the other children of u_1 at time 2. Similarly, if $c(w_i, u_2) \geq 2$, then w_i is to be scheduled at time 1 and the other children of u_2 at time 0. If $c(u_1, v_i)$ and $c(w_i, u_2)$ do not exceed 1, then let $\sigma(v_i) = 2$ and $\sigma(w_i) = 0$ for all $i \geq 2$. The starting times of v_1 and w_1 remain unknown.

Suppose u_1 is the only parent of u_2. Then the only tasks that can start at time 1 on the same processor as u_1 is either a child of u_1 or an independent task. So we may assume that $\sigma(u_2) = 1$

and that $\sigma(v_i) = 2$ for all $i \in \{1, \ldots, k\}$. Similarly, suppose u_1 does not have other children. Then let $\sigma(u_1) = 1$ and $\sigma(w_i) = 0$ for all $i \in \{1, \ldots, \ell\}$.

In all cases, we may assume that, in a feasible schedule of length three, u_1, u_2 and a child v_i of u_1 or a parent w_i of u_2 are scheduled on one processor.

Consider a unit-length task u_1 with $k \geq 2$ children $v_1, \ldots, v_k$, such that $c(u_1, v_i) = 1$ for all $i \in \{1, \ldots, k\}$. We may assume that u_1 has not been assigned a starting time. In a feasible schedule for (G, m, μ, c), at most one successor of u_1 can start immediately after u_1. So $\sigma(u_1) = 0$ and $\sigma(v_i) = 2$ for all $i \in \{2, \ldots, k\}$. Note that all children of u_1 must be sinks of G of length one. The children $v_2, \ldots, v_k$ can be executed on any idle processor. If v_1 were to start at time 1, then it must be scheduled on the same processor as u_1.

Similarly, let u_2 be a task of length one without a starting time. Suppose u_2 has $\ell \geq 2$ parents $w_1, \ldots, w_\ell$, such that $c(w_i, u_2) = 1$ for all $i \in \{1, \ldots, \ell\}$. We may assume that each task w_i is a source of G of unit length. Let $\sigma(u_2) = 2$ and $\sigma(w_i) = 0$ for all $i \in \{2, \ldots, \ell\}$. Tasks $w_2, \ldots, w_\ell$ can be scheduled on any processor that is idle at time 0. If w_1 starts at time 1, then it must be scheduled on the same processor as u_2.

Now consider the tasks that have not been assigned a starting time. The independent tasks of length two do not have a starting time. The same holds for unit-length tasks u_1 and u_2 of G, such that $u_1 \prec_0 u_2$, $c(u_1, u_2) = 1$ and u_1 and u_2 do no have other parents or children. Furthermore, there are sinks of fork graphs and sources of join graphs without a starting time. In addition, for every pair of unit-length tasks u_1 and u_2 of G, such that $u_1 \prec_0 u_2$ and $c(u_1, u_2) \geq 2$, there may be two tasks v_1 and v_2 of G of unit length, such that $u_1 \prec_0 u_2$, $u_1 \prec_0 v_1$, $v_2 \prec_0 u_2$, $c(u_1, u_2) \geq 2$, $c(u_1, v_1) = c(v_2, u_2) = 1$ and v_1 and v_2 do not have a starting time.

First, we will consider the last type of tasks. Let u_1, u_2, v_1 and v_2 be unit-length tasks of G, such that $u_1 \prec_0 u_2$, $u_1 \prec_0 v_1$, $v_2 \prec_0 u_2$, $c(u_1, u_2) \geq 2$ and $c(u_1, v_1) = c(v_2, u_2) = 1$. In a feasible schedule (σ, π) of length at most three, $\sigma(u_1) = 0$, $\sigma(u_2) = 2$ and $\pi(u_1) = \pi(u_2)$. The only tasks (without a starting time) that can be executed at time 1 on the same processor as u_1 and u_2 are v_1, v_2 and independent tasks of length one. Since we assumed that there are no independent tasks of length one, we may assume that either $\sigma(v_1) = 1$, $\sigma(v_2) = 0$ and $\pi(v_1) = \pi(u_1)$, or $\sigma(v_1) = 2$, $\sigma(v_2) = 1$ and $\pi(v_2) = \pi(u_1)$.

Hence the tasks of such quadruples (u_1, u_2, v_1, v_2) can be scheduled in two possible ways: one processor is completely filled and one task starts at time 0 or 2. This task can be scheduled on any empty processor. So the total number of way to schedule the tasks of the quadruples (u_1, u_2, v_1, v_2) is bounded by n. Such a possibility is described by the number of tasks v_1 that start at time 2 (and the number of tasks v_2 that start at time 0).

