The complexity of scheduling graphs of bounded width subject to non-zero communication delays*

Jacques Verriet

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

Abstract

In this report, we study the complexity of scheduling problems for precedence graphs of bounded width. For such graphs, the size of a maximum anti-chain is bounded by a constant. It is shown that for graphs of bounded width with unit-length tasks and unit communication delays, a minimum-length schedule on m processors can be constructed in polynomial time using a dynamic-programming algorithm. This approach can be generalised to minimise different objective functions.

For graphs of width two with arbitrary task lengths, a polynomial-time algorithm is presented that constructs minimum-length and minimum-tardiness schedules on two processors. If its width is at least three, then constructing a minimum-length schedule for a graph with arbitrary task lengths on two processors is shown to be NP-hard. For the case that the width of the graph equals the number of processors, a dynamic-programming algorithm is presented that constructs minimum-length schedules in polynomial time.

1 Introduction

An important part of the complexity of parallel computing is due to communication. During the execution of a parallel program on a distributed memory computer architecture, there are delays between the execution of dependent tasks on different processors. These delays are needed to send the result of the computation of a task from one processor to the other. Classical scheduling problems do not take these communication delays into account and hence do not capture the complexity of parallel programming.

In this report, we consider the problem of scheduling with unit-length interprocessor communication delays on a finite set of identical processors. Most scheduling problems with unit communication delays are NP-hard, even if all tasks have unit length: Rayward-Smith [16] proved that constructing minimum-length schedules for arbitrary precedence graphs on m processors is NP-hard. Lenstra, et al. [14] showed the same for the case that the precedence constraints form a tree.

The only scheduling problems that do not neglect communication delays and are solvable in polynomial time consider unit-length tasks and a special kind of precedence constraints: Varvarigou, et al [18] presented an algorithm that constructs a minimum-length schedule on m processors for an outforest in $O(n^{2m-2})$ time. Finta, et al. [9] showed that for series-parallel graphs, a minimum-length schedule on two processors can be constructed in polynomial time. Ali and El-Rewini [1] presented a polynomial-time algorithm that constructs minimum-length schedules for sets of interval-ordered tasks.

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

For these special classes of precedence graphs, we cannot expect to find polynomial-time algorithms, if the task lengths are not restricted: the well-known NP-hard problem PARTITION [11] can be formulated as the problem of checking the existence of a schedule on two processors of length at most D for trees, series-parallel graphs and interval orders with arbitrary task lengths.

In this report, we will also consider special types of precedence constraints, but we will not restrict ourselves to unit-length tasks. The precedence relations are given by a directed acyclic graph of bounded width. In such graphs, the number of pairwise incomparable tasks (the size of a maximum anti-chain) is bounded by a constant integer w. In Section 3, it will be shown that graphs of width w can be considered as the disjoint union of w chains. These decompositions into chains are used by the algorithms that will be presented in this report.

Three polynomial-time algorithms are presented that construct minimum-length schedules for graphs of bounded width. The first considers unit-length tasks that have to be scheduled subject to unit communication delays. It uses the property that at any time t at most 2^w possible combinations of tasks can be executed. The algorithm uses a dynamic-programming approach similar to the ones presented by Möhring [15] and Veltman [19]. Given the set of tasks executed at or before time t, it considers all possible assignments of tasks to time $t+1$ and determines a set which yields a schedule whose length is minimum with respect to the tasks that have been scheduled earlier.

The dynamic-programming algorithm uses $O(n^w)$ time to construct a minimum-length schedule. It can be generalised to construct schedules that are optimal with respect to other objective functions in $O(n^{w+1})$ or $O(n^{w+2})$ time, depending on the objective function. In addition, it can be used to minimise the makespan for graphs in which the sum of the task lengths is bounded by a polynomial in n and the communication delays are bounded by a constant.

Bodlaender and Fellows [3] showed that constructing a minimum-length schedule for an arbitrary graph on k processors without communication delays is a $W[2]$-hard problem ($W[2]$ is the second class of the W-hierarchy for parametrised problems, that was introduced by Downey and Fellows [8]). Consequently, it is unlikely that an algorithm exists with an $O(n^c)$ running time that constructs minimum-length schedules on k processors for some constant c independent of k. In the $W[2]$-hardness proof, a graph of width $k+1$ is constructed. Therefore the same proof shows that minimising the maximum completion time for graphs of width k on $m < k$ processors is $W[2]$-hard. Using a simple reduction from this problem, one can easily prove that the same holds for scheduling with non-zero communication delays. As a result, the existence of an algorithm that constructs minimum-length schedules for graphs of width w in $O(n^c)$ time for all constant w is unlikely.

The second algorithm is presented in Section 5. It considers graphs of width at most two with arbitrary task lengths. It is a generalisation of an algorithm that constructs minimum-tardiness schedules for interval-ordered tasks of unit length with non-uniform deadlines [20]. It will be proved that the algorithm for interval orders also constructs minimum-tardiness schedules on two processors for graphs of width two with unit-length tasks subject to unit communication delays. Such a schedule is constructed in $O(n^2)$ time. The structure of the schedules for graphs with tasks of length one constructed by this algorithm is used to present a generalisation for scheduling graphs of width two with arbitrary task lengths. The generalised algorithm constructs minimum-tardiness schedules in $O(n^3)$ time. If the deadlines are chosen equal for each task, it constructs minimum-length schedules in $O(n^3)$ time for graphs with arbitrary task lengths and in $O(n^2)$ time for graphs with unit-length tasks.

The last two sections of this report consider graphs of width $w \geq 3$ with arbitrary task lengths. Using a polynomial reduction from PARTITION [11], we will prove that constructing a minimum-length schedule for a graph consisting of three chains on two processors is an NP-hard problem. This result implies that constructing schedules for graphs of width w on m processors with the objective of minimising to makespan is NP-hard for all constants w and m, such that $w \geq 3$ and $m < w$.

The third algorithm is presented in Section 7. It deals with the case that the number of

processors equals the width of the graph. It uses a dynamic-programming approach to construct minimum-length schedules for graphs of bounded width w with arbitrary task lengths on w processors in polynomial time. Like the dynamic-programming algorithm for graphs with unit-length tasks, the algorithm can be generalised, such that it constructs schedules that are optimal with respect to other objective functions.

2 Preliminary definitions

In this report, I will present several algorithms that construct schedules for parallel programs represented by a directed acyclic graph on m identical processors. An instance of such a scheduling problem is represented by a tuple (G, m, μ), where $G = (V, E)$ is a directed acyclic graph or precedence graph, m is the number of processors and $\mu : V \to \mathbb{Z}^+$ is a function assigning an execution length to every node of G. A node of G corresponds to a task of the parallel program. The execution of a task u on a processor takes $\mu(u)$ time units.

The set of arcs E represents the data dependencies between the tasks of G: if there is an arc from u_1 to u_2, then the result of the execution of u_1 is needed to compute u_2. If these tasks are executed on different processors, the result of the computation of u_1 has to be sent to another processor. This takes unit time. During this delay, both processors can execute another task. If u_1 and u_2 are executed by the same processor, no communication delay is required.

Let G be a precedence graph. The set V_G denotes the set of nodes of G and E_G the set of arcs. Throughout this report, we will assume V_G contains n tasks and E_G contains e arcs. Let u_1, u_2 be two tasks of G. u_1 is a *predecessor* of u_2, if there is a directed path in G from u_1 to u_2. In that case, u_2 is called a *successor* of u_1, which is denoted by $u_1 \prec u_2$. Tasks without successors will be called *sinks*, tasks without predecessors will be called *sources*. u_2 is called a *child* of u_1 if (u_1, u_2) is an arc of G. If u_2 is a child of u_1, then u_1 is called a *parent* of u_2. The number of children of a task u is the *outdegree* of u; its *indegree* is the number of parents of u.

Two tasks u_1 and u_2 of G are called *incomparable* if neither $u_1 \prec u_2$, nor $u_2 \prec u_1$. Otherwise, they are called *comparable*. The *width* of G is the maximum number of pairwise incomparable tasks of G. Consequently, if G is a graph of width w, then every subset of G with at least $w + 1$ elements contains at least two comparable tasks. A *chain* is a set of pairwise comparable tasks, a set of pairwise incomparable tasks is called an *anti-chain*. Hence, the width of G equals the size of a maximum-size anti-chain of G.

A graph of width w can be viewed as the disjoint union of w chains: such a graph consists of w chains with precedence constraints between the tasks in the different chains. In the next section, an algorithm is presented that constructs a decomposition into w chains in polynomial time. These decompositions are used by the dynamic-programming algorithms presented in this report.

A *schedule* for an instance (G, m, μ) is a pair of functions $\sigma : V_G \to \mathbb{N}$ and $\pi : V_G \to \{1, \ldots, m\}$. $\sigma(u)$ denotes the starting time of u and $\pi(u)$ the processor on which u is executed. An assignment of starting times σ is called *feasible* for (G, m, μ) if, for all tasks $u_1 \neq u_2$ of G and all $t \leq \max_u (\sigma(u) + \mu(u))$,

1. $|\{u \in G \mid \sigma(u) \leq t < \sigma(u) + \mu(u)\}| \leq m$;
2. if $u_1 \prec u_2$, then $\sigma(u_1) + \mu(u_1) \leq \sigma(u_2)$;
3. there is at most one predecessor v of u_1 with $\sigma(v) + \mu(v) = \sigma(u_1)$; and
4. there is at most one successor v of u_1 with $\sigma(v) = \sigma(u_1) + \mu(u_1)$.

A schedule (σ, π) for (G, m, μ) is called a *feasible schedule* if σ is a feasible assignment of starting times for (G, m, μ) and for all tasks $u_1 \neq u_2$ of G, such that, for all times t, if $\sigma(u_1) \leq t < \sigma(u_1) + \mu(u_1)$ and $\sigma(u_2) \leq t < \sigma(u_2) + \mu(u_2)$, then $\pi(u_1) \neq \pi(u_2)$.

Consider a feasible schedule (σ, π) for an instance (G, m, μ). Let u be a task of G. If $\sigma(u) = t$, then u is *scheduled at time* t. In addition, u is said to be executed at times $\sigma(u), \ldots, \sigma(u)+\mu(u)-1$ on processor $\pi(u)$. $\sigma(u) + \mu(u)$ is the *completion time* of u. The *length* of σ (and (σ, π)) is the maximum completion time of a task of G.

For a feasible assignment of starting times σ for (G, m, μ), there is a processor assignment π, such that (σ, π) is a feasible schedule for (G, m, μ). Such a processor assignment is constructed by Algorithm PROCESSOR ASSIGNMENT COMPUTATION shown in Figure 1 using a feasible assignment of starting times σ. The following notations are used. $idle(j)$ denotes the minimum time t, such that processor j is idle from time t onward. The algorithm repeatedly assigns a processor to all tasks with the minimum starting time. i_1 denotes the index of the first of these tasks, u_{i_2} is the first task with a larger starting time than u_{i_1}. Assume u_{n+1} is a dummy task with starting time $\sigma(u_{n+1}) = \infty$.

Algorithm PROCESSOR ASSIGNMENT COMPUTATION
Input: A feasible assignment of starting times σ for (G, m, μ).
Output: An assignment of processors π, such that (σ, π) is a feasible schedule for (G, m, μ).

```
1.  assume σ(u_1) ≤ ... ≤ σ(u_n) ≤ σ(u_{n+1}) = ∞
2.  for j := 1 to m
3.      do idle(j) := 0
4.  i_1 := 1
5.  i_2 := min{i > i_1 | σ(u_i) > σ(u_{i_1})}
6.  t := σ(u_{i_1})
7.  while i_1 ≤ n
8.      do U := ∅
9.          for i := i_1 to i_2 − 1
10.             do if u_i has a predecessor v with σ(v) + μ(v) = t
11.                 then π(u_i) := π(v)
12.                      idle(π(u_i)) := t + μ(u_i)
13.                 else U := U ∪ {u_i}
14.         for u ∈ U
15.             do determine j, such that idle(j) ≤ t
16.                π(u) := j
17.                idle(j) := t + μ(u)
18.         i_1 := i_2
19.         i_2 := min{i > i_1 | σ(u_i) > σ(u_{i_1})}
20.         t := σ(u_{i_1})
```

Figure 1: The processor assignment algorithm

Since Algorithm PROCESSOR ASSIGNMENT COMPUTATION starts with a feasible assignment of starting times σ for an instance (G, m, μ), at any time t at most m tasks are executed. So at all times t considered by Algorithm PROCESSOR ASSIGNMENT COMPUTATION, the number of idle processors is sufficient to schedule every task with starting time t. As a result, if processor assignment π is constructed by Algorithm PROCESSOR ASSIGNMENT COMPUTATION using assignment of starting times σ, then the schedule (σ, π) is a feasible schedule for (G, m, μ).

Sorting the tasks by non-decreasing starting times takes $O(n \log n)$ time. Calculating i_1 and i_2 can be done by traversing the list of tasks exactly once. Hence this requires $O(n)$ time in total. The same holds for the computation of the set U. Determining a processor for u_i takes $O(indegree(u_i))$ time. Hence Algorithm PROCESSOR ASSIGNMENT COMPUTATION constructs a correct processor assignment in $O(m + n \log n + e)$ time.

In a feasible schedule (σ, π), comparable tasks are not executed simultaneously. Hence, for

graphs of width w, the number of tasks that can be executed at the same time is at most w. Therefore $|\{u \in G \mid \sigma(u) \leq t < \sigma(u) + \mu(u)\}| \leq w$ for all t. So we may assume $m \leq w$. So, given a feasible assignment of starting time σ for (G, m, μ), Algorithm PROCESSOR ASSIGNMENT COMPUTATION constructs an assignment of processors π in $O(n \log n + e)$ time. If all tasks are of unit length, then we may assume no starting time exceeds $n - 1$. In that case, Algorithm PROCESSOR ASSIGNMENT COMPUTATION requires only $O(n + e)$ time, because sorting n numbers whose values are at most n requires only linear time using counting sort [6].

3 Decomposition into chains

Every graph can be viewed as a collection of disjoint chains with some dependencies between tasks in different chains: every graph with n nodes can be considered as the disjoint union of n chains consisting of one task. Obviously, graphs that do not consist of n pairwise incomparable tasks can be decomposed into a smaller number of chains.

In Figure 2, a decomposition of a graph G of width two into two disjoint chains $C_1 = \{c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}\}$ and $C_2 = \{c_{21}, c_{22}, c_{23}, c_{24}\}$ is shown. Note that C_2 does not correspond to a connected subgraph of G: a chain is a set of pairwise comparable tasks. Consequently, a chain is a connected subgraph of the transitive closure of G. In addition, a decomposition into disjoint chains is not unique: the graph of Figure 2 has more than one decomposition. Another decomposition of this graph is given by the chains $C_1' = \{c_{11}, c_{12}, c_{13}, c_{22}, c_{23}, c_{24}\}$ and $C_2' = \{c_{21}, c_{14}, c_{15}, c_{16}\}$.

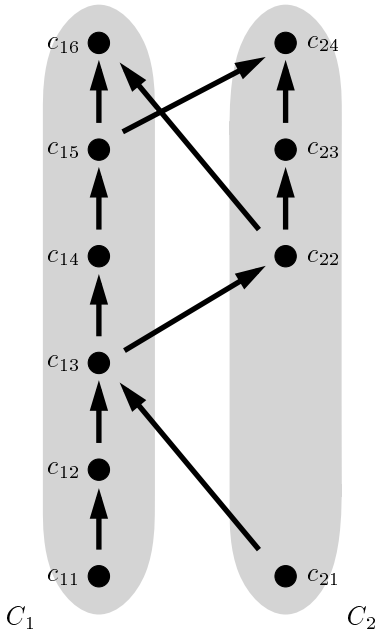


Figure 2: A graph of width two and a decomposition into two disjoint chains

Clearly, a graph of width w cannot be split into less than w chains. Dilworth [7] proved that a graph of width w can be viewed as the disjoint union of w chains.

Theorem 3.1 (Dilworth's decomposition theorem). *Let G be a graph of width w. There are w chains $C_1, \ldots, C_w$ in G, such that G is the disjoint union of $C_1, \ldots, C_w$.*

Such a collection of chains is called a *decomposition* of G. A decomposition of a graph of width w into w chains will be used by the dynamic-programming algorithms presented in Sections 4

and 7. Unfortunately, Dilworth's proof is not constructive: using his proof, one cannot construct a decomposition in an efficient way.

The proof by Fulkerson [10] is constructive. In his proof of Dilworth's decomposition theorem, Fulkerson presented Algorithm CHAIN DECOMPOSITION shown in Figure 3 and proved that it constructs chain decompositions consisting of w chains for graphs of width w.

Algorithm CHAIN DECOMPOSITION
Input: A precedence graph G of width w.
Output: A chain decomposition $C_1, \ldots, C_w$ of G.
1. assume $V_G = \{u_1, \ldots, u_n\}$
2. $V := \{a_1, \ldots, a_n\} \cup \{b_1, \ldots, b_n\}$
3. $E := \{(a_i, b_j) \mid u_i \prec u_j\}$
4. let M be a maximum matching of $H = (V, E)$
5. $E' := \{(u_i, u_j) \mid (a_i, b_j) \in M\}$
6. $i := 1$
7. **while** G contains unmarked tasks
8. **do** let u be an unmarked task of G
9. $C_i := \{v \in V_G \mid$ there is an undirected path from u to v in $(V_G, E')\}$
10. mark all tasks in C_i
11. $i := i + 1$

Figure 3: The chain decomposition algorithm

If G is a transitive closure, then the construction of the undirected bipartite graph H takes $O(e)$ time. Otherwise, compute the transitive closure G^+ of G. This takes $O(n^{2.376})$ time [5]. Afterward, construct H using the arcs of G^+. Then H is constructed in $O(n^{2.376})$ time.

It is not difficult to see that the time needed to construct the decomposition using bipartite graph H is dominated by the time needed to compute a maximum matching M of H. Hopcroft and Karp [13] presented an algorithm that computes a maximum matching in $O(e\sqrt{n})$ time for bipartite graphs with n nodes and e edges. Alt, et al. [2] presented an algorithm whose running time is better for dense graphs: it constructs a maximum matching in $O(n\sqrt{ne/\log n})$ time.

