Scheduling interval orders with release dates and deadlines

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

We study the problem of scheduling unit-length tasks with release dates and deadlines subject to precedence constraints and unit communication delays. Two polynomial-time algorithms are defined: one constructs schedules for graphs with uniform release dates, the other for graphs with arbitrary release dates. They have a special structure: unlike most scheduling algorithms, they do not consider individual tasks, but pairs of tasks. It is proved that the algorithms find minimum-lateness schedules for interval orders on an arbitrary number of processors.

1 Introduction

Finding a shortest schedule for a precedence graph is a very difficult problem: deciding whether a schedule for a set of tasks of length at most D exists on an arbitrary number of processors is an NP-complete problem, even without precedence constraints. The problem remains NP-complete if all task lengths are equal and the number of processors is arbitrary [11]. Only for special classes of precedence relations there are polynomial-time algorithms that find minimum-length schedules [8, 10].

In real computer architectures, a large delay occurs between the execution of dependent tasks on different processors. If these communication delays are taken into account, finding a schedule of minimum length is computationally intractable, even under tree precedence constraints [9]. For interval orders, however, Ali and El-Rewini [1, 2] defined a polynomial-time algorithm that takes communication delays into account and finds a minimum-length schedule.

In this report we consider the problem of scheduling unit length tasks subject to precedence constraints, communication delays and individual release dates and deadlines. On one hand, this is a generalisation of scheduling precedence graphs with release dates and deadlines [6, 7]. On the other, scheduling with a uniform deadline subject to communication delays [2, 3, 9] is a special case of this problem.

The problem has been considered by Verriet [12]. His algorithms have the same structure as those of Garey and Johnson [6, 7] for scheduling without communication delays: first the individual deadlines are modified, and second the tasks are scheduled by a list scheduling algorithm applied to the set of tasks ordered by non-decreasing modified deadlines. The precise deadline modifications of Verriet [12] and Garey and Johnson [6, 7] of a task u depend on the subgraph of successors of u, but not on the predecessors of u. For the case of two processors without communication delays [6, 7], this turns out to be sufficient: the algorithms find minimum-lateness schedules. In case of two processors with communication delays, Verriet was only able to solve the problem for graphs satisfying the least urgent parent property. This restriction is closely related to the two basic patterns in which communication delays have an impact on scheduling.

Successor pattern If u has immediate successors $v_1, \ldots, v_k$, then at most one of these can be executed immediately after u. It is unknown which one.

Predecessor pattern If u has immediate predecessors $w_1, \ldots, w_l$, then only one can be executed immediately before u. Which predecessor is unknown.

Restriction to (series-parallel) graphs with the least urgent parent property answers the question associated with the predecessor pattern at the beginning. The question corresponding to the successor pattern is dealt with in the deadline modification stage.

In this report we will take both patterns into account using a different approach to the deadline modification. We will assign a deadline to pairs (w_i, w_j) of immediate predecessors of u. Either w_i or w_j has to be completed before this deadline. Hence if (w_i, w_j) has a deadline strictly smaller than the deadlines of both w_i and w_j, one of these should be scheduled earlier than required by its own deadline.

After some preliminary definitions we will present two algorithms: one finds schedules for graphs with a uniform release date, the other for graphs with arbitrary release dates. They first compute a deadline for every pair of tasks, after which every task is assigned a starting time using a list scheduling algorithm. We will show that application of these algorithms to interval orders yields minimum-lateness schedules, even for an arbitrary number of processors.

2 Preliminary definitions

We will consider precedence graphs in which every task has unit processing length. Each task has to be executed without interruption on one processor. An edge (u, v) denotes a data dependency between nodes u and v: to be able to execute v, the result of the computation of u must be known. If u and v are executed on different processors, it is necessary to send data from one processor to another. This takes unit time during which both the sending and the receiving processor can execute another task. If u and v are executed on the same processor, no communication delay occurs.

Throughout this report, G will denote a precedence graph in which every task u has a deadline $D(u)$ and possibly a release date $R(u)$. Both are assumed to be non-negative integers. If a task has only a deadline, its release date is considered zero.

We will assume G contains n nodes and e edges. Let u, v be nodes of G. If (u, v) is an edge of G, v is called a child of u and u a parent of v. If a directed path from u to v exists, u is a predecessor of v and v a successor of u. This is denoted by $u \prec v$. A node u is a successor (child) of a set of nodes V if some task of V is a predecessor (parent) of u. $Succ(u)$ denotes the set of successors of u. A source is a node without predecessors, a node without successors is called a sink.

A schedule for a graph G on m processors is a list of subsets of G which are called time slots. A schedule $S = (S_0, \ldots, S_{l-1})$ is valid for G if the following properties are satisfied.

1. $\bigcup_{t=0}^{l-1} S_t = G$.
2. $S_t \cap S_{t'} = \emptyset$ for all $t \neq t'$.
3. $|S_t| \leq m$ for all t.
4. If $u \prec v$, $u \in S_t$ and $v \in S_{t'}$, then $t < t'$.
5. If $u \in S_t$, then S_{t+1} contains at most one child of u.
6. If $u \in S_{t+1}$, then S_t contains at most one parent of u.