To determine the existence of a schedule of length three, we have to consider all possibilities of scheduling the tasks of the quadruples (u_1, u_2, v_1, v_2). There is a feasible schedule of length at most three if and only if one of the possible partial schedules can be extended to a feasible schedule of length at most three.

So we will to consider one of the ways in which the quadruples (u_1, u_2, v_1, v_2) can be scheduled. A number of starting times is known. If the starting time of u_1 and u_2 is known and u_1 and u_2 should be executed on the same processor, then the total task length of the tasks that should be executed on this processor equals three. Consequently, we can ignore this processor and the tasks that must be executed on it.

Suppose m' processors are not completely filled. There are tasks with starting time 0 and 2 that can be scheduled on any of the m' empty processors. There are no tasks that start at time 1 that need not be executed on a full processor. In addition, the tasks without a starting time are not comparable with a tasks on a processor that is completely filled.

Now we will divide the tasks without a starting time in four sets. V_1 contains the independent tasks of length two. The elements of V_2 are the unit-length tasks u_1 and u_2, such that $u_1 \prec_0 u_2$ and u_1 and u_2 do not have other children or parents. V_3 is the set of sinks of fork graphs that do not have a starting time. The sources of join graph without a starting time are contained in V_4.

Note that the tasks in V_2, V_3 and V_4 are all of unit length.

In a schedule of length three, a task of V_1 has starting time 0 or 1. Since there is no difference between the tasks of V_1, there are at most $|V_1| + 1 = O(n)$ way in which the independent tasks of length two can be scheduled. Let u_1 and u_2 be two tasks of V_2, such that $u_1 \prec_0 u_2$. There are three ways in which u_1 and u_2 can be executed in a schedule (σ, π) of length three: $\sigma(u_1) = 0$ and $\sigma(u_2) = 1$, $\sigma(u_1) = 0$ and $\sigma(u_2) = 2$ or $\sigma(u_1) = 1$ and $\sigma(u_2) = 2$. The first and the last possibility require that u_1 and u_2 are scheduled on the same processor. Since the tasks in V_2 form independent chains, there are $O(n^2)$ different ways in which the tasks of V_2 can be scheduled.

Consider a task u of V_3. u must be scheduled at time 1 or 2. If it starts at time 1, then it must be scheduled on the same processor as its parent, which has been assigned starting time 0. At time 2, it can be scheduled on any empty processor. For every task v_1 with starting time 0 that is not scheduled on a full processor, there is exactly one child v_2 in V_3. The pairs (v_1, v_2) are all similar, so there are at most $|V_3| + 1 = O(n)$ ways in which the tasks of V_3 can be scheduled.

Similarly, there are at most $|V_4| + 1 = O(n)$ possibilities of scheduling the tasks of V_4. These tasks must start at time 0 or 1. If they are executed at time 1, then they must be scheduled on the same processor as their children that are assigned starting time 2.

Hence there are $O(n^5)$ ways to assign starting times to the tasks of V_1, V_2, V_3 and V_4. Such a possible assignment can be described by a tuple $(n_{1,0}, n_{1,1}, n_{2,0,1}, n_{2,0,2}, n_{2,1,2}, n_{3,1}, n_{3,2}, n_{4,0}, n_{4,1})$, where $n_{i,t}$ denotes the number of tasks of V_i that start at time t and n_{2,t_1,t_2} equals the number of pairs of tasks (u_1, u_2) of V_2, such that $u_1 \prec_0 u_2$, u_1 starts at time t_1 and u_2 at time t_2. For each assignment of starting times σ, we will try to construct an assignment of processors π, such that (σ, π) is a feasible schedule of length at most three. In order to this, we only need to consider the tasks that are not executed on a processor that is completely filled.

So, given an assignment of starting times σ, we have to determine whether there exists a schedule (σ, π) of length at most three. Note that we may assume that $\sigma(u) + \mu(u) \leq 3$ for all tasks u. π must schedule all tasks on m' available processors.

First, consider the tasks of length two and all unit-length tasks u_1 and u_2, such that $\sigma(u_2) = \sigma(u_1) + 1$. These have to be scheduled on separate processors. If the number of available processors is not sufficient, then there is no schedule for (G, m, μ, c) of length three. Second, consider the remaining tasks. The tasks that have not been scheduled on a processor are unit-length tasks with starting time 0 or 2. Such a task u can be executed by any processor that is idle at time $\sigma(u)$. So there is a feasible schedule (σ, π) if and only if the number of processors that are empty at times 0 and 2 is sufficient to schedule the remaining tasks with starting time 0 and 2.