Let e^+ be the number of arcs in G^+. The number of edges in H equals e^+. As a result, a maximum matching M in H can be constructed in $O(\min\{e^+\sqrt{n}, n\sqrt{ne^+/\log n}\})$ time. So constructing a decomposition of a graph of width w into w disjoint chains takes $O(\min\{e^+\sqrt{n}, n\sqrt{ne^+/\log n}\})$ time.

Theorem 3.2. *Let G be a graph of width w. Algorithm* CHAIN DECOMPOSITION *constructs a decomposition of G into w chains in $O(\min\{e^+\sqrt{n}, n\sqrt{ne^+/\log n}\})$ time, where e^+ is the number of arcs in the transitive closure of G.*

Suppose G be graph of width w. Then G contains a chain of size at least $\frac{n}{w}$. The transitive closure of a chain of size ℓ contains $\frac{1}{2}\ell(\ell-1)$ arcs. Therefore the transitive closure of G contains at least $\frac{n(n-1)}{2w^2}$ tasks. Hence $e^+ = \Theta(n^2)$ if w is a constant. So constructing a chain decomposition for a graph of bounded width takes $O(\frac{n^2\sqrt{n}}{\sqrt{\log n}})$ time.

Theorem 3.3. *Let w be a constant. Let G be a graph of width w. Algorithm* CHAIN DECOMPOSITION *constructs a decomposition of G into w chains in $O(\frac{n^2\sqrt{n}}{\sqrt{\log n}})$ time.*

4 A dynamic-programming algorithm for UET-graphs

In this section, I will present a dynamic-programming algorithm that constructs minimum-length schedules for graph with unit-length tasks. For graphs of fixed width w, a minimum-length schedule

is constructed in $O(n^w)$ time. The same approach can be used to construct schedules that are optimal with respect to other objective functions.

The running time of the dynamic-programming algorithm is exponential in the width of the precedence graph. It is unlikely that there is an algorithm that constructs minimum-length schedules in $O(n^c)$ time, where c is a constant independent of the width of the graph: Bodlaender and Fellows [3] proved that constructing a minimum-length schedule for an arbitrary precedence graph (without communication delays) on k processors is $W[2]$-hard. This implies that it is unlikely that, for all fixed k, a minimum-length schedule for a graph on k processors can be constructed in $O(n^c)$ time for some constant c. In fact, Bodlaender and Fellows [3] proved this for graphs of width $k+1$ that have to be scheduled on k processors. Their result can be easily generalised to scheduling with non-zero communication delays.

Consider a feasible schedule (σ, π) for the instance $(G, m, \mathbb{1})$, where $\mathbb{1}(u) = 1$ for all tasks u of G. Since we will only consider objective functions that do not increase if a task starts at an earlier time, we may assume all tasks of G are completed at time n. Therefore σ can be viewed as an array of at most n sets of tasks, namely the sets $\sigma^{-1}(t)$ containing the tasks of G that start at time t. These sets will be called the *time slots* of σ. It is easy to see that a time slot is an anti-chain of G.

The dynamic-programming algorithm considers all feasible assignments of starting times for an instance $(G, m, \mathbb{1})$. This is done as follows. Suppose the time slots $\sigma^{-1}(0), \ldots, \sigma^{-1}(t)$ form a feasible schedule for $(G_t, m, \mathbb{1})$, where G_t is the subgraph of G induced by $\bigcup_{i=0}^{t} \sigma^{-1}(i)$. The only tasks that can be scheduled at time $t+1$ without violating the feasibility of the partial schedule are sources of $G \setminus \bigcup_{i=0}^{t} \sigma^{-1}(i)$. For graphs of width w, there are at most w such tasks, because the sources of a graph are incomparable. Hence there are at most 2^w sets of tasks that can be executed at time $t+1$. This is the basis of the dynamic-programming algorithm: for each time t, it considers all sets of tasks that can be scheduled at time $t+1$ and determines the best one.

Consider a partial schedule $\sigma^{-1}(0), \ldots, \sigma^{-1}(t)$ for the instance $(G, m, \mathbb{1})$. Because of communication delays, the execution of a set of sources of $G \setminus \bigcup_{i=0}^{t} \sigma^{-1}(i)$ might yield infeasible schedules for $(G, m, \mathbb{1})$. Let U be a set of sources of $G \setminus \bigcup_{i=0}^{t} \sigma^{-1}(i)$. U is called *available* with respect to the time slots $\sigma^{-1}(0), \ldots, \sigma^{-1}(t)$ if there is a feasible assignment of starting times σ_1 for $(G, m, \mathbb{1})$, such that

$$\sigma_1^{-1}(i) = \sigma^{-1}(i) \text{ for all } i,\ 0 \leq i \leq t, \text{ and } \sigma_1^{-1}(t+1) = U.$$

In other words, if σ is a feasible assignment of starting times for $(G_t, m, \mathbb{1})$, where G_t is the subgraph of G induced by $\bigcup_{i=0}^{t} \sigma^{-1}(i)$, then U is available with respect to $\sigma^{-1}(0), \ldots, \sigma^{-1}(t)$ if

1. $|U| \leq m$;
2. $\left|\{v \in \sigma^{-1}(t) \mid v \prec u\}\right| \leq 1$ for all u in U; and
3. $|\{u \in U \mid v \prec u\}| \leq 1$ for all v in $\sigma^{-1}(t)$.

Note that the availability of U only depends on $\bigcup_{i=0}^{t} \sigma^{-1}(i)$ and $\sigma^{-1}(t)$. Therefore U will be called available with respect to $(\bigcup_{i=0}^{t} \sigma^{-1}(i), \sigma^{-1}(t))$.

Consider an instance $(G, m, \mathbb{1})$. Let U be a set of tasks of G. U is called a *prefix* of $(G, m, \mathbb{1})$, if for all tasks u_1 and u_2 of G,

$$\text{if } u_2 \in U \text{ and } u_1 \prec u_2, \text{ then } u_1 \in U.$$

A *starred prefix* of $(G, m, \mathbb{1})$ is a pair (U_1, U_2), where U_1 is a prefix of $(G, m, \mathbb{1})$ and U_2 is a subset of the sinks of U_1. Note that for a feasible schedule (σ, π) for $(G, m, \mathbb{1})$, the pair $(\bigcup_{i=0}^{t} \sigma^{-1}(i), \sigma^{-1}(t))$ is a starred prefix of $(G, m, \mathbb{1})$ for all times t.

The length of a minimum-length schedule for $(G, m, 1\!\!1)$ can be computed in terms of starred prefixes of $(G, m, 1\!\!1)$. Let (U_1, U_2) and (U_1', U_2') be two starred prefixes of $(G, m, 1\!\!1)$. (U_1', U_2') is called *available* with respect to (U_1, U_2), if U_2' is available with respect to (U_1, U_2) and $U_1' = U_1 \cup U_2'$. $Av(U_1, U_2)$ denotes the set of starred prefixes of $(G, m, 1\!\!1)$ that are available with respect to (U_1, U_2). Note that if $U_2 = \varnothing$, then the starred prefix (U_1, U_2) is available with respect to itself. To avoid infinite computations, we will introduce the set $Av^*(U_1, U_2) = Av(U_1, U_2) \setminus \{(U_1, U_2)\}$.

Suppose (σ, π) is a feasible schedule for $(G, m, 1\!\!1)$. Then it is clear that the starred prefix $(\bigcup_{i=0}^{t} \sigma^{-1}(i), \sigma^{-1}(t))$ is available with respect to $(\bigcup_{i=0}^{t-1} \sigma^{-1}(i), \sigma^{-1}(t-1))$ for all times t. As a result, we only need to consider starred prefixes of $(G, m, 1\!\!1)$.

Let (U_1, U_2) be a starred prefix of $(G, m, 1\!\!1)$. $L(U_1, U_2)$ denotes the maximum completion time of a task in a minimum-length schedule (σ, π) for $(G', m, 1\!\!1)$, where G' is the subgraph of G induced by $(G \setminus U_1) \cup U_2$, such that $\sigma(u) = -1$ for all u in U_2. Hence we find

$$L(U_1, U_2) = 1 + \max_{(U_1', U_2') \in Av^*(U_1, U_2)} L(U_1', U_2')$$

and $L(V_G, U_2) = 0$ for all sets of sinks U_2 of G. Obviously, $L(\varnothing, \varnothing)$ equals the length of a minimum-length schedule for $(G, m, 1\!\!1)$.

Let $C = \{c_1, \ldots, c_\ell\}$ be a chain of G. Assume $c_1 \prec \ldots \prec c_\ell$. The chain $C^j = \{c_1, \ldots, c_j\}$ is a *prefix* of C for all j, $0 \leq j \leq \ell$.

Consider an instance $(G, m, 1\!\!1)$, where G is a graph of width w. Assume $C_1, \ldots, C_w$ is a decomposition of G into disjoint chains and the chains C_i contain ℓ_i tasks $c_{i1}, \ldots, c_{i\ell_i}$. It is easy to see that a prefix U of G equals the disjoint union of prefixes of $C_1, \ldots, C_w$. Hence U can be represented by a sequence of w numbers $(b_1, \ldots, b_w)$, such that $0 \leq b_i \leq \ell_i$ for each i. Moreover, every sequence $(b_1, \ldots, b_w)$ coincides with the prefix $\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}$ of $(G, m, 1\!\!1)$.

A starred prefix (U_1, U_2) of $(G, m, 1\!\!1)$ will be represented by a tuple $(b_1, \ldots, b_w, a_1, \ldots, a_w)$, where $(b_1, \ldots, b_w)$ represents the prefix U_1 and $a_i \in \{0, 1\}$, such that $a_i = 1$ if and only if $c_{ib_i} \in U_2$. In addition, if $b_i > 0$ for all i with $a_i = 1$, then the sequence $(b_1, \ldots, b_w, a_1, \ldots, a_w)$ corresponds to the starred prefix $(\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}, \{c_{ib_i} \mid a_i = 1\})$ of $(G, m, 1\!\!1)$.

Let $(G, m, 1\!\!1)$ be an instance, where G is a graph of width w with a chain decomposition $C_1, \ldots, C_w$. For the implementation of the computation of $L(\varnothing, \varnothing)$, we create a table T of dimension $2w$. For each $i \leq w$, index j_i has a lower bound 0 and an upper bound ℓ_i. For $i \geq w + 1$, index $j_i \in \{0, 1\}$. A table entry $T[b_1, \ldots, b_w, a_1, \ldots, a_w]$ will be called *feasible* if, for all i, if $a_i = 1$, then $b_i > 0$. The value of a feasible table entry $T[b_1, \ldots, b_w, a_1, \ldots, a_w]$ corresponds to $L(\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}, \{c_{ib_i} \mid a_i = 1\})$.

In the beginning, set $T[b_1, \ldots, b_w, a_1, \ldots, a_w] = \infty$ for all sequences. The goal of the algorithm is computing $L(\varnothing, \varnothing)$, which corresponds to computing $T[0, \ldots, 0, 0, \ldots, 0]$.

Algorithm UET TABLE CONSTRUCTION is presented in Figure 4. It computes the values of the table entries $T[seq(U_1, U_2)]$, where $seq(U_1, U_2)$ denotes the tuple $(b_1, \ldots, b_w, a_1, \ldots, a_w)$, such that $U_1 = \bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}$ and $U_2 = \{c_{ib_i} \mid a_i = 1\}$.

Applying Algorithm UET TABLE CONSTRUCTION to $(\varnothing, \varnothing)$ corresponds to computing the length of a minimum-length schedule for an instance $(G, m, 1\!\!1)$. Hereafter, $T[seq(U_1, U_2)]$ contains $L(U_1, U_2)$ for all starred prefixes (U_1, U_2) of $(G, m, 1\!\!1)$. Using these values, Algorithm UET SCHEDULE CONSTRUCTION shown in Figure 5 constructs a minimum-length assignment of starting times for $(G, m, 1\!\!1)$.

Let $(G, m, 1\!\!1)$ be an instance, where G is a graph of width w with chain decomposition $C_1, \ldots, C_w$. For every starred prefix (U_1, U_2) of $(G, m, 1\!\!1)$, Algorithm UET TABLE CONSTRUCTION computes $Av^*(U_1, U_2)$. A starred prefix is represented by a sequence $(b_1, \ldots, b_w, a_1, \ldots, a_w)$. Hence determining all potential elements of $Av^*(U_1, U_2)$ takes $O(2^w)$ time: if (U_1', U_2') is available with respect to $(U_1, U_2) = (\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}, \{c_{ib_i} \mid a_i = 1\})$, then $U_2' \subseteq \{c_{1b_1+1}, \ldots, c_{wb_w+1}\}$. Since prefixes contain at most w tasks, checking the availability of (U_1', U_2') with respect to (U_1, U_2) takes $O(w^2)$ time.

Algorithm UET TABLE CONSTRUCTION(U_1, U_2)
Input: An instance $(G, m, \mathbb{1})$, such that G is a graph of width w with chain decomposition $C_1, \ldots, C_w$ and a starred prefix (U_1, U_2) of $(G, m, \mathbb{1})$.
Output: A table T with $T[seq(U_1, U_2)] = L(U_1, U_2)$.

```
1.  if T[seq(U1,U2)] = ∞
2.     then if U1 = VG
3.             then T[seq(U1,U2)] := 0
4.             else for (U1',U2') in Av*(U1,U2)
5.                     do UET TABLE CONSTRUCTION(U1',U2')
6.                T[seq(U1,U2)] := 1 + min{T[seq(U1',U2')] | (U1',U2') ∈ Av*(U1,U2)}
```

Figure 4: The algorithm computing $L(U_1, U_2)$

Algorithm UET SCHEDULE CONSTRUCTION
Input: A instance $(G, m, \mathbb{1})$ and a table T with $T[seq(U_1, U_2)] = L(U_1, U_2)$ for all starred prefixes (U_1, U_2) of $(G, m, \mathbb{1})$.
Output: A minimum-length assignment of starting times σ for $(G, m, \mathbb{1})$.

```
1.  U1 := ∅
2.  U2 := ∅
3.  t := 0
4.  while U1 ≠ VG
5.     do determine (U1',U2') in Av*(U1,U2), such that T[seq(U1',U2')] is minimum
6.        for u ∈ U2'
7.           do σ(u) := t
8.        U1 := U1'
9.        U2 := U2'
10.       t := t + 1
```

Figure 5: The algorithm assigning starting times using the values $L(U_1, U_2)$

Hence for each starred prefix, $O(w^2 2^w)$ time is used. Every starred prefix of $(G, m, \mathbb{1})$ corresponds to a tuple $(b_1, \ldots, b_w, a_1, \ldots, a_w)$ with $0 \leq b_i \leq \ell_i$ and $a_i \in \{0, 1\}$. So the number of starred prefixes of $(G, m, \mathbb{1})$ is at most

$$2^w \prod_{i=1}^{w} (\ell_i + 1) \;\leq\; 2^w \prod_{i=1}^{w} 2\ell_i \;\leq\; 2^{2w} \prod_{i=1}^{w} \frac{n}{w} \;\leq\; 2^w n^w.$$

As a result, Algorithm UET TABLE CONSTRUCTION calculates the length of a minimum-length schedule for $(G, m, \mathbb{1})$ in $O(w^2 2^{2w} n^w)$ time.

Algorithm UET SCHEDULE CONSTRUCTION is used to build an assignment of starting times σ that corresponds to a minimum-length schedule for $(G, m, \mathbb{1})$. For every time t and starred prefix $(\bigcup_{i=0}^{t} \sigma^{-1}(i), \sigma^{-1}(t))$, starting with $t = 0$ and $(\varnothing, \varnothing)$, it determines the set $Av^*(U_1, U_2)$ and chooses the best element of this set. Since a minimum-length schedule has length at most n, Algorithm UET SCHEDULE CONSTRUCTION computes $O(n)$ sets $Av^*(U_1, U_2)$. Hence it uses $O(w^2 2^w n)$ time to construct an assignment of starting times corresponding to a minimum-length schedule for $(G, m, \mathbb{1})$.

Theorem 4.1. *Let $(G, m, \mathbb{1})$ be an instance, where G is a graph of width w. Algorithms* CHAIN DECOMPOSITION, UET TABLE CONSTRUCTION *and* UET SCHEDULE CONSTRUCTION *construct a minimum-length schedule for $(G, m, \mathbb{1})$ in $O(w^2 2^{2w} n^w + \min\{e^+ \sqrt{n}, n\sqrt{ne^+ / \log n}\})$ time, where e^+ is the number of arcs in the transitive closure of G.*

For constant w, a minimum-length schedule can be constructed in polynomial time.

Theorem 4.2. *Let w be a constant. Let $(G, m, 1\!\!1)$ be an instance, where G is a graph of width w. Algorithms* Chain decomposition, UET Table construction *and* UET Schedule construction *construct a minimum-length schedule for $(G, m, 1\!\!1)$ in $O(n^w + \frac{n^2\sqrt{n}}{\sqrt{\log n}})$ time.*

Algorithms UET Table construction and UET Schedule construction construct minimum-length schedules for instances $(G, m, 1\!\!1)$ in polynomial time if G is a graph of constant width w. A similar approach can be used for minimising a different objective function. However, for some of these objective functions, extra information is needed, namely the current time slot and, in some cases, the value of the objective function so far.

For example, if we want to minimise the weighted sum of completion times (without negative weights) for an instance $(G, m, 1\!\!1)$, we need the current time. For this objective function, we consider tuples (t, U_1, U_2), where t is the current time slot and (U_1, U_2) is a starred prefix of $(G, m, 1\!\!1)$. Because negative weights are not allowed, there is an optimal schedule of length at most n. So $O(n^{w+1})$ such tuples have to be taken into account. The following recursion has to be solved.

$$C(t, U_1, U_2) = \min_{(U_1', U_2') \in Av^*(U_1, U_1)} C(t+1, U_1', U_2') + (t+1) \sum_{u \in U_2'} w(u),$$

where $w(u)$ denotes the weight of u. Because of the extra t in the tuples, constructing a schedule in which the weighted sum of completion times is minimised takes $O(n^{w+1})$ time if G is a graph of constant width w. Other objective functions that need the extra element t are, for example, the number of tardy/late tasks and the total weighted lateness/tardiness (with non-negative weights).