Note that these properties do not contain information about the assignment of processors. It is easy to assign a processor to every task: if S is a valid schedule for G on m processors, a correct assignment can be found in $O(\min\{mn, n+e\})$ time.

Let S be a schedule on m processors for a graph G. If a node u is an element of time slot S_t, it is said to be scheduled at time t. u does not violate its release date if $R(u) \leq t$. u is said to meet its deadline if its execution is finished at time $D(u)$. So u meets its deadline if it is scheduled at time t, such that $D(u) \geq t+1$. If $D(u) \leq t$, u is called late and its lateness is $t+1-D(u)$. The lateness of a task meeting its deadline is 0. The lateness of a schedule S is the maximum lateness of a task scheduled in S. S is optimal if no other schedule for G on m processors has lateness less than S. S is called 0-optimal if no task violates its release date or its deadline.

A partial schedule for a graph G on m processors is a schedule $S = (S_0, \ldots, S_{l-1})$ that satisfies Properties 2, 3, 4, 5 and 6 and the following.

1′. If $v \in \bigcup_{t=0}^{l-1} S_t$ and $u \prec v$, then $u \in \bigcup_{t=0}^{l-1} S_t$.

Let $S = (S_0, \ldots, S_{l-1})$ be a partial schedule of a graph G on m processors. An unscheduled node u of G is called available at time t with respect to S if $t \geq R(u)$ and $(S_0, \ldots, S_{t-1}, S_t \cup \{u\}, S_{t+1}, \ldots, S_{l-1})$ is a partial schedule of G on m processors.

3 Interval orders

In this report we will consider a special class of graphs: the interval orders. Papadimitriou and Yannakakis [10] defined these in the following way. An interval order is a partial order $(V, \prec)$, such that every element v of V can be assigned a closed interval I_v in the real line such that for all v_1, v_2 in V

$$v_1 \prec v_2 \text{ if } x < y \text{ for all } x \in I_{v_1}, y \in I_{v_2}.$$

The incomparability graph of a partial order $(V, \prec)$ is an undirected graph (V, E), such that $(u, v) \in E$ if $u \not\prec v$ and $v \not\prec u$ for all $u, v \in V$. A chordal graph is an undirected graph in which every cycle $(u_1, \ldots, u_k)$ of length $k \geq 4$ has a chord, that is an edge (u_i, u_j) such that $|j-i| \neq 1, k-1$. The two following results were proved by Papadimitriou and Yannakakis [10].

Lemma 3.1. *Let $(V, \prec)$ be a partial order. If the incomparability graph of $(V, \prec)$ is a chordal graph, then for all $u, v \in V$*

$$Succ(u) \subseteq Succ(v) \text{ or } Succ(v) \subseteq Succ(u).$$

Lemma 3.2. *Let $(V, \prec)$ be a partial order. $(V, \prec)$ is an interval order if and only if its incomparability graph is a chordal graph.*

From the preceding two lemmas it follows that if $(V, \prec)$ is an interval order, then $Succ(u) \subseteq Succ(v)$ or $Succ(v) \subseteq Succ(u)$ for all $u, v \in V$. This result can be generalised.

Proposition 3.3. *Let $(V, \prec)$ be an interval order. Let U be a non-empty subset of V. There is a task u in U such that*

$$Succ(u) = \bigcup_{v \in U} Succ(v).$$

Proof. By easy induction on the number of elements of U. □

Figure 1 contains a graph that does not satisfy the condition stated in Proposition 3.3. Hence it cannot be an induced subgraph of the transitive closure of an interval order. As a result, the class of interval orders is not a subclass, nor a superclass of the outforests or the inforests. It is, however, a superclass of the level orders which were considered by Dolev and Warmuth [5].

Figure 1: A forbidden subgraph for interval orders

4 Scheduling with deadlines

In this section an algorithm for scheduling with deadlines on an arbitrary number of processors is defined. This algorithm is similar to the one presented by Garey and Johnson [6] for scheduling with deadlines on two processors without communication delays and the one defined by Verriet [12] for scheduling subject to unit communication delays. It consists of two steps. First the deadlines are modified, such that they are consistent with the precedence constraints. The modified deadlines are used to assign a starting time to every task.

The algorithm defined by Verriet [12] does not use all knowledge of the deadlines and the structure of 0-optimal schedules: its deadline modification part only considers a task and its successors. Therefore a task with modified deadline d has at most one successor with modified deadline $d+1$. However, since the predecessors of a task are not considered, a task with modified deadline $d+1$ can have several parents with modified deadline d. Because of the asymmetric deadline modification, it is possible to construct an interval order for which this algorithm does not construct a 0-optimal schedule even if such a schedule exists.

To define an algorithm that finds 0-optimal schedules, extra information is needed. Let u be a task of a graph that has to be scheduled on m processors. Suppose $v_1, \ldots, v_k$ are successors of u with deadlines $D(v_1) \leq \ldots \leq D(v_k)$. Because of communication delays, only one successor of u can be executed immediately after u. So, in order to meet every deadline, u has to be completed at time $D(v_k) - 1 - \lceil \frac{k-1}{m} \rceil$.