Given an instance (G, m, τ, μ, c), we can determine the existence of a schedule for (G, m, τ, μ, c) of length at most three is polynomial time. First, the instance is transformed into $O(m)$ untyped instances $(G_i, m_i, \tau_i, \mu_i, c_i)$. For each of these instances, $O(n^6)$ possible assignments of starting times are constructed. For each assignment of starting times σ, it can be determined in polynomial time, whether there is an assignment of processors π, such that (σ, π) is a feasible schedule for the untyped problem instance. If for each untyped instance $(G_i, m_i, \tau_i, \mu_i, c_i)$, there is a feasible schedule (σ_i, π_i) of length at most three, then there is a feasible schedule (σ, π) for the typed instance (G, m, τ, μ, c). Hence we have proved the following result.

Theorem 7.2. *There is a polynomial-time algorithm that determines the existence of a feasible schedule of length at most three for all instances (G, m, τ, μ, c) with non-zero communication delays; if such a schedule exists, then this algorithm constructs a feasible schedule for (G, m, τ, μ, c) of length at most three.*

7.2 Schedules of length at most four

In this section, I will prove that determining the existence of a feasible schedule of length at most four for typed instances (G, m, τ, μ, c) with non-zero communication delays is a strongly NP-complete decision problem. This was already proved by Hoogeveen et al. [15] for the special case

of untyped instances with unit-length tasks and unit-length communication delays.

The proof presented in this section is less complicated than the proof of Hoogeveen et al. [15]. Using a reduction reduction similar to the one presented in Section 6.2, it is shown that determining the existence of a schedule of length at most four for instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ is NP-complete even if G is a collection of fork graphs and all tasks and communication delays are of unit length. Let TYPED TASKS WITH UNIT COMMUNICATION DELAYS be the following decision problem.

Problem. TYPED TASKS WITH UNIT COMMUNICATION DELAYS
Instance. An instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, where G is a collection of fork graphs.
Question. Is there a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most four?

A polynomial reduction from 3SAT is used to prove the NP-completeness of TYPED TASKS WITH UNIT COMMUNICATION DELAYS.

Lemma 7.3. *There is a polynomial reduction from* 3SAT *to* TYPED TASKS WITH UNIT COMMUNICATION DELAYS.

Proof. Let $\{x_1, \ldots, x_n\}$ and $\mathcal{C}$ be an instance of 3SAT. Then $\mathcal{C}$ is a collection of clauses with three literals. Suppose $\mathcal{C} = \{C_1, \ldots, C_k\}$. Assume $C_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$ for all $i \in \{1, \ldots, k\}$. Construct an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of TYPED TASKS WITH UNIT COMMUNICATION DELAYS as follows. Let $m = n + 2k$. There are $n + k$ different types of tasks (and processors). There is one processor (processor ℓ) of type ℓ for all $\ell \in \{1, \ldots, n\}$ and two processors (processors $n+2i-1$ and $n+2i$) of type $n+i$ for all $i \in \{1, \ldots, k\}$. For every literal $y_{i,j}$, G contains a task $v_{i,j}$, such that $\tau(v_{i,j}) = n + i$. For each variable x_ℓ, G contains two tasks u_ℓ and $\bar{u}_\ell$, such that $\tau(u_\ell) = \tau(\bar{u}_\ell) = \ell$. If literal $y_{i,j}$ equals variable x_ℓ, then G contains an arc from u_ℓ to $v_{i,j}$. Similarly, if $y_{i,j}$ corresponds to $\bar{x}_\ell$, then G contains an arc from $\bar{u}_\ell$ to $v_{i,j}$. The communication requirement of each arc of G equals one. Now we will prove that there is a truth assignment that satisfies all clauses in $\mathcal{C}$ if and only if there is a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most four.