Maximum tardiness is an example of an objective function for which not only the current time t, but also the maximum tardiness of the tasks scheduled before time t is included in the tuple. Every task u has a deadline $D_0(u)$. The tardiness of a task u in a schedule (σ, π) is the amount of time that its completion time exceeds its deadline: that is $T(u) = \max\{0, \sigma(u) + 1 - D_0(u)\}$. Suppose we want to construct a minimum-tardiness schedule for an instance $(G, m, 1\!\!1, D_0)$. The maximum tardiness of a task in a minimum-tardiness schedule for $(G, m, 1\!\!1, D_0)$ is bounded by n and there is a minimum-tardiness schedule of length at most n, so $O(n^{w+2})$ tuples have to be taken into account by the dynamic-programming algorithm. These look like (t, T_t, U_1, U_2), where t is the current time, T_t denotes the maximum tardiness among the tasks scheduled before time t and (U_1, U_2) is a starred prefix of $(G, m, 1\!\!1, D_0)$. The maximum tardiness of an optimal schedule can be computed by solving the recursion

$$T(t, T_t, U_1, U_2) = \min_{(U_1', U_2') \in Av^*(U_1, U_2)} T(t+1, T_{t+1}, U_1', U_2'),$$

where $T_{t+1} = \max\{T_t, \max_{u \in U_2'} t + 1 - D_0(u)\}$. Obviously, $T(t, T_t, V_G, U_2) = T_t$ for all t and all sets of sinks U_2 of G. Then $T(0, 0, \varnothing, \varnothing)$ corresponds to the tardiness of a minimum-tardiness schedule for $(G, m, 1\!\!1, D_0)$. This recursion can be solved in $O(n^{w+2})$ time if G has width w for some constant w. The lateness of a task u equals $\sigma(u) + 1 - D_0(u)$. Constructing a schedule in which the maximum lateness of a task is minimised can be done in the same way.

For some scheduling problems, the algorithm has to be changed a bit more. For instance, if every task has a release date, then the length of a schedule that is optimal with respect to some objective function, need not be bounded by a polynomial in n. In that case, the running time of the dynamic-programming algorithm in its original form would be pseudo-polynomial. Consider an instance $(G, m, 1\!\!1, R)$, where $R : V_G \to \mathbb{N}$ is a function assigning a release date to every task of G. It is not difficult to prove that in a greedy schedule for $(G, m, 1\!\!1, R)$, every task has at most n possible starting times: task u is scheduled before time $R(u) + n$. Moreover, for all objective functions, there are optimal schedules that are greedy. So the total number of times that need to taken into account is at most n^2. Assume G is a graph of bounded width. By introducing all possible starting times into the recursion, the dynamic-programming algorithm constructs schedules for $(G, m, 1\!\!1, R)$ that are optimal with respect to several objective functions.

If the objective function is not bounded by a polynomial in n, the number of possible tuples could become pseudo-polynomial in the input length of the scheduling problem. However, in many

cases, the objective function is not bounded by a polynomial in n, but the number of potential values of the objective function for an optimal schedule is. For instance, suppose we want to construct a schedule for an instance $(G, m, 1\!\!1, D_0)$ that minimises the maximum weighted tardiness. Assume G is a graph of bounded width. The maximum weighted tardiness of an optimal schedule need not be polynomial in n. However, we know that, in a greedy schedule, each task has a tardiness at most $n-1$. Hence the maximum weighted tardiness of an optimal schedule for $(G, m, 1\!\!1, D_0)$ is an element of $\bigcup_{i=1}^{n}\{0, w(u_i), 2w(u_i), \ldots, (n-1)w(u_i)\}$, where $V_G = \{u_1, \ldots, u_n\}$ and $w(u_i)$ is the weight of u_i. So the number of values of the objective function is bounded by n^2. Consequently, the dynamic-programming approach can be used to construct a schedule for $(G, m, 1\!\!1, D_0)$ in which the maximum weighted tardiness is minimised.

The applicability of the dynamic-programming approach is not restricted to the scheduling model in which every task can be executed on every processor. It can also be used if for every task there is a set of possible processors or if every task uses some resources which are not available at every time. To use the dynamic-programming algorithm for these scheduling models, the definition of $Av(U_1, U_2)$ has to be modified. Using the new definition, the dynamic-programming algorithm uses only polynomial time: for instances $(G, m, 1\!\!1)$, where G is a graph of fixed width w, $Av(U_1, U_2)$ contains at most 2^w elements for every definition of availability.

Furthermore, the task lengths and the communication delays need not be equal. If the maximum task length is bounded by n^k for some constant k and the maximum communication delay is bounded by a constant c, then the dynamic-programming approach is successful for graphs of bounded width. By splitting every task of length μ into μ tasks of unit length (note that this does not change the width of the graph), the number of tasks remains polynomial in n. Note that the definition of availability has to be updated, because the unit-length tasks from an original tasks have to be executed without interruption.

In case of unit-length communication delays, the dynamic-programming algorithm has to consider only the last time slot. If the communication delays are bounded by c, then the availability of a task at time t depends on the tasks executed at times $t-c, \ldots, t-1$. So a dynamic-programming approach would consider tuples $(U, U_1, \ldots, U_c)$, where U is a prefix and U_i is a set of sinks of $U \setminus \bigcup_{j=i+1}^{c} U_j$. The number of such tuples is at most $n^w 2^{wc}$. $(U, U_1, \ldots, U_c)$ can be represented by a sequence $(b_1, \ldots, b_w, a_{11}, \ldots, a_{1w}, \ldots, a_{c1}, \ldots, a_{cw})$, where $(b_1, \ldots, b_w)$ corresponds to U and $a_{j1}, \ldots, c_{jw} \in \{0, 1\}$ to U_j:

$$a_{ji} = 1 \quad \text{if and only if} \quad c_{ii'} \in U_j, \text{ where } i' = b_i - |\{j' > j \mid a_{j'i} = 1\}|.$$

Using a dynamic-programming approach, we can construct a schedule that is optimal with respect to some objective function in polynomial time. In particular, the dynamic-programming approach can be used for scheduling graphs of bounded width without communication delays.

The algorithm presented in this section is similar to the one by Möhring [15] for scheduling graphs of bounded width without communication delays, and the one presented by Veltman [19] for scheduling with unit communication delays. Like the algorithm presented in this section, these algorithms use prefixes. They use the name order ideal instead of prefix. When constructing a minimum-length schedule for an instance $(G, m, 1\!\!1)$ without communication delays, Möhring [15] constructs a directed graph on the prefixes of $(G, m, 1\!\!1)$: there is an arc from U_1 to U_2, if

$$U_1 \subseteq U_2,\ |U_2 \setminus U_1| \leq m, \text{ and } U_2 \setminus U_1 \text{ does not contain comparable elements.}$$

A path $(U_0, \ldots, U_{\ell-1})$ from $\varnothing$ to V_G in this graph corresponds to a feasible schedule for $(G, m, 1\!\!1)$ of length ℓ: the tasks of $U_t \setminus U_{t-1}$ start at time t. By traversing all nodes in the directed graph on the prefixes of $(G, m, 1\!\!1)$, a minimum-length schedule can be constructed. If G is a graph of fixed width w, this takes $O(n^w)$ time, since there are $O(n^w)$ prefixes of $(G, m, 1\!\!1)$.

To construct a minimum-length schedule for $(G, m, 1\!\!1)$ with unit communication delays, a directed graph H on pairs (U_1, U_2) is constructed, where U_1 is a prefix of $(G, m, 1\!\!1)$ and U_2 is set

of sources of $V_G \setminus U_1$: there is an arc from (U_1, U_2) to (U_1', U_2'), if

$$U_2' \setminus U_2 = U_2',\ |U_2'| \leq m, \text{ and } U_2' \text{ is available with respect to } (U_1 \cup U_2, U_2).$$

So there is an arc from (U_1, U_2) to (U_1', U_2'), if the starred prefix $(U_1' \cup U_2', U_2')$ is available with respect to $(U_1 \cup U_2, U_2)$. A path in the directed graph on the pairs (U_1, U_2) corresponds to a feasible schedule for $(G, m, 1\!\!1)$. Veltman [19] states that a minimum-length schedule is constructed in $O(n^{2w})$ if G is a graph of fixed width w. This bound can be decreased to $O(n^w)$, because there are $O(n^w)$ prefixes of G, and for every prefix U_1, there are at most 2^w sets of sources of $V_G \setminus U_1$.

5 Graphs of width two

For graphs of width two, I will present an algorithm that is more efficient than the dynamic-programming algorithm presented in the previous section. In earlier work, I presented an algorithm that constructs minimum-tardiness schedules for interval-ordered tasks with non-uniform deadlines on an arbitrary number of processors [20]. In Section 5.1, I will show that this algorithm constructs minimum-tardiness schedules for graphs of width two with unit-length tasks. By choosing all deadlines equal, it also constructs minimum-length schedules.

In Section 5.2, this algorithm will be adapted, such that it constructs minimum-tardiness schedules for graphs of width two with tasks of arbitrary length. It constructs minimum-length schedules if all deadlines are equal.

5.1 Unit-length tasks

For interval orders with unit-length tasks, I presented an algorithm for scheduling subject to unit-length communication delays [20]. It constructs minimum-tardiness schedules for instances $(G, m, 1\!\!1, D_0)$, where G is an interval order and $D_0 : V_G \to \mathbb{Z}^+$ is a function assigning a deadline to every task of G. In this section, I will adapt this algorithm, such that it constructs minimum-tardiness schedules for instances $(G, 2, 1\!\!1, D_0)$, where G is a graph of width two.

The algorithm consists of two parts: before assigning a starting time to every task, it computes a (smaller) deadline $D(u)$ for each task u that is not violated in a schedule in which all tasks are completed before their original deadline. Such deadlines will be called consistent. The consistent deadlines are used to assign a starting time to every task.

Throughout this section, we will use the following definition. Let (σ, π) be a feasible schedule for (G, m, μ, D_0). Let u be a task of G. If u is completed at or before its deadline, it is called *in time*. Otherwise, u is *tardy*. The *tardiness* of u equals $\max\{0, \sigma(u) + \mu(u) - D_0(u)\}$. The *tardiness* of (σ, π) is the maximum tardiness of a task of G. If all tasks are in time, then (σ, π) is called an *in-time schedule* for (G, m, μ, D_0). In that case, σ is an in-time assignment of starting times for (G, m, μ, D_0).

These definitions are used to define consistent deadlines. Consider an in-time schedule (σ, π) for $(G, m, 1\!\!1, D_0)$. Let u be a task of G. Suppose u has $k \geq 1$ successors v with $D_0(v) \leq d$. Then k successors of u are completed at time d. Suppose u starts at time t. Then the execution of u is completed at time $t + 1$. Because of communication delays, at most one successor of u starts at time $t + 1$. Hence the last of the $k - 1$ remaining successors of u is not completed before time $t + 2 + \lceil \frac{k-1}{m} \rceil$. Since (σ, π) is in time, u is completed at time $d - 1 - \lceil \frac{k-1}{m} \rceil$.

Let u_1 and u_2 be two tasks of G. Assume they have $\ell = km + 1$ common successors v with $D_0(v) \leq d$. Because u_1 and u_2 meet their deadlines, they are completed at time $d - 1 - \lceil \frac{\ell-1}{m} \rceil = d - 1 - k$. If both tasks would start at time $d - 2 - k$, then no common successor would start before time $d - k$. In that case, one of the common successors violates its deadline. So u_1 or u_2 is completed at time $d - 2 - k$.

For this reason, we will introduce deadlines for pairs of tasks. A pair of (not necessarily different) tasks (u_1, u_2) will be assigned a deadline $D(u_1, u_2)$. We will consider instances (G, m, μ, D), where $D : V_G \times V_G \to \mathbb{Z}^+$ is a function assigning a deadline to every pair of tasks of G. We will use the shorthand notation $D(u) = D(u, u)$. Let (σ, π) be a feasible schedule for an instance (G, m, μ, D) with pairwise deadlines. The pair (u_1, u_2) *meets its deadline* if $\sigma(u_1) + \mu(u_1) \leq D(u_1, u_2)$ or $\sigma(u_2) + \mu(u_2) \leq D(u_1, u_2)$. If no deadline $D(u_1, u_2)$ is violated, (σ, π) will be called an *in-time* schedule for (G, m, μ, D).

Given an instance (G, m, μ, D_0) with individual deadlines, an instance (G, m, μ, D) with pairwise deadlines can be constructed as follows. Set $D(u_1, u_2) = \min\{D_0(u_1), D_0(u_2)\}$ for every pair of tasks (u_1, u_2) of G. Then an in-time schedule for (G, m, μ, D_0) is in time for (G, m, μ, D) as well.

In a feasible schedule for $(G, m, 1\!\!1, D)$, a task is executed before its children. It is, however, possible that u is a predecessor of v and $D(u) \geq D(v)$. In that case, the deadlines are inconsistent with the precedence constraints. To define consistent instances, the following definitions are used. $N_D(u_1, u_2, d)$ equals the number of common successors v of u_1 and u_2 with $D(v) \leq d$. $P_D(u_1, u_2, d) = \max\{0, |U| - 1\}$, where U is a maximum-size subset U' of $Succ(u_1) \cap Succ(u_2)$, such that $D(v_1) \geq d + 1$ and $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in U'. Note that, in an in-time schedule for $(G, m, 1\!\!1, D)$, at most one task of such a set U starts at or after time d. Hence in every in-time schedule for $(G, m, 1\!\!1, D)$, at least $N_D(u_1, u_2, d) + P_d(u_1, u_2, d)$ common successors of u_1 and u_2 are completed at time d. For individual tasks, we use the shorthand notations $N_D(u, d) = N_D(u, u, d)$ and $P_D(u, d) = P_D(u, u, d)$.

Consider an in-time schedule for $(G, m, 1\!\!1, D)$. Let u be a task of G with $k = N_D(u, d) + P_D(u, d) \geq 1$. Since all tasks meet their deadlines, u is completed at time $d - 1 - k$. Let u_1, u_2 be two tasks of G with $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) \geq km + 1$. At least $km + 1$ common successors of u_1 and u_2 are completed at d. The first of these starts at or before time $d - 1 - k$. Because of communication delays, u_1 or u_2 is completed at time $d - 2 - k$. These observations allow the definition of instances with consistent deadlines.

Definition 5.1. *Let $(G, m, 1\!\!1, D)$ be an instance with pairwise deadlines. $(G, m, 1\!\!1, D)$ is called consistent if, for all tasks $u_1 \neq u_2$ of G and all $d \leq \max_u D(u)$,*

1. *$D(u_1, u_2) \leq \min\{D(u_1), D(u_2)\}$;*
2. *if $N_D(u_1, d) + P_D(u_1, d) \geq 1$, then $D(u_1) \leq d - 1 - \lceil \frac{1}{m}(N_D(u_1, d) + P_D(u_1, d) - 1) \rceil$;*
3. *if $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) \geq km + 1$, then $D(u_1, u_2) \leq d - 2 - k$.*

Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines. (G, m, D) is D_0-consistent if it is consistent and $D(u) \leq D_0(u)$ for all tasks u of G.

If we have two D_0-consistent instances, these can be combined in a new one with larger deadlines. Let $(G, m, 1\!\!1, D_1)$ and $(G, m, 1\!\!1, D_2)$ be two D_0-consistent instances. Define $D_{\max}(u_1, u_2) = \max\{D_1(u_1, u_2), D_2(u_1, u_2)\}$. Then it is not difficult to prove that $(G, m, 1\!\!1, D_{\max})$ is a D_0-consistent instance. This allows the definition of strongly D_0-consistent instances.

Definition 5.2. *Let $(G, m, 1\!\!1, D)$ be a D_0-consistent instance. $(G, m, 1\!\!1, D)$ is called strongly D_0-consistent if for all D_0-consistent instances $(G, m, 1\!\!1, D')$, $D(u_1, u_2) \geq D'(u_1, u_2)$ for all pairs of tasks (u_1, u_2) of G.*

The following observation is due to the definition of strongly D_0-consistent instances.

Observation 5.3. *Let $(G, m, 1\!\!1, D_0)$ and $(G, m, 1\!\!1, D_0')$ be instances with individual deadlines, such that $D_0'(u) = D_0(u) + c$ for all tasks u of G. If $(G, m, 1\!\!1, D)$ is strongly D_0-consistent and $(G, m, 1\!\!1, D')$ is strongly D_0'-consistent, then $D'(u_1, u_2) = D(u_1, u_2) + c$ for all pairs of tasks (u_1, u_2) of G.*

It is not difficult to see that the deadlines of strongly D_0-consistent instances are met in all in-time schedules for $(G, m, 1\!\!1, D_0)$.

Lemma 5.4. *Let $(G, m, \mathbb{1}, D_0)$ be an instance with individual deadlines and $(G, m, \mathbb{1}, D)$ a strongly D_0-consistent instance. Let (σ, π) be a schedule for $(G, m, \mathbb{1}, D_0)$. Then (σ, π) is an in-time schedule for $(G, m, \mathbb{1}, D_0)$ if and only if (σ, π) is an in-time schedule for $(G, m, \mathbb{1}, D)$.*

Proof. Let $(G, m, \mathbb{1}, D_0)$ be an instance with individual deadlines and $(G, m, \mathbb{1}, D)$ a strongly D_0-consistent instance. Let (σ, π) be a schedule for $(G, m, \mathbb{1}, D_0)$. If (σ, π) is an in-time schedule for $(G, m, \mathbb{1}, D)$, then (σ, π) is an in-time schedule for (G, m, D_0), because $D(u) \leq D_0(u)$ for all tasks u of G. Suppose (σ, π) is an in-time schedule for $(G, m, \mathbb{1}, D_0)$. Define $D_\sigma(u_1, u_2) = \min\{\sigma(u_1) + 1, \sigma(u_2) + 1\}$. Then $(G, m, \mathbb{1}, D_\sigma)$ is D_0-consistent. From Definition 5.2, $D(u_1, u_2) \geq D_\sigma(u_1, u_2)$ for all pairs of tasks (u_1, u_2) of G. Since every deadline $D_\sigma(u_1, u_2)$ is met, (σ, π) is an in-time schedule for $(G, m, \mathbb{1}, D)$. □

The following properties follow from the definition of strongly D_0-consistent instances.

Lemma 5.5. *Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u be a task of G. If $D(u) < D_0(u)$, then there is an integer d with $D(u) = d - 1 - \lceil \frac{1}{m}(N_D(u, d) + P_D(u, d) - 1) \rceil$ and $N_D(u, d) + P_D(u, d) \geq 1$.*

Proof. Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u be a task of G. Suppose $D(u) < D_0(u)$. Then u is not a sink. So $N_D(u, d) + P_D(u, d) \geq 1$ for some d. Suppose $D(u) < d - 1 - \lceil \frac{1}{m}(N_D(u, d) + P_D(u, d) - 1) \rceil$ for all d with $N_D(u, d) + P_D(u, d) \geq 1$. Define $D'(u_1, u_2)$ as follows. Set $D'(u_1, u_2) = D(u_1, u_2)$ for all tasks $u_1 \neq u_2$ of G, $D'(v) = D(v)$ for all tasks $v \neq u$ of G and $D'(u) = \min\{d - 1 - \lceil \frac{1}{m}(N_D(u, d) + P_D(u, d) - 1) \rceil \mid N_D(u, d) + P_D(u, d) \geq 1\}$. Then (G, m, D') is D_0-consistent and $D'(u) > D(u)$. Contradiction. So $D(u) = d - 1 - \lceil \frac{1}{m}(N_D(u, d) + P_D(u, d) - 1) \rceil$ for some d with $N_D(u, d) + P_D(u, d) \geq 1$. □

Lemma 5.6. *Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u_1 and u_2 be two tasks of G. If $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$, then there are integers d and k with $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) = km + 1$ and $D(u_1, u_2) = d - 2 - k$.*

Proof. Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u_1 and u_2 be two tasks of G. Suppose $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$. From Definition 5.1, $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) \geq km + 1$ and $D(u_1, u_2) \leq d - 2 - k$ for some d and k. Since (G, m, D) is strongly D_0-consistent, we may assume $D(u_1, u_2) = d - 2 - k$. Suppose $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) \geq km + 2$. Then $N_D(u_1, d) + P_D(u_1, d) \geq km + 2$. With Definition 5.1, $D(u_1) \leq d - 2 - k = D(u_1, u_2)$. Contradiction. Hence $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) = km + 1$ and $D(u_1, u_2) = d - 2 - k$. □

These properties allow us to prove the following results, which will be used to present the deadline modification algorithm.

Lemma 5.7. *Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u_1, u_2 be two tasks of G. If $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$, then $D(u_1) = D(u_2) = D(u_1, u_2) + 1$.*

Proof. Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u_1, u_2 be two tasks of G. Suppose $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$. From Lemma 5.6, there are integers d and k, such that $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) = km + 1$ and $D(u_1, u_2) = d - 2 - k$. Obviously, $Succ(u_1) \cap Succ(u_2) \subseteq Succ(u_1), Succ(u_2)$. Therefore $N_D(u_1, d), N_D(u_2, d) \geq N_D(u_1, u_2, d)$ and $P_D(u_1, d), P_D(u_2, d) \geq P_D(u_1, u_2, d)$. With Definition 5.1, $D(u_1), D(u_2) \leq d - 1 - k$. Since $D(u_1, u_2) < D(u_1), D(u_2)$, we find $D(u_1) = D(u_2) = d - 1 - k = D(u_1, u_2) + 1$. □

Lemma 5.7 shows that to compute strongly D_0-consistent deadlines, we only need to compute the deadline of pairs of tasks (u_1, u_2) whose deadlines are equal. If $D(u_1) \neq D(u_2)$, then $D(u_1, u_2)$ can be set to $\min\{D(u_1), D(u_2)\}$. The following result allows us to restrict this number even further.

Lemma 5.8. *Let $(G, m, \mathbb{1}, D)$ be a strongly D_0-consistent instance. Let u_1 and u_2 be two tasks of G. If $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$, then there is an integer d with, for $i \in \{1, 2\}$,*

$$N_D(u_i, d) + P_D(u_i, d) = N_D(u_1, u_2, d) + P_D(u_1, u_2, d) = (d - D(u_1) - 1)m + 1.$$

Proof. Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance. Let u_1 and u_2 be two tasks of G. Suppose $D(u_1, u_2) < \min\{D(u_1), D(u_2)\}$. With Lemma 5.5, $D(u_1, u_2) = d - 2 - k$ and $N_D(u_1, u_2, d) + P_D(u_1, u_2, d) = km + 1$ for some d and k. From Lemma 5.7, $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$. Using $D(u_1, u_2) = d - 2 - k$, we find $k = d - D(u_1) - 1$. Clearly, $N_D(u_1, d) + P_D(u_1, d), N_D(u_2, d) + P_D(u_2, d) \geq N_D(u_1, u_2, d) + P_D(u_1, u_2, d)$. Suppose $N_D(u_i, d) + P_D(u_i, d) \geq km + 2$ for $i \in \{1, 2\}$. Then $D(u_i) \leq d - 1 - \left\lceil \frac{km+1}{m} \right\rceil = d - 2 - k = D(u_1, u_2)$. Contradiction. Hence $N_D(u_1, d) + P_D(u_1, d) = N_D(u_2, d) + P_D(u_2, d) = km + 1 = (d - D(u_1) - 1)m + 1$. □

Lemmas 5.7 and 5.8 show that, for the computation of strongly D_0-consistent instances, we only need to consider pairs of tasks (u_1, u_2) with $D(u_1) = D(u_2)$ and $N_D(u_1, d) + P_D(u_1, d) = N_D(u_2, d) + P_D(u_2, d) = (d - D(u_1) - 1)m + 1$ for some $d > D(u_1)$.

Consider an instance $(G, m, 1\!\!1, D)$. $P_D(u_1, u_2, d)$ is defined in terms of sets of common successors of u_1 and u_2. Computing $P_D(u_1, u_2, d)$ coincides with finding a maximum clique in an undirected graph. The nodes of this graph are the common successors of u_1 and u_2 with deadline $d+1$; there is an edge between nodes v_1 and v_2, if $D(v_1, v_2) = d$. Since this is an NP-hard problem [11], it is unlikely that this definition allows an efficient method of determining $P_D(u_1, u_2, d)$ for arbitrary precedence graphs. The following lemmas prove that $P_D(u_1, u_2, d)$ can be computed efficiently if G is a graph of bounded width.

Lemma 5.9. *Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Then G contains at most w tasks u with $D(u) = d$.*

Proof. Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Suppose G contains $w + 1$ tasks u with $D(u) = d$. Let $u_1, \ldots, u_{w+1}$ be such tasks. Since G has width w, we may assume $u_1 \prec u_2$. From Definition 5.1, $D(u_1) \leq D(u_2) - 1$. Contradiction. So G contains at most w tasks u with $D(u) = d$. □

Corollary 5.10. *Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Let u_1, u_2 be two tasks of G. Then $P_D(u_1, u_2, d) \leq w - 1$ for all d.*

Proof. Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Let u_1, u_2 be two tasks of G. $P_D(u_1, u_2, d) = \max\{0, |U| - 1\}$, where U is a maximum-size subset U' of $Succ(u_1) \cap Succ(u_2)$, such that $D(v) \geq d + 1$ for all v in U' and $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in U'. Consider such a set U. From Lemma 5.7, $D(v) = d + 1$ for all v in U. Lemma 5.9 shows that G contains at most w tasks u with $D(u) = d + 1$. Hence $|U| \leq w$. Consequently, $P_D(u_1, u_2, d) \leq w - 1$. □

Given an instance $(G, m, 1\!\!1, D_0)$ with individual deadlines, Algorithm UET DEADLINE MODIFICATION shown in Figure 6 computes a strongly D_0-consistent instance $(G, m, 1\!\!1, D)$.

Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines and $(G, m, 1\!\!1, D)$ the instance constructed by Algorithm UET DEADLINE MODIFICATION. Using Lemmas 5.7 and 5.8 and Definition 5.2, it is not difficult to prove that $(G, m, 1\!\!1, D)$ is a strongly D_0-consistent instance.

Assume G is a transitive closure and w is the width of G. For each task u of G, Algorithm UET DEADLINE MODIFICATION computes $N_D(u, d)$ and $P_D(u, d)$ for all d. Since there are minimum-tardiness schedules of length at most n, we may assume that the maximum original deadline is at most n. The values $N_D(u, d)$ can be computed as follows: for each d, count the number of successors of u with deadline d. Compute all values $N_D(u, d)$ by a prefix sum on these numbers. Since G is a transitive closure and the maximum deadline is at most n, the values $N_D(u, d)$ can be computed in linear time for all d simultaneously.

$P_D(u, d)$ can also be calculated by traversing all successors of u: traverse all successors of u and, for each d, create a set U_d containing the successors v of u with $D(v) = d + 1$. From Lemma 5.9, every set U_d contains at most w tasks. Finding a maximum-size subset W_d of U_d, such that every pair of different tasks in W_d has deadline d takes $O(w^2 2^w)$ time. Hence computing these sets W_d takes $O(2^w w^2 n)$ time. So, for each u, all values $P_D(u, d)$ can be computed in $O(2^w w^2 n)$ time.

Algorithm UET Deadline modification
Input: An instance $(G, m, 1\!\!1, D_0)$ with individual deadlines.
Output: A strongly D_0-consistent instance $(G, m, 1\!\!1, D)$.

```
1.  let (u_1, ..., u_n) be a topologically sorted list of tasks of G
2.  D := max_u D_0(u)
3.  for i := n downto 1
4.      do D(u_i) := D_0(u_i)
5.          for d := 1 to D
6.              do if N_D(u_i, d) + P_D(u_i, d) ≥ 1
7.                     then D(u_i) := min{D(u_i), d − 1 − ⌈(1/m)(N_D(u_i, d) + P_D(u_i, d) − 1)⌉}
8.          D(u_i, u_i) := D(u_i)
9.          for j := n downto i + 1
10.             do D(u_i, u_j) := min {D(u_i), D(u_j)}
11.                D(u_j, u_i) := min {D(u_i), D(u_j)}
12.                if D(u_j) = D(u_i)
13.                   then for d := 1 to D
14.                               do if N_D(u_i, u_j, d) + P_D(u_i, u_j, d) = (d − D(u_i) − 1)m + 1
15.                                     then D(u_i, u_j) := D(u_i) − 1
16.                                          D(u_j, u_i) := D(u_i) − 1
```

Figure 6: The deadline modification algorithm for instances $(G, m, 1\!\!1, D_0)$

First, deadlines $D(u_1, u_2)$ are set to the minimum of the individual deadlines. Because of Lemma 5.7, $D(u_1, u_2)$ can only be smaller than this minimum, if $D(u_1) = D(u_2)$. Lemma 5.9 shows that there are at most $w - 1$ other tasks with deadlines equal to $D(u_1)$. Consequently, the values $N_D(u_1, u_2, d)$ and $P_D(u_1, u_2, d)$ have to be calculated for at most $(w - 1)n$ pairs of tasks. This takes $O(2^w w^2 n)$ time for each pair, so $O(2^w w^3 n^2)$ time in total.

Goralčíkova and Koubek [12] proved that the transitive closure of a directed graph G can be computed in $O(n + e + ne^-)$ time, where e^- is the number of arcs in the transitive reduction of G. Obviously, in the transitive reduction of a graph of width w, every task has at most w outgoing arcs. Consequently, the transitive reduction contains at most wn arcs. So the transitive closure can be computed in $O(wn^2)$ time and Algorithm UET Deadline modification computes strongly D_0-consistent instances in $O(2^w w^3 n^2)$ time.

Theorem 5.11. *Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines, where G is a graph of width w. Then Algorithm* UET Deadline modification *constructs a strongly D_0-consistent instance in $O(2^w w^3 n^2)$ time.*

In case G is a graph of bounded width, the strongly D_0-modified instance $(G, m, 1\!\!1, D)$ is computed in quadratic time.

Theorem 5.12. *Let w be a constant. Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines, where G is a graph of width w. Then Algorithm* UET Deadline modification *constructs a strongly D_0-consistent instance in $O(n^2)$ time.*

Consider a strongly D_0-consistent instance $(G, m, 1\!\!1, D)$. A list scheduling algorithm assigns a starting time to every task of G. The algorithm uses a list $L = (u_1, \ldots, u_n)$ of all tasks of G, such that $D(u_1) \leq \ldots \leq D(u_n)$. Such a list will be called a *priority list* for $(G, m, 1\!\!1, D)$. The list scheduling algorithm does not consider pairs of tasks. Algorithm UET List scheduling is shown in Figure 7. $par(u)$ denotes the number of unscheduled parents of u. In the beginning, $par(u)$ equals the indegree of u. The set *Ready* contains the unscheduled tasks whose parents have already been scheduled.

Algorithm UET List scheduling uses a new notion of availability. Let G be a graph. Let U be a prefix of $(G, m, 1\!\!1, D)$ and let σ be a feasible assignment of starting times for $(G', m, 1\!\!1, D)$,

where G' is the subgraph of G induced by U. Let u a source of $V_G \setminus U$. u is called *available* at time t with respect to σ, if the assignment of starting times σ_1 with $\sigma_1(v) = \sigma(v)$ for all tasks v of G' and $\sigma_1(u) = t$ is a feasible assignment of starting times for $(G'', m, 1\!\!1, D)$, where G'' is the subgraph of G induced by $U \cup \{u\}$.

Algorithm UET LIST SCHEDULING
Input: A strongly D_0-consistent instance $(G, m, 1\!\!1, D)$ and a priority list $L = (u_1, \ldots, u_n)$ for $(G, m, 1\!\!1, D)$.
Output: A feasible assignment of starting times σ for $(G, m, 1\!\!1, D_0)$.

```
1.  t := 0
2.  Ready := {u | u is a source of G}
3.  while there are unscheduled tasks
4.      do while Ready contains available tasks
5.          do let u be the available task in Ready with the smallest index in L
6.              σ(u) := t
7.              Ready := Ready \ {u}
8.              for all children v of u
9.                  do par(v) := par(v) − 1
10.                     if par(v) = 0
11.                        then Ready := Ready ∪ {v}
12.         t := t + 1
```

Figure 7: The list scheduling algorithm for strongly D_0-consistent instances

It is not difficult to see that Algorithm UET LIST SCHEDULING constructs feasible assignments of starting times for arbitrary instances $(G, m, 1\!\!1, D_0)$. Assume G is a graph of width w. Since *Ready* contains at most w elements, Algorithm UET LIST SCHEDULING can be implemented such that it constructs a feasible assignment of starting times for G in $O(w^2 n + e)$ time.

Theorem 5.13. *Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Let L be a priority list for $(G, m, 1\!\!1, D)$. Using L, Algorithm* UET LIST SCHEDULING *constructs a feasible schedule for $(G, m, 1\!\!1, D_0)$ in $O(w^2 n + e)$ time.*

If G is a graph of bounded width, Algorithm UET LIST SCHEDULING uses only linear time.

Theorem 5.14. *Let w be a constant. Let $(G, m, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width w. Let L be a priority list for $(G, m, 1\!\!1, D)$. Using L, Algorithm* UET LIST SCHEDULING *constructs a feasible schedule for $(G, m, 1\!\!1, D_0)$ in $O(n + e)$ time.*

The following theorem shows that the assignment of starting times for a strongly D_0-consistent instance $(G, 2, 1\!\!1, D)$, where G is a graph of width two constructed by Algorithm UET LIST SCHEDULING is in time for $(G, 2, 1\!\!1, D_0)$.

Theorem 5.15. *Let $(G, 2, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let L be a priority list for $(G, 2, 1\!\!1, D)$. Let σ be the assignment of starting times for $(G, 2, 1\!\!1, D)$ constructed by Algorithm* UET LIST SCHEDULING *using L. Then if there is an in-time schedule for $(G, 2, 1\!\!1, D_0)$, then σ is an in-time assignment of starting times for $(G, 2, 1\!\!1, D_0)$.*

Proof. Let $(G, 2, 1\!\!1, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let L be a priority list for $(G, 2, 1\!\!1, D)$. Let σ be the assignment of starting times for $(G, 2, 1\!\!1, D)$ constructed by Algorithm UET LIST SCHEDULING using L. Suppose there is an in-time schedule for $(G, 2, 1\!\!1, D_0)$. Let σ be the assignment of starting times for $(G, 2, 1\!\!1, D_0)$ constructed by Algorithm UET LIST SCHEDULING using L. From Lemma 5.4, we need to prove that σ is an in-time assignment of starting times for $(G, 2, 1\!\!1, D)$. Suppose σ is not in time for $(G, 2, 1\!\!1, D)$. Assume at time t, a task u_1 is executed that is contained in a pair of tasks (u_1, u_2) that violates

deadline $D(u_1, u_2)$. Assume that t is minimum. Then $\sigma(u_2) \geq \sigma(u_1) = t$ and $D(u_1, u_2) \leq t$. From Lemma 5.7, there are three possibilities: $D(u_1) \leq t$, $D(u_2) \leq t$ or $D(u_1, u_2) = t$ and $D(u_1) = D(u_2) = t + 1$.

Case 1. $D(u_1) \leq t$.

Because there is an in-time schedule for $(G, 2, \mathbb{1}, D)$, there are at most $2t$ tasks x with $D(x) \leq t$. Hence there is a time before t at which at most one task x with $D(x) \leq t$ starts. Let $t' - 1$ be the last such time. Let H_1 be the subgraph of G induced by $\bigcup_{i=t'}^{t-1} \sigma^{-1}(i) \cup \{u_1\}$. H_1 contains $2(t - t') + 1$ tasks x with $D(x) \leq t$. No task of H_1 is available at time $t' - 1$. Let u be a source of H_1. u is not available at time $t' - 1$, because (at least) one of the following statements is true.

1. A parent of u starts at time $t' - 1$.
2. Two parents of u start at time $t' - 2$.
3. A parent of u starts at time $t' - 2$ and a child $v \neq u$ of this parent starts at time $t' - 1$.

Hence every task of H_1 has a predecessor that starts at time $t' - 2$ or $t' - 1$.

Case 1.1. Every task of H_1 has a predecessor in $\sigma^{-1}(t' - 1)$.

Define $Q = \{v \in G \mid \sigma(v) = t' - 1 \wedge D(v) \leq t\}$. At time $t' - 1$, at most one task x with $D(x) \leq t$ starts. Since $D(x) \leq t$ for every task of H_1, every predecessor of a task of H_1 has a deadline at most $t - 1$. So Q contains exactly one task w. Because of communication delays, at most one successor of w is scheduled at time t'. Hence $t = t'$. As a result, w is a parent of u_1. So $N_D(w, t) \geq 1$ and $D(w) \leq t - 1 = t' - 1$. Therefore w is not completed before time $D(w)$. Contradiction.

Case 1.2. Not every task of H_1 has a predecessor in $\sigma^{-1}(t' - 1)$.