($\Rightarrow$) Suppose there is a truth assignment that satisfies all clauses of $\mathcal{C}$. Consider such an assignment. A schedule σ for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ can be constructed as follows. If x_ℓ is true, then let $\sigma(u_\ell) = 0$ and $\sigma(\bar{u}_\ell) = 1$. Otherwise, let $\sigma(\bar{u}_\ell) = 0$ and $\sigma(u_\ell) = 1$. Obviously, $\pi(u_\ell) = \pi(\bar{u}_\ell) = \ell$. Every clause C_i contains at least one literal that is satisfied. Let $y_{i,j}$ be such a literal. Let $\sigma(v_{i,j}) = 2$ and $\pi(v_{i,j}) = n + 2i - 1$. Schedule the remaining literal tasks $v_{i,j}$ at time 3 on processors $n + 2i - 1$ and $n + 2i$. Clearly, (σ, π) is a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$. The length of (σ, π) equals four.

($\Leftarrow$) Let (σ, π) be a feasible schedule for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of length at most four. Every literal task $v_{i,j}$ has a predecessor that must be scheduled on a processor of a different type. So the tasks $v_{i,j}$ are executed at time 2 or 3. For each clause C_i, there is at least one task $v_{i,j}$, such that $\sigma(v_{i,j}) = 2$. The predecessor of $v_{i,j}$ is executed at time 0. Construct a truth assignment as follows. If $\sigma(u_\ell) = 0$, then let x_ℓ be true. Otherwise, let x_ℓ be false. Since the tasks $v_{i,j}$ scheduled at time 2 are successors of task scheduled at time 0, each clause contains a literal that is satisfied. So there is a truth assignment that satisfies all clauses of $\mathcal{C}$.

□

Lemma 7.3 shows that TYPED TASKS WITH COMMUNICATION DELAYS is a strongly NP-complete decision problem.

Theorem 7.4. TYPED TASKS WITH UNIT COMMUNICATION DELAYS *is a strongly* NP*-complete decision problem.*

Proof. Obvious from Lemma 7.3. □

Theorem 7.4 shows the following result. Hoogeveen et al. [15] proved the same for untyped instances with unit-length tasks and unit-length communication requirements.

Corollary 7.5. *Unless* P *equals* NP*, there is no polynomial-time algorithm that constructs schedules for instances* (G, m, τ, μ, c)*, such that* G *is a collection of fork graphs, whose lengths are less than* $\frac{5}{4}$ *times the length of a minimum-length schedule for* (G, m, τ, μ, c)*.*

8 Typed interval-ordered tasks

In this last section, we deal with a special type of precedence graphs, the interval orders. In Section 8.1, I will present an algorithm that constructs minimum-length schedules for typed instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order. In Section 8.2, it is proved that minimising the makespan for instances (G, m, π, μ, c) is NP-hard even if G is an interval order.

Papadimitriou and Yannakakis [22] defined interval orders as follows. An interval order is a partial order $(V, \prec)$, for which every element v of V can be assigned a closed interval $I(v)$ in the real line such that for all elements v_1 and v_2 of V,

$$v_1 \prec v_2 \quad \text{if and only if} \quad x < y \text{ for all } x \in I(v_1), y \in I(v_2).$$

Interval orders have a very nice property: the sets of successors of the tasks of an interval order form a total order. If u_1 and u_2 are two tasks of an interval order, then

$$Succ(u_1) \subseteq Succ(u_2) \quad \text{or} \quad Succ(u_2) \subseteq Succ(u_1),$$

where $Succ(u)$ denotes the set of successors of u.

8.1 Unit-length tasks

In this section, we will consider the problem of scheduling instances $(G, m, \tau, \mathbf{1}_{V(G)}, c)$, such that G is an interval order. Jansen [17, 18] proved that without communication delays, a minimum-length schedule can be constructed in polynomial time. Kellerer and Woeginger [19] showed that if each task has an arbitrary set of processors on which it can be scheduled, then a minimum-length schedule on two processors for interval orders can be constructed in polynomial time.

In this section, an algorithm will be presented that constructs minimum-length schedules for instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order. This algorithm is a generalisation of the algorithm presented by Ali and El-Rewini [1] that minimises the makespan for untyped instances $(G, m, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order.

Algorithm TYPED INTERVAL ORDER SCHEDULING is shown in Figure 1. It only constructs an assignment of starting times. An assignment of starting times for an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ is called a *feasible assignment of starting times* if, for all tasks $u_1 \neq u_2$ of G, all times t and all types i,

1. $|\{u \in V(G) \mid \sigma(u) = t \wedge \tau(u) = i\}| \leq m_i$;
2. if $u_1 \prec u_2$, then $\sigma(u_2) \geq \sigma(u_1) + 1$;
3. if $u_1 \prec u_2$ and $\tau(u_1) \neq \tau(u_2)$, then $\sigma(u_2) \geq \sigma(u_1) + 2$;
4. there is at most one child v of u_1, such that $\sigma(v) = \sigma(u_1) + 1$; and
5. there is at most one parent v of u_1, such that $\sigma(v) = \sigma(u_1) - 1$.