Let v be a task of H_1 without a predecessor that starts at time $t' - 1$. A predecessor w_1 of v starts at time $t' - 2$.

Case 1.2.1. $\sigma^{-1}(t' - 2)$ contains only one predecessor of H_1.

v is not available at time $t' - 1$. So a child $v' \neq v$ of w_1 starts at time $t' - 1$. Since v' is scheduled instead of v, $D(v') \leq D(v)$. Hence $N_D(w_1, t) \geq 2(t - t') + 2$. As a result, $D(w_1) \leq t' - 2$ and w_1 is not completed before time $D(w_1)$. Contradiction.

Case 1.2.2. $\sigma^{-1}(t' - 2)$ contains two predecessors of H_1.

Let w_2 be the other predecessor of H_1 executed at time $t' - 2$. Because G is a graph of width two and w_1 and w_2 are incomparable tasks, every task of H_1 is a successor of w_1 or w_2.

Case 1.2.2.1. Every task of H_1 is a successor of w_1 and w_2.

Then $N_D(w_1, w_2, t) \geq 2(t - t') + 1$. Consequently, $D(w_1, w_2) \leq t' - 2$. So (w_1, w_2) violates its deadline $D(w_1, w_2)$. Contradiction.

Case 1.2.2.2. H_1 contains a successor of w_1, that is no successor of w_2.

Let x_1 be such a task. Assume x_1 is a source of H_1. x_1 is not available at time $t' - 1$. Hence at time $t' - 1$, a child y_1 of w_1 is executed. Because y_1 is scheduled instead of x_1, $D(y_1) \leq D(x_1) \leq t$. Suppose there is a task x_2 of H_1 that is a successor of w_2, but no successor of w_1. In that case $\sigma^{-1}(t' - 1)$ contains a child y_2 of w_2, such that $D(y_2) \leq D(x_2) \leq t$. At time $t' - 1$, at most one task x with $D(x) \leq t$ is executed. So $y_1 = y_2$. In that case, σ is not a feasible assignment of starting times. Contradiction. So every task of H_1 is a successor of w_1. Hence $N_D(w_1, t) \geq 2(t - t') + 2$ and $D(w_1) \leq t' - 2$. So w_1 is not completed at time $D(w_1)$. Contradiction.

Case 1.2.2.3. H_1 contains a successor of w_2, that is no successor of w_1.

Similar to Case 1.2.2.2.

Case 2. $D(u_2) \leq t$.
Similar to Case 1.

Case 3. $D(u_1) = D(u_2) = t+1$ and $D(u_1, u_2) = t$.
In an in-time schedule for $(G, m, \mathbb{1}, D)$, u_1 or u_2 is completed at time d. Since there is an in-time schedule for $(G, m, \mathbb{1}, D)$, there are at most $2t-1$ tasks x with $D(x) \leq t$. Let $t'-1$ be the last time before t at which at most one task x with $D(x) \leq t$ is scheduled. Let H_2 be the subgraph of G induced by $\bigcup_{i=t'}^{t-1} \sigma^{-1}(i) \cup \{u_1, u_2\} \cup \{v \in \bigcup_{i \geq t} \sigma^{-1}(i) \mid v \prec u_2\}$. Then H_2 contains at least $2(t-t')+2$ tasks. Let u be a source of H_2. u is not available at time $t'-1$, because (at least) one of the following three conditions is satisfied.

1. A parent of u is scheduled at time $t'-1$.
2. Two parents of u start at time $t'-2$.
3. A parent u' of u starts at time $t'-2$ and at time $t'-1$ another child of u' is scheduled.

Thus every task of H_2 has a predecessor that starts at time $t'-2$ or $t'-1$.

Case 3.1. Every task of H_2 is a successor of $\sigma^{-1}(t'-1)$.
Define $Q = \{v \in G \mid \sigma(v) = t'-1 \wedge D(v) \leq t\}$. Clearly, Q contains one task. Let w be this task. H_2 contains at least $2(t-t')$ tasks x with $D(x) \leq t$, so $N_D(w,t) \geq 2(t-t')$. Furthermore, u_1 and u_2 are successors of w, hence $P_D(w,t) = 1$. Consequently, $D(w) \leq t'-1$. So w is not completed at time $D(w)$. Contradiction.

Case 3.2. Not every source of H_2 has a parent scheduled at time $t'-1$.
Let v be a task of H_2 that has no predecessor in $\sigma^{-1}(t'-1)$. A predecessor w_1 of v is executed at time $t'-2$.

Case 3.2.1. w_1 is the only predecessor of H_2 that starts at time $t'-2$.
v is not available at time $t'-1$. So a child $v' \neq v$ of w_1 with $D(v') \leq D(v)$ is executed at time $t'-1$. So $N_D(w_1,t) \geq 2(t-t')+1$. Hence $N_D(w_1,t)+P_D(w_1,t) \geq 2(t-t')+2$ and $D(w_1) \leq t'-2$. So w_1 is not completed at time $D(w_1)$. Contradiction.

Case 3.2.2. $\sigma^{-1}(t'-2)$ contains another predecessor w_2 of H_2.
Because G is a graph of width at most two and w_1 and w_2 are two incomparable tasks, every task of H_2 is a successor of w_1 or w_2.

Case 3.2.2.1. Every task of H_2 is a successor of w_1 and w_2.
Then $N_D(w_1, w_2, t) + P_D(w_1, w_2, t) \geq 2(t-t')+1$. Consequently, $D(w_1, w_2) \leq t'-2$. So (w_1, w_2) violates its deadline $D(w_1, w_2)$. Contradiction.

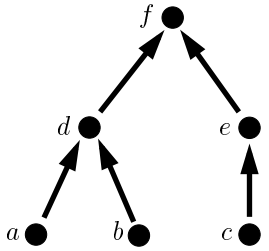
Case 3.2.2.2. H_2 contains a successor of w_1, that is no successor of w_2.
Let x_1 be such a task. We may assume that x_1 is a source of H_2. x_1 is not available at time $t'-1$. Hence at time $t'-1$ a child y_1 of w_1 is executed. y_1 is scheduled instead of x_1, so $D(y_1) \leq D(x_1) \leq t$. Since y_1 is executed at time $t'-1$, y_1 is not a child of w_2. Suppose there is a task x_2 of H_2 that is a successor of w_2, but no successor of w_1. In that case, $\sigma^{-1}(t'-1)$ contains a child y_2 of w_2 with $D(y_2) \leq D(x_2) \leq t$. y_2 is no successor of w_1, so $y_1 \neq y_2$. Consequently, two tasks x with $D(x) \leq t$ are executed at time $t'-1$. Contradiction. Therefore every task of H_2 is a successor of w_1. Hence $N_D(w_1,t)+P_D(w_1,t) \geq 2(t-t')+2$ and $D(w_1) \leq t'-2$. So w_1 is not completed at time $D(w_1)$. Contradiction.

Case 3.2.2.3. H_2 contains a successor of w_2, that is not a successor of w_1.
Similar to Case 3.2.2.2.

□

Algorithms UET List scheduling and UET Deadline modification do not construct in-time schedules for instances $(G, m, \mathbb{1}, D_0)$, where G is a graph of width $w \geq 3$ and m is at most three. In Figure 8, an instance $(G_1, 2, \mathbb{1}, D_0)$ is shown. There is an in-time schedule for

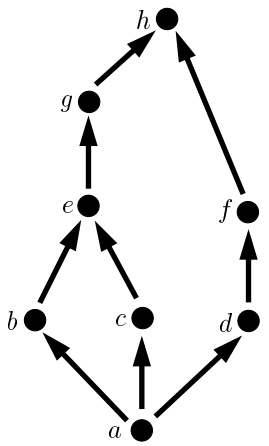
$(G_1, 2, 1\!\!1, D_0)$. For the strongly D_0-consistent instance $(G_1, 2, 1\!\!1, D)$, it is not difficult to see that $D(u) = D_0(u)$ for all tasks u of G_1. Using priority list $L_1 = (a, b, c, d, e, f)$, Algorithm UET LIST SCHEDULING constructs an assignment of starting times for $(G_1, 2, 1\!\!1, D_0)$ in which some tasks violate their deadline.



$D_0(a) = 1, D_0(b) = 2, D_0(c) = 2, D_0(d) = 3, D_0(e) = 3, D_0(f) = 4$

Figure 8: The instance $(G_1, 2, 1\!\!1, D_0)$

Figure 9 contains an instance $(G_2, 3, 1\!\!1, D_0)$. G_2 is a graph of width three. There is an in-time schedule for $(G_2, 3, 1\!\!1, D_0)$. Assume $(G_2, 3, 1\!\!1, D)$ is strongly D_0-consistent. Then, for all tasks u of G_2, $D(u) = D_0(u)$. Hence $L_2 = (a, d, b, c, e, f, g, h)$ is a priority list for $(G_2, 3, 1\!\!1, D)$. Using L_2, Algorithm UET LIST SCHEDULING constructs an assignment of starting times for $(G_2, 3, 1\!\!1, D_0)$ that is not in time.



$D_0(a) = 1, D_0(b) = 3, D_0(c) = 3, D_0(d) = 3, D_0(e) = 4, D_0(f) = 4, D_0(g) = 5, D_0(h) = 6$

Figure 9: The instance $(G_2, 3, 1\!\!1, D_0)$

Let $(G, 2, 1\!\!1, D_0)$ be an instance with individual deadlines, where G is a graph of width two. From Theorem 5.15, Algorithms UET DEADLINE MODIFICATION and UET LIST SCHEDULING construct an in-time assignment of starting times for $(G, 2, 1\!\!1, D_0)$, if such an assignment exists. If such an assignment does not exist, the algorithms construct a minimum-tardiness assignment of starting times.

Theorem 5.16. *Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines, where G is a graph of width two. Algorithms* UET DEADLINE MODIFICATION *and* UET LIST SCHEDULING *construct a minimum-tardiness assignment of starting times for $(G, m, 1\!\!1, D_0)$ in $O(n^2)$ time.*

Proof. Let $(G, m, 1\!\!1, D_0)$ be an instance with individual deadlines, where G is a graph of width two. Assume the tardiness of a minimum-tardiness schedule for $(G, m, 1\!\!1, D_0)$ equals ℓ. Consider

the instance $(G, 2, 1\!\!1, D'_0)$, where $D'_0(u) = D_0(u) + \ell$ for all tasks u of G. Then every minimum-tardiness schedule for $(G, m, 1\!\!1, D_0)$ is an in-time schedule for $(G, 2, 1\!\!1, D'_0)$. Let $(G, m, 1\!\!1, D')$ be the strongly D'_0-consistent instance constructed by Algorithm UET DEADLINE MODIFICATION and $(G, 2, 1\!\!1, D)$ a strongly D_0-consistent instance. Using Observation 5.3, we find $D(u) = D'(u) - \ell$ for all tasks u of G. Consequently, every priority list for $(G, 2, 1\!\!1, D')$ is a priority list for $(G, 2, 1\!\!1, D)$ as well. Theorem 5.15 states that, using a priority list of $(G, 2, 1\!\!1, D')$, Algorithm LIST SCHEDULING constructs an in-time assignment of starting times σ for $(G, 2, 1\!\!1, D'_0)$. σ is also feasible for $(G, 2, 1\!\!1, D_0)$. Since every task is completed at or before time $D_0(u) + \ell$, the maximum tardiness of σ (as an assignment of starting times for $(G, 2, 1\!\!1, D_0)$) is at most ℓ. Therefore σ is a minimum-tardiness assignment of starting times for $(G, 2, 1\!\!1, D_0)$. □

The algorithm can also be used to construct minimum-length schedules for instances $(G, 2, 1\!\!1)$, where G is a graph of width two.

Theorem 5.17. *Let $(G, 2, 1\!\!1)$ be an instance with G a graph of width two. Let $D_0(u) = n$ for all tasks u of G. Then the assignment of starting times for $(G, 2, 1\!\!1, D_0)$ constructed by Algorithms* UET DEADLINE MODIFICATION *and* UET LIST SCHEDULING *is a minimum-length assignment of starting times for $(G, 2, 1\!\!1)$.*

Proof. Let $(G, 2, 1\!\!1)$ be an instance with G a graph of width two. Define $D_0(u) = n$ for all tasks u of G. Let $(G, 2, 1\!\!1, D)$ be the strongly D_0-consistent instance computed by Algorithm UET DEADLINE MODIFICATION. There is an in-time schedule for $(G, 2, 1\!\!1, D_0)$. Let ℓ be the length of a minimum-length schedule for $(G, 2, 1\!\!1)$. Define $D'_0(u) = \ell$ for all tasks u of G. Then there is an in-time schedule for $(G, 2, 1\!\!1, D'_0)$. Assume $(G, 2, 1\!\!1, D')$ is the strongly D'_0-consistent instance computed by Algorithm UET DEADLINE MODIFICATION. From Observation 5.3, $D(u) = D'(u) + (n - \ell)$ for all tasks u of G. Hence any priority list for $(G, 2, 1\!\!1, D)$ is a priority list for $(G, 2, 1\!\!1, D')$. Using a priority list for $(G, 2, 1\!\!1, D')$, Algorithm UET LIST SCHEDULING constructs an in-time assignment of starting times for $(G, 2, 1\!\!1, D'_0)$. Hence it constructs such an assignment using a priority list for $(G, 2, 1\!\!1, D)$. Let σ be this assignment of starting times. Since σ is an in-time assignment of starting times for $(G, 2, 1\!\!1, D'_0)$, $\sigma(u) \leq \ell - 1$ for all tasks u of G. Hence σ corresponds to a minimum-length schedule for $(G, 2, 1\!\!1)$. □

So minimum-length schedules for instances $(G, 2, 1\!\!1)$, where G is a graph of width two can be constructed in $O(n^2)$ time. This is more efficient than the dynamic-programming algorithm presented in the previous section, which uses $O(\frac{n^2\sqrt{n}}{\sqrt{\log n}})$ time to construct a decomposition into disjoint chains. As a result, for each fixed $w \geq 2$, a minimum-length schedule for instances $(G, m, 1\!\!1)$, where G is a graph of width w can be constructed in $O(n^w)$ time. If G is a graph of width one (a single chain), a minimum-length schedule for an instance $(G, m, 1\!\!1)$ can be constructed in $O(n + e)$ time.

5.2 Arbitrary task lengths

In this section, we give a generalisation of the algorithm for scheduling instances $(G, 2, 1\!\!1)$ that was presented in Section 5.1. A straightforward way of constructing a schedule for an instance $(G, 2, \mu)$, where G is a graph of width two is splitting each task u of G in $\mu(u)$ unit-length tasks. This results in an instance $(G^{1\!\!1}, 2, 1\!\!1)$. However, the number of tasks of $G^{1\!\!1}$ equals $\sum_u \mu(u)$, which is pseudo-polynomial in the input length of the original instance $(G, 2, \mu)$. In addition, the length of a minimum-length schedule for $(G^{1\!\!1}, 2, 1\!\!1)$ is not polynomial in the length of $(G, 2, \mu)$. In order to present a polynomial-time algorithm that constructs minimum-length schedules for instances $(G, 2, \mu)$, we have to prove some properties of minimum-tardiness schedules for instances $(G^{1\!\!1}, 2, 1\!\!1)$ constructed by Algorithms UET DEADLINE MODIFICATION and UET LIST SCHEDULING and use these to construct schedules for instances $(G, 2, \mu)$. In this section, we will only consider instances $(G, 2, \mu)$, where G is a graph of width two, since Algorithms UET DEADLINE MODIFICATION and UET LIST SCHEDULING the UET-algorithm do not construct minimum-tardiness schedules for instances $(G, m, 1\!\!1)$ with G a graph of width $w \geq 3$.

Lemma 5.18. *Let $(G, 2, 1\!\!1, D)$ be an instance with strongly D_0-consistent deadlines, where G is a graph of width two. Let u_1, u_2 be two tasks of G. If u_2 is the only child of u_1, then $D(u_1) = \min\{D_0(u_1), D(u_2) - 1\}$.*

Proof. Let $(G, 2, 1\!\!1, D)$ be an instance with strongly D_0-consistent deadlines, where G is a graph of width two. Let u_1 and u_2 be tasks of G, such that u_2 is the only child of u_1. Then $N_D(u_1, D(u_2)) \geq 1$, so $D(u_1) \leq D(u_2) - 1$. We will assume that $D(u_1) \neq D_0(u_1)$. In that case, $D(u_1) = d - 1 - \lceil \frac{1}{2}(N_D(u_1, d) + P_D(u_1, d) - 1) \rceil$ for some d. If $d = D(u_2)$, then $N_D(u_1, d) \geq 1$ and $D(u_1) = D(u_2) - 1$. We may assume $d > D(u_2)$. $N_D(u_1, d) = N_D(u_2, d) + 1$, because every successor $v \neq u_2$ of u_1 is a successor of u_2 as well. In addition, $P_D(u_1, d) = P_D(u_2, d)$. Therefore

$$\begin{aligned} D(u_1) &= d - 1 - \lceil \tfrac{1}{2}(N(u_1, d) + P(u_1, d) - 1) \rceil \\ &= d - 1 - \lceil \tfrac{1}{2}(N(u_2, d) + P(u_2, d)) \rceil \\ &\geq d - 2 - \lceil \tfrac{1}{2}(N(u_2, d) + P(u_2, d) - 1) \rceil \\ &\geq D(u_2) - 1. \end{aligned}$$

So $D(u_1) = D(u_2) + 1$. As a result, $D(u_1) = \min\{D_0(u_1), D(u_2) - 1\}$. □

Consider an instance $(G, 2, 1\!\!1, D)$ with strongly D_0-consistent deadlines and G a graph of width two. Let u_1 and u_2 be two tasks of G, such that there is an arc from u_1 to u_2. If u_2 is the only child of u_1 and u_1 is the only parent of u_2, then the arc from u_1 to u_2 is called a *bridge*.

Lemma 5.19. *Let $(G, 2, 1\!\!1, D)$ be an instance with strongly D_0-consistent deadlines, where G is a graph of width two. Let σ be the assignment of starting times for $(G, 2, 1\!\!1, D)$ constructed by Algorithm* UET LIST SCHEDULING. *Let u_1 and u_2 be two tasks of G, such that u_1 is a parent u_2. If the arc from u_1 to u_2 is a bridge, then $\sigma(u_2) = \sigma(u_1) + 1$.*

Proof. Let $(G, 2, 1\!\!1, D)$ be an instance with strongly D_0-consistent deadlines, where G is a graph of width two. Let σ be the assignment of starting times for $(G, 2, 1\!\!1, D)$ constructed by Algorithm UET LIST SCHEDULING. Let u_1 and u_2 be two tasks of G. Assume u_2 is the only child of u_1 and u_1 is the only parent of u_2. Obviously, $\sigma(u_2) \geq \sigma(u_1) + 1$. Suppose $\sigma(u_2) > \sigma(u_1) + 1$. u_2 is available at time $\sigma(u_1) + 1$, but is not scheduled. At most two tasks of G are ready at time $\sigma(u_1) + 1$. Consequently, either two parents of u_2 are executed at time $\sigma(u_1)$, or another child of u_1 is scheduled at time $\sigma(u_1) + 1$. Since the arc from u_1 to u_2 is a bridge, this is impossible. So $\sigma(u_2) = \sigma(u_1) + 1$. □

The idea of constructing a schedule for instances $(G, 2, \mu, D_0)$ coincides with that of constructing schedules for instances with unit-length tasks. Let $(G^{1\!\!1}, 2, 1\!\!1, D_0^{1\!\!1})$ be the instance constructed from $(G, 2, \mu, D_0)$ by splitting every task of G into a chain of unit-length tasks. Such an instance is defined as follows. If u is a task of G, then $(u, 1), \ldots, (u, \mu(u))$ are tasks of $G^{1\!\!1}$, such that $(u, 1) \prec \ldots \prec (u, \mu(u))$ and for every arc (u_1, u_2) of G, $((u_1, \mu(u_1)), (u_2, 1))$ is an arc of $G^{1\!\!1}$. Moreover, $D_0^{1\!\!1}(u, i) = D_0(u) - \mu(u) + i$.

Deadline modification is used to compute instances $(G, 2, \mu, D)$, such that the deadlines $D(u)$ coincide with deadlines $D^{1\!\!1}$ of strongly $D_0^{1\!\!1}$-consistent instances $(G^{1\!\!1}, 2, 1\!\!1, D^{1\!\!1})$. This will allow us to construct a schedule for $(G, 2, \mu, D_0)$ that does not violate any deadline using the structure of a schedule for $(G^{1\!\!1}, 2, 1\!\!1, D_0^{1\!\!1})$.

In order to compute instances $(G, 2, \mu, D)$ with deadlines corresponding to those of strongly $D_0^{1\!\!1}$-consistent instances $(G^{1\!\!1}, 2, 1\!\!1, D^{1\!\!1})$, not only complete tasks have to be taken into account. Consider an instance $(G, 2, \mu, D)$, where G is a graph of width two. Let u be a task of G. $\mu(u, d)$ denotes the number of unit-length parts of u that are completed before time d, if u is executed at time $D(u) - \mu(u)$. More formally,

$$\begin{aligned} \mu(u, d) &= 0, && \text{if } d \leq D(u) - \mu(u) \\ &= \mu(u) - D(u) + d, && \text{if } D(u) - \mu(u) < d < D(u) \\ &= \mu(u), && \text{if } d \geq D(u). \end{aligned}$$

This is used to define $N_D(u_1, u_2, d)$.

$$N_D(u_1, u_2, d) = \sum_{u_1, u_2 \prec v} \mu(v, d).$$

$N_D(u_1, u_2, d)$ equals the total number of unit-length parts of the common successors u_1 and u_2 that are completed at time d in an in-time schedule for $(G, 2, \mu, D)$. The definition of $P_D(u_1, u_2, d)$ coincides with the one for instances $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$ with $G^{\mathbb{1}}$ a graph of width two.

$$\begin{array}{rcll} P_D(u_1, u_2, d) & = & 1, & \text{if there are two common successors } v_1 \text{ and } v_2 \text{ of } u_1 \text{ and } u_2, \\ & & & \text{such that } D(v_1, v_2) = d \text{ and } D(v_1) = D(v_2) = d + 1 \\ & = & 0, & \text{otherwise.} \end{array}$$

Again we use the shorthand notations $N_D(u, d) = N_D(u, u, d)$ and $P_D(u, d) = P_D(u, u, d)$.

The definitions of $N_D(u_1, u_2, d)$ and $P_D(u_1, u_2, d)$ can be used to define instances with strongly D_0-consistent deadlines. This definition coincides with Definition 5.2 for instances $(G, 2, \mathbb{1}, D)$.

We will present the deadline modification algorithm for instances $(G, 2, \mu, D_0)$, such that only tasks of G of length one have more than one parent or more than one child. Such a graph can be constructed from a graph with arbitrary task lengths by splitting each task u of length at least two into three tasks u^1, u^2 and u^3, such that $u^1 \prec u^2 \prec u^3$ and $\mu(u^1) = \mu(u^3) = 1$ and $\mu(u^2) = \mu(u) - 2$. Note that if $\mu(u) = 2$, then the second task does not exist. This construction takes only linear time. From now on, we will only consider this kind of graphs.

The deadline modification algorithm, Algorithm DEADLINE MODIFICATION, is similar to Algorithm UET DEADLINE MODIFICATION. It is shown in Figure 10. Like for scheduling of instances $(G, m, \mathbb{1}, D_0)$, D denotes the maximum original deadline.

Algorithm DEADLINE MODIFICATION
Input: An instance $(G, 2, \mu, D_0)$ with individual deadlines, where G is a graph of width two.
Output: An strongly D_0-consistent instance $(G, 2, \mu, D)$.

```
1.  let (u_1, ..., u_n) be a topologically sorted list of tasks of G
2.  for i := n downto 1
3.      do D(u_i) := D_0(u_i)
4.          if u_1 has exactly one child v
5.            then D(u_i) := min{D(u_i), D(v) - μ(v)}
6.            else  for d := 1 to D
7.                      do if N_D(u_i, d) + P_D(u_i, d) ≥ 1
8.                            then D(u_i) := min{D(u_i), d - 1 - ⌈½(N_D(u_i, d) + P_D(u_i, d) - 1)⌉}
9.          D(u_i, u_i) := D(u_i)
10.         for j := n downto i + 1
11.             do D(u_i, u_j) := min {D(u_i), D(u_j)}
12.                D(u_j, u_i) := min {D(u_i), D(u_j)}
13.                if D(u_j) = D(u_i)
14.                  then for d := 1 to D
15.                            do if N_D(u_i, u_j, d) + P_D(u_i, u_j, d) = 2(d - D(u_i) - 1) + 1
16.                                  then D(u_i, u_j) := D(u_i) - 1
17.                                       D(u_j, u_i) := D(u_i) - 1
```

Figure 10: The deadline modification algorithm for instances $(G, 2, \mu, D_0)$

For instances $(G, 2, \mu, D_0)$, the length of a minimum-length schedule is not polynomial in the number of tasks. If all values d are considered by the algorithm, then its running time is pseudo-polynomial in its input length. We will show that Algorithm DEADLINE MODIFICATION needs to consider only $O(n)$ different values of d.

Lemma 5.20. *Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let u be a task of G and let $d \leq D$. Then*

$$\mu(u, d-1) \leq \mu(u, d) \leq \mu(u, d-1) + 1.$$

Proof. Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let u be a task of G and let $d \leq D$.

Case 1. $d \leq D(u) - \mu(u)$.
Clearly, $\mu(u, d) = \mu(u, d-1) = 0$.

Case 2. $D(u) - \mu(u) < d < D(u)$.
$\mu(u, d) = \mu(u) - D(u) + d$. If $D(u) - \mu(u) < d - 1 < D(u)$, then $\mu(u, d-1) = \mu(u) - D(u) + d - 1 = \mu(u, d) - 1$. Otherwise, $d - 1 = D(u) - \mu(u)$ and $\mu(u, d) = \mu(u) - D(u) + d = 1$. So $\mu(u, d-1) = 0 = \mu(u, d) - 1$.

Case 3. $d \geq D(u)$.
$\mu(u, d) = \mu(u)$. If $d - 1 \geq D(u)$, then $\mu(u, d-1) = \mu(u) = \mu(u, d)$. Otherwise, $D(u) = d$ and $\mu(u, d-1) = \mu(u) - D(u) + d - 1 = \mu(u) - 1 = \mu(u, d) - 1$.

Hence $\mu(u, d) = \mu(u, d-1)$ or $\mu(u, d) = \mu(u, d-1) + 1$. □

Lemma 5.21. *Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let u_1, u_2 be two tasks of G. Let $d_1 < d_2 \leq D$. Then*

$$N_D(u_1, u_2, d_2) - N_D(u_1, u_2, d_1) \leq 2(d_2 - d_1).$$

Proof. Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let u_1, u_2 be two tasks of G. Let $d_1 < d_2 \leq D$.

$$N_D(u_1, u_2, d_2) - N_D(u_1, u_2, d_1) = \sum_{d=d_1}^{d_2-1} \left(N_D(u_1, u_2, d) - N_D(u_1, u_2, d-1) \right).$$

Suppose $N_D(u_1, u_2, d_2) - N_D(u_1, u_2, d_1) > 2(d_2 - d_1)$. Then, for some d, $d_1 \leq d \leq d_2 - 1$, $N_D(u_1, u_2, d) - N_D(u_1, u_2, d-1) \geq 3$. From Lemma 5.20, there are at least three tasks v_1, v_2, v_3, such that $\mu(v_i, d) = \mu(v_i, d-1) + 1$. $\mu(v_i, d) \geq 1$, so $d \geq D(v_i) - \mu(u_i) + 1$. Because G has width two, we may assume $v_1 \prec v_2$. Since $(G, 2, \mu, D)$ is strongly D_0-consistent, $D(v_1) \leq D(v_2) - \mu(v_2) \leq d - 1$. Hence $\mu(v_1, d-1) = \mu(v_1, d) = \mu(v_1)$. Contradiction. Therefore $N_D(u_1, u_2, d_2) - N_D(u_1, u_2, d_1) \leq 2(d_2 - d_1)$. □

Now we can prove that the deadline modification algorithm only needs to consider values d for which a task or pair of tasks with deadline d exists.

Lemma 5.22. *Algorithm* DEADLINE MODIFICATION *needs to consider only $O(n)$ values of d.*

Proof. Suppose that during the execution of the deadline modification algorithm a deadline $D(u)$ or $D(u_1, u_2)$ is decreased when a pair (u, d) or a triple (u_1, u_2, d) is considered and there is no task or pair of tasks with deadline d.

Case 1. $D(u)$ is decreased, when (u, d) is considered.
In that case, $N_D(u, d) + P_D(u, d) \geq 1$ and $D(u) > d - 1 - \left\lceil \frac{1}{2}(N_D(u, d) + P_D(u, d) - 1) \right\rceil$. There are no pairs with deadline d, hence $P_D(u, d) = 0$. We may assume that u has more than one child. Consequently, every child of u has unit length. Since $N_D(u, d) \geq 1$, there is a child v of u, such that $\mu(v, d) = 1$. Let $d' = \min_{u \prec w} D(w)$. In that case, $d > d' > D(u)$ and $N_D(u, d') \geq 1$. From Lemma 5.21, $N_D(u, d') \geq N_D(u, d) - 2(d - d')$. Obviously, $P_D(u, d') \geq P_D(u, d)$ and $N_D(u, d') + P_D(u, d') \geq 1$. Therefore

$$\begin{aligned} D(u) &\leq d' - 1 - \left\lceil \tfrac{1}{2}(N_D(u,d') + P_D(u,d') - 1) \right\rceil \\ &\leq d' - 1 - \left\lceil \tfrac{1}{2}(N_D(u,d) + P_D(u,d) - 2(d-d') - 1) \right\rceil \\ &= d' - (d'-d) - \left\lceil \tfrac{1}{2}(N_D(u,d) + P_D(u,d) - 1) \right\rceil \\ &= d - 1 - \left\lceil \tfrac{1}{2}(N_D(u,d) + P_D(u,d) - 1) \right\rceil . \end{aligned}$$

Contradiction. So $D(u)$ is not decreased, when (u,d) is considered.

Case 2. $D(u_1,u_2)$ is modified, when (u_1,u_2,d) is taken into account.
In that case, $N_D(u_1,u_2,d) + P_D(u_1,u_2,d) = 2k+1$ and $D(u_1) = D(u_2) = d-1-k$, where $k = d - D(u_1) - 1$. Since $D(u_1) = D(u_2)$, u_1 and u_2 are incomparable. We may assume $D(u_1,u_2) > d-2-k$, otherwise this modification is not necessary. Since there are no pairs of tasks with deadlines d, $P_D(u_1,u_2,d) = 0$. Suppose u_1 has a child v, such that the arc from u_1 to v is a bridge. Then v is no successor of u_2, so $N_D(u_1,d) \geq N_D(u_1,u_2,d) + 1 = 2k+2$. Therefore $D(u_1) \leq d-2-k$. So u_1 cannot have a child v, such that the arc from u_1 to v is a bridge. The same holds for u_2. So all children of u_1 and u_2 have unit length. Define $d' = \min_{u_1,u_2 \prec w} D(w)$. Clearly, $d' > D(u_1) = D(u_2)$ and $d' < d$. Using Lemma 5.21, we find $N_D(u_1,u_2,d') \geq N_D(u_1,u_2,d) - 2(d-d')$. So $N_D(u_1,u_2,d') \geq 2(d' - D(u_1) - 1) + 1$. If $N_D(u_1,u_2,d') > 2(d' - D(u_1) - 1) + 1$, then $N_D(u_1,d') \geq 2(d' - D(u_1) - 1) + 2$ and $D(u_1) \leq d' - 2 - (d' - D(u_1) - 1) = D(u_1) - 1$, which is impossible. Therefore $N_D(u_1,u_2,d') = 2(d' - D(u_1) - 1) + 1$. Consequently, $D(u_1,u_2) \leq d' - 2 - (d' - D(u_1) - 1) = D(u_1) - 1 = d - 2 - k$. Contradiction. So this modification is not necessary.

Hence Algorithm DEADLINE MODIFICATION only has to consider values d for which a task or pair of tasks with deadline d exists. Since $\min\{D(u_1), D(u_2)\} - 1 \leq D(u_1,u_2) \leq \min\{D(u_1), D(u_2)\}$, at most $2n$ values d have to be taken into account. □

This result shows that a strongly D_0-consistent instance $(G, 2, \mu, D)$ with G a graph of width two can be computed in polynomial time. Like for instances $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$, $O(n)$ tasks and pairs of tasks are taken into account. For every pair of tasks (u_1,u_2) of G, the algorithm computes values $N_D(u_1,u_2,d)$ and $P_D(u_1,u_2,d)$ for all d, such that a task or pair of tasks exists with deadline d. Computing a value $N_D(u_1,u_2,d)$ takes constant time for each common successor of u_1 and u_2, so $O(n)$ time for every d. The values $P_D(u_1,u_2,d)$ can be computed in $O(n)$ time by traversing all common successors of u_1 and u_2 and looking for tasks v_1 and v_2, such that $D(v_1) = D(v_2) = d+1$ and $D(v_1,v_2) = d$ for some d.

Obviously, the rest of the algorithm uses $O(n^2)$ time. Since the algorithm needs to consider $O(n)$ pairs of tasks, a strongly D_0-consistent instance is constructed in $O(n^3)$ time.

Theorem 5.23. *Let $(G, 2, \mu, D_0)$ be an instance with individual deadlines, where G is a graph of width two. Then Algorithm* DEADLINE MODIFICATION *computes a strongly D_0-consistent instance $(G, 2, \mu, D)$ in $O(n^3)$ time.*

Next, it will be shown that the deadlines $D(u)$ for an instance $(G, 2, \mu, D)$ constructed by Algorithm DEADLINE MODIFICATION correspond to the deadlines $D^{\mathbb{1}}(u)$ of the strongly $D_0^{\mathbb{1}}$-consistent instance $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$. We will use notations $N^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,i_1),(u_2,i_2),d)$ and $P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,i_1),(u_2,i_2),d)$ for instances $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$.

It will be shown that $D(u)$ equals $D^{\mathbb{1}}(u, \mu(u))$. This is done by proving that $N_D(u_1,u_2,d) = N^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d)$ and $P_D(u_1,u_2,d) = P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d)$ for all tasks u_1, u_2 of G and all d.

Lemma 5.24. *Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$ be a strongly $D_0^{\mathbb{1}}$-consistent instance. Let u_1, u_2 be two tasks of G. Then $D(u_1,u_2) = D^{\mathbb{1}}((u_1,\mu(u_1)),(u_2,\mu(u_2)))$.*

Proof. Let $(G, 2, \mu, D)$ be a strongly D_0-consistent instance, where G is a graph of width two. Let $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$ be a strongly $D_0^{\mathbb{1}}$-consistent instance. Let u_1, u_2 be two tasks of G. Assume for all pairs of successors (v_1, v_2) of u_1 and u_2 in G, $D(v_1,v_2) = D^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,\mu(v_1)),(v_2,\mu(v_2)))$.

Let $d \leq \max_u D(u)$. Suppose v is a common successor of u_1 and u_2. In $G^{\mathbb{1}}$, v is split into $\mu(v)$ unit-length tasks $(v,1),\ldots,(v,\mu(v))$. We need to prove that $\mu(v,d)$ equals the number of tasks of $(v,1),\ldots,(v,\mu(v))$ with deadline at most d. Since $(G,2,\mu,D)$ is strongly D_0-consistent and $(G^{\mathbb{1}},2,\mathbb{1},D^{\mathbb{1}})$ strongly $D_0^{\mathbb{1}}$-consistent, $D^{\mathbb{1}}(v,i) = D(v) - \mu(v) + i$. Hence $D^{\mathbb{1}}(v,i) \leq d$, if $i \leq d - D(u) + \mu(v)$.

Case 1. $d \leq D(v) - \mu(v)$.
$\mu(v,d) = 0$. From Lemma 5.18, $D^{\mathbb{1}}(v,i) = D(v) - \mu(v) + i \geq (d + \mu(v)) - \mu(v) + i \geq d + 1$ for all i. Consequently, $\mu(v,d)$ equals the number of tasks (v,i) with $D^{\mathbb{1}}(v,i) \leq d$.

Case 2. $D(v) - \mu(v) < d < D(v)$.
$\mu(v,d) = \mu(v) - D(v) + d$. Using Lemma 5.18, we find that $D^{\mathbb{1}}(v,i) \leq d$, if $i \leq d - D(u) + \mu(v)$. So there are exactly $\mu(v) - D(v) + d$ tasks (v,i) with $D^{\mathbb{1}}(v,i) \leq d$.