The first constraint ensures that the number of tasks of the same type that are scheduled at the same time does not exceed the number of processors of that type. The second constraint states that a task must be scheduled after its predecessors. The third constraint ensures that there is a delay of at least one time unit between two comparable tasks of different types; there should be

such a delay, because such tasks are not scheduled on the same processor. The fourth and fifth constraint state that at most one child and at most one parent of a task are scheduled without interruption.

Obviously, every feasible schedule (σ, π) for an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ implies a feasible assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$. In addition, it is not difficult to show that given a feasible assignment of starting time σ, a feasible schedule (σ, π) can be constructed in polynomial time.

The following notation is used. $N(t, i)$ denotes the number of tasks of type i that start at time t. $ready(u)$ equals the earliest time at which u becomes available for scheduling. $last(u)$ is a parent of u with maximum starting time. $last2(u) \neq last(u)$ is a parent of u with maximum starting time among the parents of u except $last(u)$. These parents may not exist; in that case, they represent empty tasks. We will assume that the starting time of an empty task equals $-\infty$.

Algorithm TYPED INTERVAL ORDER SCHEDULING
Input. An instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order, and a list $(u_1, \ldots, u_n)$ containing all tasks of G, such that $Succ(u_1) \supseteq \cdots \supseteq Succ(u_n)$.
Output. A feasible assignment of starting times σ for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$.

1. **for** $t := 0$ **to** $n - 1$
2. **do for** all types i
3. **do** $N(t, i) := 0$
4. **for** $i := 1$ **to** n
5. **do** $last(u_i) := \varnothing$
6. $last2(u_i) := \varnothing$
7. **for** all parents v of u_i
8. **do if** $\sigma(v) > \sigma(last(u_i))$ **or** $\sigma(v) = \sigma(last(u_i))$ **and** $\tau(v) \neq \tau(u_i)$
9. **then** $last2(u_i) := last(u_i)$
10. $last(u_i) := v$
11. **else if** $\sigma(v) > \sigma(last2(u_i))$
12. **then** $last2(u_i) := v$
13. **if** $\tau(last(u_i)) \neq \tau(u_i)$ **or** $\sigma(last(u)) = \sigma(last2(u))$
14. **then** $ready(u_i) := \sigma(last(u_i)) + 2$
15. **else if** there is a child w of $last(u_i)$, such that $\sigma(w) = \sigma(last(u_i)) + 1$
16. **then** $ready(u_i) := \sigma(last(u_i)) + 2$
17. **else** $ready(u_i) := \sigma(last(u_i)) + 1$
18. $\sigma(u_i) := \min\{t \geq ready(u_i) \mid N(t, \tau(u_i)) < m_i\}$
19. $N(\sigma(u_i), \tau(u_i)) := N(\sigma(u_i), \tau(u_i)) + 1$

Figure 1. Algorithm TYPED INTERVAL ORDER SCHEDULING

Consider an instance $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order. Let σ be the assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ constructed by Algorithm TYPED INTERVAL ORDER SCHEDULING. It is not difficult to see that a task u of type i is scheduled at a time at which at most $m_i - 1$ tasks of type i are scheduled and no precedence constraint or communication requirement is violated. So σ is a feasible assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$.

The time complexity of Algorithm TYPED INTERVAL ORDER SCHEDULING can be determined as follows. First we will compute the transitive closure of G. A topological order of G can be constructed in $O(n + e)$ time [8]. Using a topological order of an interval order G, the set of successors of each task can be computed. Assume $u_1, \ldots, u_n$ is a topological order of G. Assume $Succ(u_{i+1}), \ldots, Succ(u_n)$ have been computed. Let $v_1, \ldots, v_k$ be the children of u_i with $Succ(v_1) \subseteq \cdots \subseteq Succ(v_k)$. Then $Succ(u_i) = Succ(v_k) \cup \{v_1, \ldots, v_k\}$. For every task v in $Succ(u)$, add an arc from u to v to G. The resulting precedence graph is the transitive closure of G. It is

constructed in $O(n + e^+)$ time, where e^+ is the number of arcs in the transitive closure of G.

Using the transitive closure of G, the tasks of G can be ordered by non-increasing number of successors in $O(n)$ time using Counting sort [8].

For each task u of G, $ready(u)$ is computed by traversing all parents of u. Hence the parents $last(u)$ and $last2(u)$ are computed in $O(indegree(u))$ time. Then $ready(u)$ is determined in $O(outdegree(last(u))$ time by checking all children of $last(u)$. Hence, throughout the execution of the algorithm, $O(e)$ time used to determine the ready time of a task. The computation of the starting time of u takes $O(n)$ for each task u, so $O(n^2)$ time in total.

Lemma 8.1. *For all instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order, Algorithm* TYPED INTERVAL ORDER SCHEDULING *constructs a feasible assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ in $O(n^2)$ time.*

With an exchange argument, we can show that Algorithm TYPED INTERVAL ORDER SCHEDULING constructs assignments of starting times that correspond to minimum-length schedules for typed interval orders with unit-length tasks. Let the *length* of an assignment of starting times be the maximum completion time of a task.

Theorem 8.2. *Algorithm* TYPED INTERVAL ORDER SCHEDULING *constructs assignments of starting times of minimum length for all instances $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that G is an interval order.*

Proof. Let $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ be a typed instance, such that G is an interval order. Let σ be the assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ constructed by Algorithm TYPED INTERVAL ORDER SCHEDULING. We will prove by induction that for all $j \in \{1, \ldots, n\}$, there is a minimum-length assignment of starting times σ_j for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that $\sigma(u_i) = \sigma_j(u_i)$ for all $i \in \{1, \ldots, j\}$. Obviously, there is a minimum-length assignment of starting times σ_0 for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that $\sigma(u_i) = \sigma_0(u_i)$ for all $i \in \{1, \ldots, 0\} = \varnothing$.

Assume by induction that there is a minimum-length assignment of starting times σ_k for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$, such that $\sigma(u_i) = \sigma_k(u_i)$ for all $i \in \{1, \ldots, k\}$. Consider u_{k+1}. We may assume that $\sigma(u_{k+1}) \neq \sigma_k(u_{k+1})$. Then $\sigma_k(u_{k+1}) > \sigma(u_{k+1})$, otherwise, u_{k+1} was ready at an earlier time and $\sigma(u_{k+1})$ would be smaller. We may assume that there is a task v, such that $\sigma_k(v) = \sigma(u_{k+1})$, $\tau(v) = \tau(u_{k+1})$ and v is not an element of $\{u_1, \ldots, u_k\}$. Then $Succ(v) \subseteq Succ(u_{k+1})$. There is at most one successor w of u_{k+1}, such that $\sigma_k(w) = \sigma_k(u_{k+1}) + 1$. Such a successor has the same type as v and u_{k+1}. Since v is ready at time $\sigma_k(v)$, it can also be scheduled at time $\sigma_k(u_{k+1})$. So v and u_{k+1} can be exchanged. This gives a feasible assignment of starting times σ', such that $\sigma'(u_i) = \sigma(u_i)$ for all $i \in \{1, \ldots, k+1\}$. Moreover, the length of σ' is the same as that of the minimum-length assignment σ_k.

By induction, there is an assignment of starting times σ_n for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$ of minimum length, such that $\sigma(u_i) = \sigma_n(u_i)$ for all $i \in \{1, \ldots, n\}$. As a result, σ is a minimum-length assignment of starting times for $(G, m, \tau, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$. □

So, for interval orders, constructing a minimum-length schedule for typed instances with unit-length tasks and unit-length communication delays takes polynomial time. If the communication delays are not of unit length, then the problem becomes more complicated: Schäffter [24] proved that minimising the makespan for untyped instances $(G, \infty, \mathbf{1}_{V(G)}, c)$, such that G is an interval order and $c : E(G) \to \{0, 1\}$, is an NP-hard optimisation problem. He proved this using a polynomial reduction similar to the one presented by Hoogeveen et al. [15] used for proving the NP-hardness of constructing minimum-length schedules for untyped instances $(G, \infty, \mathbf{1}_{V(G)}, \mathbf{1}_{E(G)})$.

8.2 Arbitrary task lengths

In this section, I will show that minimising the makespan for preallocated interval-ordered tasks with arbitrary lengths and unit-length communication delays is an NP-hard problem. This is proved using a polynomial reduction from PARTITION [10], which is the following well-known NP-complete decision problem [10].

Problem. PARTITION
Instance. A set of positive integers $A = \{a_1, \ldots, a_k\}$.
Question. Is there a subset A' of A, such that $\sum_{a \in A'} a = \sum_{a \notin A'} a$?