Case 3. $d \geq D(v)$.
$\mu(v,d) = \mu(d)$. Obviously, $D^{\mathbb{1}}(v,i) \leq D(v) \leq d$ for all i. Hence $\mu(v,d)$ equals the number of tasks (v,i) with $D^{\mathbb{1}}(v,i) \leq d$.

So $N_D(u_1,u_2,d) = N^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d)$. Next, it will be shown that $P_D(u_1,u_2,d) = 1$ if and only if $P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d) = 1$.

($\Rightarrow$) Suppose $P_D(u_1,u_2,d) = 1$. Then u_1 and u_2 have two common successors v_1 and v_2 in G, such that $D(v_1) = D(v_2) = d+1$ and $D(v_1,v_2) = d$. Using induction, we get $D^{\mathbb{1}}(v_1,\mu(v_1)) = D^{\mathbb{1}}(v_2,\mu(v_2)) = d+1$ and $D^{\mathbb{1}}((v_1,\mu(v_1)),(v_2,\mu(v_2))) = d$. As a result, $P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d) = 1$.

($\Leftarrow$) Suppose $P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d) = 1$. Then there are two common successors (v_1,i_1) and (v_2,i_2) of $(u_1,\mu(u_1))$ and $(u_2,\mu(u_2))$, such that $D^{\mathbb{1}}(v_1,i_1) = D^{\mathbb{1}}(v_2,i_2) = d+1$ and $D^{\mathbb{1}}((v_1,i_1),(v_2,i_2)) = d$. From Lemma 5.8, for some d', $D^{\mathbb{1}}((v_1,i_1),(v_2,i_2)) = d' - 2 - k$ and $N^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_1),(v_2,i_2),d') + P^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_1),(v_2,i_2),d') = 2k+1$, where $k = d' - d - 1$. We may assume $d' > d$. If $N^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_1),d') + P^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_1),d') \geq 2k+2$, then $D^{\mathbb{1}}(v_1,i_1) \leq d - 2 - k$. Hence $N^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_1),d') + P^{\mathbb{1}}_{D^{\mathbb{1}}}((v_1,i_2),d') = 2k+1$. The same holds for (v_2,i_2). If (v_1,i_1) has more than one immediate successor, then $i_1 = \mu(v_1)$. So assume (w,i) is the only child of (v_1,i_1) in $G^{\mathbb{1}}$, then $D^{\mathbb{1}}(w,i)$ is minimum among the deadlines of the successors of (v_1,i_1). So (w,i) is also a successor of (v_2,i_2). Hence the arc from (v_1,i_1) to (w,i) is not a bridge. So $i_1 = \mu(v_1)$ and $i_2 = \mu(v_2)$. Using induction, we find $D(v_1,v_2) = D^{\mathbb{1}}((v_1,i_1),(v_2,i_2)) = d$, $D(v_1) = D^{\mathbb{1}}(v_1,i_1) = d+1$ and $D(v_2) = D^{\mathbb{1}}(v_2,i_2) = d+1$. As a result, $P_D(u_1,u_2,d) = 1$.

So $P_D(u_1,u_2,d) = 1$ if and only if $P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1),(u_2,\mu(u_2),d) = 1$. Therefore $P_D(u_1,u_2,d) = P^{\mathbb{1}}_{D^{\mathbb{1}}}((u_1,\mu(u_1)),(u_2,\mu(u_2)),d)$ for all d. As a result, $D(u_1,u_2) = D^{\mathbb{1}}((u_1,\mu(u_1)),(u_2,\mu(u_2)))$. □

We consider instances $(G,2,\mu,D)$, where G is a graph of width two in which every task u of length at least two is divided in three parts u^1, u^2, u^3. From Lemma 5.19, u^2 and u^3 may be executed immediately after u^1. Hence we will only consider the tasks u^1 to determine a starting time for every task. Algorithm LIST SCHEDULING shown in Figure 11 assigns a starting to every task.

Obviously, the only tasks that can be executed at time t are the ready tasks. Since G is a graph of width two, a task that becomes ready at time t is executed at time t or at time $t+1$. The list scheduling algorithm only considers times t, such that a task finishes at time t or at time $t-1$. So $O(n)$ times are taken into account.

If we consider the transitive reduction of G, then the availability of a task u^1 can be checked in constant time: u^1 is available at time t, if

1. every parent of u^1 is completed at time t;
2. at most one parent of u^1 finishes at time t; and

Algorithm LIST SCHEDULING
Input: A strongly D_0-consistent instance $(G, 2, \mu, D)$, where G is a graph of width two in which only tasks of length one have more than child or more than one parent.
Output: A feasible assignment of starting times σ for $(G, 2, \mu, D)$.

```
1.  assume L = (u_1^1, ..., u_n^1) with D(u_1^1) ≤ ... ≤ D(u_n^1)
2.  t := 0
3.  while L contains unscheduled tasks
4.      do t_next := ∞
5.          for i := 1 to n
6.              do if u_i^1 is ready
7.                      then if u_i^1 is available at time t
8.                              then σ(u_i^1) := t
9.                                   σ(u_i^2) := σ(u_i^1) + μ(u_i^1)
10.                                  σ(u_i^3) := σ(u_i^2) + μ(u_i^2)
11.                                  t_next := min{t_next, σ(u_i^3) + μ(u_i^3)}
12.                             else  t_next := t + 1
13.         t := t_next
```

Figure 11: The list scheduling algorithm for instances $(G, 2, \mu, D)$

3. for all parents v of u^1, if v finishes at time t, then no other child of v starts at time t.

Since every task has at most two parents and at most two children, checking the availability of a task takes $O(1)$ time.

Hence, given the transitive reduction of G, Algorithm DEADLINE MODIFICATION constructs a feasible assignment of starting times in $O(n^2)$ time. Using the result of Goralčíkova and Koubek [12], we find that the transitive reduction of a graph of width two can be constructed in $O(n^2)$ time. Consequently, assigning starting times takes $O(n^2)$ time.

Theorem 5.25. *Let $(G, 2, \mu, D)$ be an instance with strongly D_0-consistent deadlines with G a graph of width two. Then Algorithm* LIST SCHEDULING *constructs a feasible assignment of starting times for $(G, 2, \mu, D)$ in $O(n^2)$ time.*

To prove that Algorithms DEADLINE MODIFICATION and LIST SCHEDULING construct in-time assignments of starting times for instances $(G, 2, \mu, D_0)$, it will be proved that the starting times for $(G, 2, \mu, D_0)$ coincide with the starting times for $(G^{\mathbb{1}}, 2, \mathbb{1}, D_0^{\mathbb{1}})$ computed by Algorithms UET DEADLINE MODIFICATION and UET LIST SCHEDULING. Lemma 5.24 shows that the deadlines of $(G, 2, \mu, D)$ coincide with those of $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$. In particular, $D(u^1) = D^{\mathbb{1}}(u^1, 1)$ for all tasks u of G. In addition, Algorithm UET LIST SCHEDULING constructs an assignment of starting times $\sigma^{\mathbb{1}}$ in which all tasks $(u, 1), \ldots, (u, \mu(u))$ are executed without interruption. Therefore we only need to show that $\sigma(u^1) = \sigma^{\mathbb{1}}(u^1, 1)$ for all tasks u of G, where σ is the assignment of starting times for $(G, 2, \mu, D_0)$.

Lemma 5.26. *Let $(G, 2, \mu, D_0)$ be an instance with individual deadlines, where G is a graph of width two. Let σ be the assignment of starting times for $(G, 2, \mu, D_0)$ constructed by Algorithms* DEADLINE MODIFICATION *and* LIST SCHEDULING. *There is an assignment of starting times $\sigma^{\mathbb{1}}$ for $(G^{\mathbb{1}}, 2, \mathbb{1}, D_0^{\mathbb{1}})$ constructed by Algorithms* UET DEADLINE MODIFICATION *and* UET LIST SCHEDULING, *such that $\sigma(u^1) = \sigma^{\mathbb{1}}(u^1, 1)$ for all tasks u of G.*

Proof. Let $(G, 2, \mu, D_0)$ be an instance with individual deadline with G a graph of width two. Let σ be the assignment of starting times for $(G, 2, \mu, D_0)$ constructed by Algorithms DEADLINE MODIFICATION and LIST SCHEDULING. Assume σ is constructed using list $L = (u_1^1, \ldots, u_n^1)$. In that case, $D(u_1^1) \leq \ldots \leq D(u_n^1)$. Let $L^{\mathbb{1}}$ be a priority list for the strongly D_0-consistent instance $(G^{\mathbb{1}}, 2, \mathbb{1}, D^{\mathbb{1}})$. Let $\sigma^{\mathbb{1}}$ be the assignment of starting times constructed by Algorithm UET LIST SCHEDULING using $L^{\mathbb{1}}$. Using induction, one can prove that $\sigma(u^1) = \sigma^{\mathbb{1}}(u^1, 1)$ for all tasks u of

G. This is obvious for the sources of G. Let u be a task of G, such that $\sigma(v^1) = \sigma^{\mathbb{1}}(v^1, 1)$ for all tasks v occurring before u in L. The last predecessors of u and $(u^1, 1)$ finish at the same time, so both tasks become available at the same time. Since $((u_1^1, 1), \ldots, (u_n^1, 1))$ is a sublist of $L^{\mathbb{1}}$, $\sigma(u^1) = \sigma^{\mathbb{1}}(u^1, 1)$. Using induction, we find $\sigma(u^1) = \sigma^{\mathbb{1}}(u^1, 1)$ for all tasks u of G. □

Since the schedules constructed by Algorithms DEADLINE MODIFICATION and LIST SCHEDULING coincide with those built by Algorithms UET DEADLINE MODIFICATION and UET LIST SCHEDULING, minimum-tardiness schedules for instances $(G, 2, \mu, D_0)$ with G a graph of width two can be constructed in polynomial time.

Theorem 5.27. *Let $(G, 2, \mu, D_0)$ be an instance with individual deadlines with G a graph of width two. Then Algorithms* DEADLINE MODIFICATION *and* LIST SCHEDULING *construct a minimum-tardiness schedule for $(G, 2, \mu, D_0)$ in $O(n^3)$ time.*

Consider the instance $(G, 2, \mu)$, where G is a graph of width two. Define $D_0(u) = \sum_v \mu(v)$ for all tasks u of G. Using Observation 5.3, it is not difficult to prove that the schedule for $(G, 2, \mu, D_0)$ constructed by Algorithms DEADLINE MODIFICATION and LIST SCHEDULING is a minimum-length schedule for $(G, 2, \mu)$.

Theorem 5.28. *Let $(G, 2, \mu)$ be an instance with G a graph of width two. Let $D_0(u) = \sum_v \mu(v)$ for all tasks u of G. Then the assignment of starting times for $(G, 2, \mu, D_0)$ constructed by Algorithms* DEADLINE MODIFICATION *and* LIST SCHEDULING *is a minimum-length assignment of starting times for $(G, 2, \mu)$.*

6 An NP-completeness result

In the previous section, a polynomial-time algorithm for constructing minimum-length schedules for instances $(G, 2, \mu)$ with G a graph of width two was presented. In this section, it will be shown that if G has width $w \geq 3$, then constructing a minimum-length schedule for $(G, 2, \mu)$ is NP-hard. This is done using a reduction from PARTITION. PARTITION is the following well-known problem [11].

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

$$\sum_{a \in A'} a = \sum_{a \in A \setminus A'} a?$$

PARTITION is an NP-complete problem [11]. Let WIDTH3ON2 be the following problem.

Problem. WIDTH3ON2
Instance. An instance $(G, 2, \mu)$, where G is a graph of width three and a positive integer B.
Question. Is there a schedule for $(G, 2, \mu)$ of length at most B?

Using a reduction from PARTITION, it will be shown that WIDTH3ON2 is an NP-complete problem.

Lemma 6.1. *There is a polynomial reduction from* PARTITION *to* WIDTH3ON2.

Proof. Let $A = \{a_1, \ldots, a_n\}$ be an instance of PARTITION. Define $N = \sum_{a \in A} a$ and $M = N + 1$. Construct an instance $(G, 2, \mu)$ as follows. G is a graph consisting of three chains. The first two chains, C_1 and C_2, consist of $n+1$ tasks c_{1i}, c_{2i} of length $\mu(c_{ji}) = M$, such that $c_{j0} \prec \ldots \prec c_{jn}$. The third chain, C_3, consists of n tasks $u_1, \ldots, u_n$ with lengths $\mu(u_i) = a_i$ and precedence constraints $u_1 \prec \ldots \prec u_n$. Let $B = \frac{1}{2}N + (n+1)M$. Now we can prove that A can be divided into two disjoint subsets with equal sum if and only if there is a schedule for $(G, 2, \mu)$ of length at most B.

($\Rightarrow$) Suppose there is a subset A_1 of A, such that $\sum_{a\in A_1} a = \sum_{a\in A\setminus A_1} a$. Define $A_2 = A \setminus A_1$. For each i, such that $a_i \in A_p$, set

$$\pi(u_i) = p \text{ and } \sigma(u_i) = iM + \sum_{j<i:a_j\in A_p} a_j.$$

Furthermore, set $\pi(c_{1i}) = 1$ and $\pi(c_{2i}) = 2$ for all i and

$$\sigma(c_{1i}) = iM + \sum_{j\le i:a_j\in A_1} a_j \text{ and } \sigma(c_{2i}) = iM + \sum_{j\le i:a_j\in A_2} a_j.$$

Clearly, $\sigma(c_{ji+1}) \ge \sigma(c_{ji}) + M$ for all i, j, $0 \le i \le n$, $j = 1, 2$. Furthermore, for all i, $0 \le i \le n-1$, such that $u_{i+1} \in A_p$,

$$\begin{array}{rcl} \sigma(u_{i+1}) & = & (i+1)M + \sum_{j<i+1:a_j\in A_p} a_j \\ & \ge & iM + M \\ & > & iM + a_i + \sum_{j<i} a_j \\ & \ge & \sigma(u_i) + \mu(u_i). \end{array}$$

So $\sigma(u_{i+1}) > \sigma(u_{i-1}) + \mu(u_{i-1})$ for all i. Therefore (σ, π) is a feasible schedule for $(G, 2, \mu)$. For all i, $1 \le i \le n$, such that $u_i \in A_p$,

$$\begin{array}{rcl} \sigma(u_i) & = & iM + \sum_{j<i:a_j\in A_p} a_j \\ & = & (i-1)M + \sum_{j<i:a_j\in A_p} a_j + M \\ & = & \sigma(c_{pi-1}) + \mu(c_{pi-1}), \end{array}$$

and

$$\begin{array}{rcl} \sigma(u_i) + \mu(u_i) & = & iM + \sum_{j<i:a_j\in A_p} a_j + a_i \\ & = & iM + \sum_{j<i+1:a_j\in A_p} a_j \\ & = & \sigma(c_{pi}). \end{array}$$

So if u_i is executed on processor p, it is executed immediately after c_{pi-1} and immediately before c_{pi}. So there is no idle time in the schedule. Hence the last task of G is completed at time

$$\frac{1}{2}\left(2(n+1)M + \sum_{a\in A} a\right) = (n+1)M + \frac{1}{2}N = B.$$

($\Leftarrow$) Suppose (σ, π) is a schedule for $(G, 2, \mu)$ of length at most B. Processor p can execute at most $n+1$ tasks c_{ji}, otherwise the schedule length is at least $(n+2)M > (n+1)M + \sum_{a\in A} a > B$. So both processors execute exactly $n+1$ tasks of length M. The sum of the execution lengths of all tasks of G is $2B$ so there is no idle time in (σ, π). Define

$$A_1 = \{a_i \mid \pi(u_i) = 1\} \text{ and } A_2 = \{a_i \mid \pi(u_i) = 2\}.$$

Since (σ, π) has no idle times,

$$\sum_{a\in A_1} a = B - (n+1)M = \frac{1}{2}\sum_{a\in A} a.$$

The sum of the lengths of the tasks of G is $O(\sum_{a\in A} a)$, so this reduction requires time and space polynomial in the length of the instance of PARTITION. So there is a polynomial reduction from PARTITION to WIDTH3ON2. □

Corollary 6.2. WIDTH3ON2 *in NP-complete.*

Proof. Clearly, WIDTH3ON2 is an element of NP. From Lemma 6.1, WIDTH3ON2 is an NP-complete problem. □

It is easy to see that the same proof can be used to prove that constructing minimum-length schedules for instances $(G, 2, \mu)$ without communication delays, where G is a graph of width three is NP-hard as well. Moreover, this proof can be generalised for instances (G, m, μ), where G is a graph of fixed width $w \geq 3$ and $m < w$.

Theorem 6.3. *Let $w \geq 3$ and $2 \leq m < w$. Constructing a minimum-length schedule for an instance (G, m, μ), where G is a graph of width w is NP-hard.*

In the previous section, I proved that a minimum-length schedule for an instance $(G, 2, \mu)$ with G a graph of width two can be constructed in polynomial time. The complexity of constructing such schedules for instances (G, w, μ), where G is a graph of width $w \geq 3$ remains open. We will deal with this problem in the next section.

7 Dynamic programming for arbitrary-length tasks

In this last section, it will be shown that a minimum-length schedule for instances (G, w, μ) with G a graph of constant width w can be constructed in polynomial time. Like in Section 4, we will use a dynamic-programming approach.

Consider an instance (G, w, μ), where G is a graph of width w. In a schedule (σ, π) for (G, w, μ), at most w tasks can be executed simultaneously. As a result, any feasible assignment of starting times for (G, ∞, μ) is also feasible for (G, w, μ). Chrétienne and Picouleau [4] proved that there exists a minimum-length schedule for (G, ∞, μ) in which incomparable tasks are executed on different processors.

An assignment of starting times σ for (G, ∞, μ) is called *greedy*, if for all times t, no task u with $\sigma(u) > t$ can start at time t without violating the feasibility of σ. For every assignment of starting times σ for (G, ∞, μ), there is a greedy assignment σ', whose length is at most that of σ. Hence there is a minimum-length schedule (σ, π) for (G, ∞, μ), such that σ is a greedy assignment of starting times and incomparable tasks are executed on different processors. For such a schedule, the number of potential starting times of a task is bounded.