PREALLOCATED INTERVAL-ORDERED TASKS is the following decision problem.

Problem. PREALLOCATED INTERVAL-ORDERED TASKS
Instance. An instance $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$, such that G is an interval orders and a positive integer D.
Question. Is there a feasible schedule for $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$ of length at most D?

The next lemma shows a simple polynomial reduction from PARTITION to PREALLOCATED INTERVAL-ORDERED TASKS.

Lemma 8.3. *There is a polynomial reduction from* PARTITION *to* PREALLOCATED INTERVAL-ORDERED TASKS.

Proof. Let $A = \{a_1, \ldots, a_k\}$ be an instance of PARTITION. Define $B = \frac{1}{2}\sum_{i=1}^{k} a_i$. Construct an instance $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$ of PREALLOCATED INTERVAL-ORDERED TASKS as follows. For each a_i, G contains an independent task u_i, such that $\mu(u_i) = a_i$ and $\pi(u_i) = 2$. In addition, G contains four tasks v_1, v_2, v_3 and w, such that $v_1 \prec_0 v_2 \prec_0 v_3$ and $v_1 \prec_0 w \prec_0 v_3$. These have the following lengths: $\mu(v_1) = \mu(v_3) = B - 1$, $\mu(v_2) = 3$ and $\mu(w) = 1$. The communication requirements of the arcs equals one. Let $\pi(v_1) = \pi(v_2) = \pi(v_3) = 1$ and $\pi(w) = 2$. Let $D = 2B + 1$. Now we will prove there is a subset A' of A, such that $\sum_{a \in A_1} a = \frac{1}{2}B$, if and only if there is a feasible schedule for $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$ of length at most D.

($\Rightarrow$) Let A_1 be a subset of A, such that $\sum_{a \in A_1} a = \frac{1}{2}B$. Let $A_2 = A \setminus A_1$. Construct a schedule σ for $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$ as follows. Let $\sigma(v_1) = 0$, $\sigma(v_2) = B - 1$, $\sigma(v_3) = B + 2$ and $\sigma(w) = B$. Moreover, for all tasks u_i in A_1, let

$$\sigma(u_i) \;=\; \sum_{j < i : u_j \in A_1} a_j$$

and, for all u_i in A_2,

$$\sigma(u_i) \;=\; B + 1 + \sum_{j < i : u_j \in A_1} a_j.$$

Then σ is a feasible schedule for $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$. The last task of G is completed at time $2B + 1 = D$.

($\Leftarrow$) Let σ be a schedule for $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$ of length at most D. There is only one way in which tasks v_1, v_2, v_3 and w are scheduled: $\sigma(v_1) = 0$, $\sigma(v_2) = B - 1$, $\sigma(v_3) = B + 2$ and $\sigma(w) = B$. The remaining tasks are executed at times $0, \ldots, B - 1$ and $B + 1, \ldots, 2B$ on processor 2. Define $A_1 = \{a_i \mid \sigma(u_i) \leq B - 1\}$ and $A_2 = \{a_i \mid \sigma(u_i) \geq B + 1\}$. Since the total task length equals $2B + 2$, no processor is idle before time D. Because a task cannot be preempted,

$$\sum_{a \in A_1} a \;=\; \sum_{i : \sigma(u_i) \leq B-1} \mu(u_i) \;=\; B.$$

□

Lemma 8.3 shows that PREALLOCATED INTERVAL-ORDERED TASKS is an NP-complete decision problem and that constructing minimum-length schedules for preallocated interval-ordered tasks is an NP-hard optimisation problem.

Theorem 8.4. *Constructing minimum-length schedules for instances* $(G, 2, \pi, \mu, \mathbf{1}_{E(G)})$*, such that* G *is an interval order, is an* NP*-hard optimisation problem.*

Proof. Obvious from Lemma 8.3. □

Clearly, the same can be proved for scheduling preallocated interval-ordered tasks without communication delays.

Theorem 8.5. *Constructing minimum-length schedules for instances* $(G, 2, \pi, \mu, \mathbf{0}_{E(G)})$*, such that* G *is an interval order, is an* NP*-hard optimisation problem.*

9 Concluding remarks

In this report, I have shown that, if the existence of short schedules is difficult to be determined, then scheduling with non-zero communication delays is less complicated than scheduling without communication delays. Hoogeveen et al. [15] proved this as well.