Let $est(u)$ denote the earliest starting time of a task u in a schedule for (G, ∞, μ) without communication delays. In other words,

$$\begin{array}{rcll} est(u) & = & 0, & \text{if } u \text{ is a source of } G, \\ & = & \max\limits_{v \prec u} (est(v) + \mu(v)), & \text{otherwise.} \end{array}$$

In schedules (σ, π), such that σ is a greedy assignment of starting times and $\pi(u_1) \neq \pi(u_2)$ for all incomparable tasks u_1, u_2, a task u is executed within the first n time slots after $est(u)$.

Lemma 7.1. *Let (σ, π) be a schedule for (G, ∞, μ), such that σ is a greedy assignment of starting times and $\pi(u_1) \neq \pi(u_2)$ for all incomparable tasks u_1, u_2 of G. Then, for all tasks u of G,*

$$est(u) \leq \sigma(u) \leq est(u) + n - 1.$$

Proof. Let (σ, π) be a schedule for (G, ∞, μ), such that σ is a greedy assignment of starting times and $\pi(u_1) \neq \pi(u_2)$ for all incomparable tasks u_1, u_2 of G. Obviously, $\sigma(u) \geq est(u)$. Denote by $lpp(u)$ the maximum number of predecessors of u on a path to u. Hence

$$\begin{array}{rcll} lpp(u) & = & 0, & \text{if } u \text{ is a source of } G, \\ & = & \max\limits_{v \prec u} (lpp(v) + 1), & \text{otherwise.} \end{array}$$

We can prove that $\sigma(u) \leq est(u) + lpp(u)$ for all tasks u of G. This is obvious for the sources of G. Suppose $\sigma(v) \leq est(v) + lpp(v)$ for all predecessors v of u. Obviously, a task is executed at most one time slot after the completion of the last of its predecessors. Consequently,

$$\begin{aligned}\sigma(u) &\leq \max_{v \prec u}(\sigma(v)+\mu(v)+1)\\ &\leq \max_{v \prec u}(est(v)+lpp(v)+\mu(v)+1)\\ &\leq \max_{v \prec u}(est(v)+\mu(v))+\max_{v \prec u}(lpp(v)+1)\\ &= est(u)+lpp(u).\end{aligned}$$

Clearly, $lpp(u) \leq n-1$. So $est(u) \leq \sigma(u) \leq est(u)+n-1$. □

So there is a minimum-length schedule for (G, ∞, μ), in which every task has at most n possible starting times. Consequently, in a minimum-length schedule for (G, w, μ), where G is a graph of width w, each task has at most n potential starting times. It is easy to see that Lemma 7.1 is not true for instances (G, m, μ), such that m is smaller than the width of G.

The limited number of potential starting times will be used to present a dynamic-programming algorithm. Let U be a prefix of (G, ∞, μ) and σ_U a feasible assignment of starting times for (G', ∞, μ), where G' is the subgraph of G induced by U and $\sigma_U(u) < t$ for all u in U. Let U' be a set of sources of $G \setminus U$. The set of tasks U' is called *available* at time t with respect to U and σ_U, if

1. $\sigma(v)+\mu(v) \leq t$ for all $v \prec u$, $u \in U'$;
2. for every u in U', at most one predecessor of u finishes at time t; and
3. every task in U with completion time t has at most one successor in U'.

Note that the availability of U' depends only on the starting times of the sinks of U: since $\sigma_U(u) < t$ for all u in U, a task of U that finishes at time t is a sink of U.

Consider an instance (G, ∞, μ). A *timed prefix* of (G, ∞, μ) is a pair (U, σ_U), where U is a prefix of (G, ∞, μ) and σ_U is a feasible assignment of starting times for (G', ∞, μ), where G' is the subgraph of G induced by U and $est(u) \leq \sigma_U(u) \leq est(u)+n-1$ for all u in U.

Let (U_1, σ_{U_1}) and (U_2, σ_{U_2}) be two timed prefixes of (G, ∞, μ) with $U_1 \subsetneq U_2$. (U_2, σ_{U_2}) is called *available* with respect to (U_1, σ_{U_1}), if there is a time t, such that

1. $\sigma_{U_1}(u) < t$ for all u in U_1;
2. $U_2 \setminus U_1$ is available at time t with respect to U_1 and σ_{U_1};
3. $\sigma_{U_2}(u) = \sigma_{U_1}(u)$ for all u in U_1;
4. $\sigma_{U_2}(u) = t$ for all u in $U_2 \setminus U_1$; and
5. $est(u) \leq t \leq est(u)+n-1$ for all u in $U_2 \setminus U_1$.

Let $Av(U, \sigma_U)$ denote the collection of timed prefixes of (G, ∞, μ) that are available with respect to (U, σ_U).

Let σ be a feasible greedy assignment of starting times for (G, ∞, μ). Let $t \leq \max_u \sigma(u)$. Define $U_1 = \{u \in G \mid \sigma(u) < t\}$ and $U_2 = \{u \in G \mid \sigma(u) \leq t\}$. Let σ_{U_1} and σ_{U_2} be the restrictions of σ to U_1 and U_2. Obviously, (U_2, σ_{U_2}) is available at time t with respect to (U_1, σ_{U_1}). Hence we only need to consider timed prefixes of (G, ∞, μ).

Let (U, σ_U) be a timed prefix of (G, ∞, μ). $L(U, \sigma_U)$ represents the length of a minimum-length schedule (σ, π) for (G, ∞, μ), such that $\sigma(u) = \sigma_U(u)$ for all u in U. Then

$$L(V_G, \sigma_{V_G}) = \max_{u \in G} \sigma_{V_G}(u) + \mu(u)$$

and

$$L(U, \sigma_U) = \min_{(U', \sigma_{U'}) \in Av(U, \sigma_U)} L(U', \sigma_{U'}).$$

$L(\varnothing, \sigma_\varnothing)$ corresponds to the length of a minimum-length schedule for (G, ∞, μ).

Consider an instance (G, ∞, μ), where G is a graph of width w with chain decomposition $C_1, \ldots, C_w$. Assume $C_i = \{c_{i1}, \ldots, c_{i\ell_i}\}$ with $c_{i1} \prec \ldots \prec c_{i\ell_i}$. Like in Section 4, a prefix of (G, ∞, μ) can be represented by a sequence $(b_1, \ldots, b_w)$ with $0 \leq b_i \leq \ell_i$. A timed prefix (U, σ_U) of (G, ∞, μ) can be represented by a tuple $(b_1, t_1, \ldots, b_w, t_w)$, where $(b_1, \ldots, b_w)$ corresponds to U and $t_i = \sigma_U(c_{ib_i})$, if $b_i \neq 0$. Otherwise, $t_i = -\infty$.

Moreover, suppose $0 \leq b_i \leq \ell_i$ and $est(c_{ib_i}) \leq t_i \leq est(c_{ib_i}) + n - 1$ for all i with $b_i > 0$ and $t_i = -\infty$ for i, such that $b_i = 0$, then the sequence $(b_1, t_1, \ldots, b_w, t_w)$ coincides with the timed prefix $(\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}, \sigma)$, where σ is a (feasible) assignment of starting times for the instance (G', ∞, μ), where G' is the subgraph of G induced by $\bigcup_{i=1}^{w} \{c_{i1}, \ldots, c_{ib_i}\}$ and $\sigma(c_{ib_i}) = t_i$ for all i with $b_i \neq 0$.

To implement the computation of $L(\varnothing, \sigma_\varnothing)$, a table T of dimension $2w$ is built. T contains entries $T[b_1, t_1, \ldots, b_w, t_w]$, where $0 \leq b_i \leq \ell_i$ and $est(c_{ib_i}) \leq t_i \leq est(c_{ib_i}) + n - 1$ or $t_i = -\infty$. We start by setting $T[b_1, t_1, \ldots, b_w, t_w] = \infty$ for all sequences. Algorithm TABLE CONSTRUCTION shown in Figure 12 computes $L(U, \sigma_U)$. Like in Section 4, $seq(U, \sigma_U)$ represents the tuple that corresponds to the timed prefix (U, σ_U).

Algorithm TABLE CONSTRUCTION(U, σ_U)
Input: An instance (G, ∞, μ), where G is a graph of width w with chain decomposition $C_1, \ldots, C_w$ and a timed prefix (U, σ_U) of (G, ∞, μ).
Output: A table T with $T[seq(U, \sigma_U)] = L(U, \sigma_U)$.

```
1.  if T[seq(U, σ_U)] = ∞
2.     then if U = V_G
3.             then T[seq(U, σ_U)] := max_u σ_U(u) + μ(u)
4.             else for (U', σ_U') in Av(U, σ_U)
5.                     do TABLE CONSTRUCTION(U', σ_U')
6.                  T[seq(U, σ_U)] := min{T[seq(U', σ_U')] | (U', σ_U') ∈ Av(U, σ_U)}
```

Figure 12: The algorithm computing $L(U, \sigma_U)$

By applying Algorithm TABLE CONSTRUCTION to the timed prefix $(\varnothing, \sigma_\varnothing)$, we construct a table T with $T[0, -\infty, \ldots, 0, -\infty] = L(\varnothing, \sigma_\varnothing)$. In addition, $T[seq(U, \sigma_U)] = L(U, \sigma_U)$ for all timed prefixes (U, σ_U) of (G, ∞, μ) for which a greedy schedule (σ, π) exists with $\sigma(u) = \sigma_U(u)$ for all u in U. Using these values, Algorithm SCHEDULE CONSTRUCTION constructs a minimum-length assignment of starting times for (G, ∞, μ).

Consider an instance (G, w, μ), where G is a graph of width w with a chain decomposition $C_1, \ldots, C_w$. For a timed prefix (U, σ_U) of (G, w, μ), Algorithm TABLE CONSTRUCTION determines $Av(U, \sigma_U)$. Since timed prefixes are represented by sequences, the potential elements of this set can be computed in $O(2^w n)$ time as follows. If $(U', \sigma_{U'})$ is available with respect to (U, σ_U), then $U' \setminus U$ is a set of sources of $G \setminus U$, and for some t, $\sigma_{U'}(u) = t$ for all $u \in U' \setminus U$ and $\sigma_{U'}(u) = \sigma_U(u)$ for all u in U. In addition, $est(u) \leq t \leq est(u) + n - 1$ for all u in $U' \setminus U$. Assume U is represented by $(b_1, \ldots, b_w)$, then U' corresponds to a sequence $(b_1 + x_1, \ldots, b_w + x_w)$ with $x_i \in \{0, 1\}$. So computing all potential sets takes $O(2^w)$ time. For every set, there are at most n possible starting times t. Checking the availability of a potential element takes $O(w^2)$ time. Consequently, Algorithm TABLE CONSTRUCTION uses $O(w^2 2^w n)$ time for a timed prefix (U, σ_U) of (G, w, μ).

A timed prefix coincides with a tuple $(b_1, t_1, \ldots, b_w, t_w)$ with $0 \leq b_i \leq \ell_i$ and $est(c_{ib_i}) \leq t_i \leq est(c_{ib_i}) + n - 1$, if $b_i > 0$ and $t_i = -\infty$, otherwise. Therefore the number of timed prefixes

Algorithm SCHEDULE CONSTRUCTION
Input: An instance (G, ∞, μ) and a table T, such that $T[seq(U, \sigma_U)] = L(U, \sigma_U)$ for all timed prefixes (U, σ_U) of (G, ∞, μ) that might occur in a greedy schedule for (G, ∞, μ).
Output: A minimum-length assignment of starting times σ for (G, ∞, μ).

```
1.  t := 0
2.  U := ∅
3.  σ := ∅
4.  while U ≠ V_G
5.      do determine (U', σ_U') in Av(U, σ_U) with minimum T[seq(U', σ_U')]
6.          for u ∈ U' \ U
7.              do σ(u) := t
8.          t := t'
9.          U := U'
```

Figure 13: The algorithm constructing a minimum-length schedule

considered by Algorithm TABLE CONSTRUCTION is at most

$$\prod_{i=1}^{w}(n\ell_i + 1) \;\leq\; 2^w n^w \prod_{i=1}^{w} \ell_i \;\leq\; 2^w n^w \prod_{i=1}^{w} \frac{n}{w} \;\leq\; n^{2w}.$$

As a result, Algorithm TABLE CONSTRUCTION determines the length of an optimal schedule for (G, w, μ) in $O(w^2 2^w n^{2w+1})$ time.

Algorithm SCHEDULE CONSTRUCTION starts with timed prefix $(\varnothing, \sigma_\varnothing)$. For the timed prefix (U, σ_U), it determines the best timed prefix $(U', \sigma_{U'})$ in $Av(U, \sigma_U)$ and continues with $(U', \sigma_{U'})$. Then $U \subsetneq U'$, so at most n sets $Av(U, \sigma_U)$ have to be computed by Algorithm SCHEDULE CONSTRUCTION. So it uses $O(w^2 2^w n^2)$ time to build an optimal schedule for G.

Theorem 7.2. *Let (G, w, μ) be an instance, such that G is a graph of width w. Then Algorithms* CHAIN DECOMPOSITION, TABLE CONSTRUCTION *and* SCHEDULE CONSTRUCTION *construct a minimum-length assignment of starting times for (G, w, μ) in $O(w^2 2^w n^{2w+1})$ time.*

For every fixed w, Algorithms TABLE CONSTRUCTION and SCHEDULE CONSTRUCTION construct optimal schedules in polynomial time.

Theorem 7.3. *Let w be a constant. Let (G, w, μ) be an instance, where G is a graph of width w. Then Algorithms* CHAIN DECOMPOSITION, TABLE CONSTRUCTION *and* SCHEDULE CONSTRUCTION *construct a minimum-length assignment of starting times for (G, w, μ) in $O(n^{2w+1})$ time.*

It is not difficult to see that the dynamic-programming approach can be used to construct schedules that are optimal with respect to other objective functions. Unlike the dynamic-programming algorithm presented in Section 4, it cannot be used for other scheduling models.

For example, if the maximum communication delay exceeds the smallest task length, then there need not be a minimum-length schedule in which incomparable tasks are executed on different processors. Hence the number of potential starting times of a task is not polynomial in the number of tasks. In that case, the running time of the dynamic-programming algorithm is no longer polynomial in the number of tasks.

In addition, the dynamic-programming algorithm cannot be used if the processor assignment is known in advance. Consider an instance (G, ∞, μ, π) in which, for every task u, the processor assignment $\pi(u)$ is known in advance. In that case, the number of possible starting times in a greedy schedule for (G, ∞, μ, π) is not polynomial in the number of tasks of G. As a result, the dynamic-programming approach is a pseudo-polynomial algorithm. It is unlikely that there is a polynomial-time algorithm that constructs minimum-length schedules for instances (G, ∞, μ, π): Sotskov and Shakhlevich [17] proved that constructing a minimum-length schedule on three processors for a job shop with three jobs is an NP-hard problem. This problem is a special case of

the problem of constructing minimum-length a schedule for an instance $(G, 3, \mu, \pi)$, where G is a graph of width three.

Concluding remarks

In this report, we studied the problem of scheduling graphs of bounded width with the objective of minimising the makespan. An overview of the results is given in the following table. Consider an instance (G, m, μ). The following notations are used. w denotes the width of G, $\mu(G) = \sum_{u \in V_G} \mu(u)$ and c is a positive integer.

	unit-length tasks	$\mu(G) = O(n^c)$	arbitrary task lengths
$w = 1$	$O(n + e)$	$O(n + e)$	$O(n + e)$
$w = 2$	$O(n^2)$	$O(n^{\min\{2c,3\}})$	$O(n^3)$
$3 \leq w \leq m$	$O(n^w)$	$O(n^{\min\{cw,2w+1\}})$	$O(n^{2w+1})$
$w > m$	$O(n^w)$	$O(n^{cw})$	NP-hard

Note that, in order to achieve the running times for instances (G, m, μ), such that the sum of the execution times is bounded by a polynomial $p(n)$, one of the algorithms presented in this report is used. Which algorithm to use depends on the degree of the polynomial p.

The algorithms for scheduling instances $(G, m, 1\!\!1)$ (Algorithms UET Deadline modification and UET List scheduling or Algorithms UET Table construction and UET Schedule construction) have to be applied on the graph in which every task is split into a chain of unit-length tasks. The dynamic-programming algorithm, Algorithms UET Table construction and UET Schedule construction, have to use a different definition of availability, since the tasks of a chain corresponding to one task in the original graph have to be executed on one processor without interruption. This alternative definition does not increase the complexity of the algorithms.

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):301–307, October 1995.

[2] H. Alt, N. Blum, K. Mehlhorn and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Information Processing Letters*, 37(4):237–240, February 1991.

[3] H.L. Bodlaender and M.R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. Technical Report UU-CS-1994-14, Department of Computer Science, Utrecht University, 1994.

[4] 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*, pages 65–90. John Wiley & Sons, 1995.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via algorithmic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[6] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[7] R.P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, January 1950.

[8] R.G. Downey and M.R. Fellows. Fixed-parameter intractability and completeness I: basic results. *SIAM Journal on Computing*, 24(2):873–921, August 1995.

[9] L. Finta, Z. Liu, I. Milis and E. Bampis. Scheduling UET-UCT series-parallel graphs on two processors. Technical Report RR 2566, INRIA, Sophia-Antipolis, France, May 1995.

[10] D.R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. *Proceedings of the AMS*, 7:701–702, 1956.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[12] A. Goralčíkova and V. Koubek. A reduct-and-closure algorithm for graphs. In J. Bečvář, editor, *Mathematical Foundations of Computer Science 1979*, number 74 in Lecture Notes in Computer Science, pages 301–307, Berlin, 1979. Springer-Verlag.

[13] J.E. Hopcroft and R.M. Karp. A $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

[14] J.K. Lenstra, M. Veldhorst and B. Veltman. The complexity of scheduling trees with communication delays. *Journal of Algorithms*, 20(1):157–173, January 1996.

[15] R.H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–194, Dordrecht, the Netherlands, 1989. Kluwer Academic Publishers.

[16] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18:55–71, 1987.

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

[18] T.A. Varvarigou, V.P. Roychowdhury, T. Kailath and E. Lawler. Scheduling in and out forests in the presence of communication delays. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1065–1074, October 1996.

[19] B. Veltman. *Multiprocessor scheduling with communication delays*. PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 1993.

[20] J. Verriet. Scheduling interval orders with release dates and deadlines. Technical Report UU-CS-1996-12, Department of Computer Science, Utrecht University, March 1996. This report has been revised and submitted for publication in Parallel Computing.