The communication delays restrict the number of possible starting times in a short feasible schedule: every potential starting time of a task u in a schedule of length ℓ for an instance (G, m, τ, μ, c) with (non-zero) communication delays is also a possible starting time of u in a schedule for the instance $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$ of length ℓ, whereas a possible starting in the schedule for $(G, m, \tau, \mu, \mathbf{0}_{E(G)})$ need not be a potential starting time in the schedule for (G, m, τ, μ, c). These observations make it easier to determine the existence of short schedules for instances with non-zero communication delays.

References

[1] H.H. Ali and H. El-Rewini. An optimal algorithm for scheduling interval ordered tasks with communication on n processors. *Journal of Computer and System Sciences*, 51(2):301–306, October 1995.

[2] D. Bernstein, J.M. Jaffe and I. Gertner. Scheduling arithmetic and load operations in parallel with no spilling. In *Proceedings of the Fourteenth Annual ACM Conference on Principles of Programming Languages*, pages 263–273, 1987.

[3] D. Bernstein, M. Rodeh and I. Gertner. Approximation algorithm for scheduling arithmetic expressions on pipelined machines. *Journal of Algorithms*, 10:120–139, 1989.

[4] D. Bernstein, M. Rodeh and I. Gertner. On the complexity of scheduling problems for parallel/pipelined machines. *IEEE Transactions on Computers*, 38(9):1308–1313, September 1989.

[5] P. Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40:353–359, 1988.

[6] T.C.E. Cheng and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1990.

[7] P. Chrétienne and C. Picouleau. Scheduling with communication delays: a survey. In P. Chrétienne, E.G. Coffman, Jr., J.K. Lenstra and Z. Liu, editors, *Scheduling Theory and its Applications*, Chapter 4, pages 65–90. John Wiley & Sons, Chichester, United Kingdom, 1995.

[8] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge MA, United States, 1990.

[9] S. Even, A. Itai and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, December 1976.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York NY, United States, 1979.

[11] M.R. Garey, D.S. Johnson and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, May 1976.

[12] L.A. Goldberg, M.S. Paterson, A. Srinivasan and E. Sweedyk. Better approximation guarantees for job-shop scheduling. Technical report CS-RR-312, Department of Computer Science, University of Warwick, Coventry, United Kingdom, August 1996.

[13] D.K. Goyal. Scheduling processor bound systems. Technical report CS-76-036, Computer Science Department, Washington State University, Pullman WA, United States, November 1976.

[14] N. Hefetz and I. Adiri. An efficient optimal algorithm for the two-machine unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, 7(3):354–360, August 1982.

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

[16] J.M. Jaffe. Bounds on the scheduling of typed task systems. *SIAM Journal on Computing*, 9(3):541–551, August 1980.

[17] K. Jansen. On scheduling problems restricted to interval orders. In E.W. Mayr, editor, *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, volume 657, pages 27–36. Springer-Verlag, Berlin, Germany, 1992.

[18] K. Jansen. Analysis of scheduling problems with typed task systems. *Discrete Applied Mathematics*, 52:223–232, 1994.

[19] H. Kellerer and G.J. Woeginger. UET-scheduling with constrained processor allocations. *Computers and Operations Research*, 19(1):1–8, 1992.

[20] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[21] J.K. Lenstra and A.H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[22] C.H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8(3):405–409, August 1979.

[23] C. Picouleau. *Etude de Problèmes d'Optimisation dans les Systèmes Distribués.* Thèse de doctorat, Université Pierre et Marie Curie, Paris, France, 1992.

[24] M.W. Schäffter. *Scheduling Jobs with Communication Delays: Complexity Results and Approximation Algorithms.* Dissertation, Technische Universität Berlin, Berlin, Germany, November 1996.

[25] J.P. Schmidt, A. Siegel and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 331–340, 1993.

[26] D.B. Shmoys, C. Stein and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, June 1994.

[27] Yu.N. Sotskov. The complexity of shop-scheduling problems with two or three jobs. *European Journal of Operational Research*, 53:326–336, 1991.

[28] Yu.N. Sotskov and N.V. Shakhlevich. NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, May 1995.

[29] J. Verriet. The complexity of scheduling graphs of bounded width subject to non-zero communication delays. Technical report UU-CS-1997-01, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, January 1997.

[30] J. Verriet. *Scheduling with communication for multiprocessor computation.* PhD thesis, Utrecht University, Utrecht, the Netherlands, June 1998.

[31] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'janov and D.B. Shmoys. Short shop schedules. *Operations Research*, 45(2):288–294, March–April 1997.