Suppose two tasks u_1, u_2 have $l = km+1$ common successors $v_1, \ldots, v_l$ with deadlines $D(v_1) \leq \ldots \leq D(v_l)$. In order to meet its deadline, u_1 must be completed at time $\leq D(v_l) - 1 - \lceil \frac{l-1}{m} \rceil = D(v_l) - 1 - k$. The same holds for u_2. If both tasks are scheduled at time $D(v_l) - 2 - k$, one of their common successors violates its deadline, because the first task of $v_1, \ldots, v_l$ cannot be executed until time $D(v_l) - k$. So either u_1 or u_2 has to be scheduled at time $\leq D(v_l) - 3 - k$.

In order to use this knowledge, we will consider pairs of tasks instead of individual tasks. A pair of (not necessarily different) tasks will be assigned a deadline. Let (u_1, u_2) be a pair with deadline d. (u_1, u_2) meets its deadline if u_1 or u_2 is completed at time d. The computation of a modified deadline for every task and every pair of tasks is done by the algorithm shown in Figure 2, which uses the following definitions.

Let G be a graph. Let $D = \max_u D(u)$ and $d \leq D$. $N(u_1, u_2, d)$ denotes the number of common successors of u_1 and u_2 with deadlines $\leq d$. $P(u_1, u_2, d) = \max\{0, |V| - 1\}$, where V is a set of tasks, such that V satisfies the following properties and no other set of tasks that satisfies these properties contains more tasks than V.

1. $V \subseteq Succ(u_1) \cap Succ(u_2)$.
2. $D(v) = d + 1$ for all $v \in V$.
3. $D(v_1, v_2) = d$ for all $v_1 \neq v_2$ in V.

It is clear that, in order to meet every deadline, at most one element of V can be executed at time $\geq d$. For individual tasks we use shorthand notations: $N(u, d) = N(u, u, d)$ and $P(u, d) = P(u, u, d)$. $D_{min}(u_1)$ denotes the smallest deadline $D(u_1, u_2)$ for all tasks u_2 such that $D(u_2) = D(u_1)$.

The following lemma shows the consistency of the modified deadlines: it is proved that if every task meets its original deadline, then no modified deadline is violated.

```
DEADLINE MODIFICATION()
 1  let (u_1, ..., u_n) be a topologically sorted list of tasks of G
 2  for i = n downto 1
 3  do for d = 1 to D
 4      do if N(u_i, d) + P(u_i, d) ≥ 1
 5            then D(u_i) = min {D(u_i), d − 1 − ⌈(1/m)(N(u_i, d) + P(u_i, d) − 1)⌉}
 6           D(u_i, u_i) = D(u_i)
 7           D_min(u_i) = D(u_i)
 8      for j = n downto i + 1
 9      do for d = 1 to D
10          do if N(u_i, u_j, d) + P(u_i, u_j, d) = km + 1 and
11                D(u_i) = D(u_j) = d − 1 − k for some k
12             then D(u_i, u_j) = D(u_i) − 1
13                  D(u_j, u_i) = D(u_i) − 1
14                  D_min(u_i) = D(u_i) − 1
15                  D_min(u_j) = D(u_j) − 1
16             else D(u_i, u_j) = min {D(u_i), D(u_j)}
17                  D(u_j, u_i) = min {D(u_i), D(u_j)}
```

Figure 2: The deadline modification algorithm

Lemma 4.1. *Let G be a graph. Let S be a valid schedule for G. If in S no task violates its original deadline, then every task and every pair of tasks meets its modified deadline.*

Proof. Let G be a graph. Let S be a valid schedule for G. Suppose in S every task is completed before its original deadline.

1. Let u be a task such that every successor and every pair of successors of u meets its modified deadline. Let $d \leq \max_u D(u)$. Assume $N(u,d) + P(u,d) \geq 1$. At least $N(u,d) + P(u,d)$ successors of u are completed at time d. At most one of these tasks is executed immediately after u, so u is scheduled at time $\leq d - 2 - \lceil \frac{1}{m}(N(u,d) + P(u,d) - 1) \rceil$. So u is completed before its modified deadline.

2. Let (u_1, u_2) be a pair of tasks, such that u_1, u_2, and all successors and pairs of successors of u_1 or u_2 meet their modified deadlines. If $D(u_1, u_2) = \min\{D(u_1), D(u_2)\}$, then (u_1, u_2) meets its modified deadline. So we may assume $D(u_1, u_2) \neq \min\{D(u_1), D(u_2)\}$. In that case, $D(u_1) = D(u_2) = d - 1 - k$ for some d and k, such that $N(u_1, u_2, d) + P(u_1, u_2, d) = km + 1$. So $km + 1$ common successors of u_1 and u_2 are completed at time d. A common successor of u_1 and u_2 is scheduled at time $\leq d - 1 - k$. Therefore either u_1 or u_2 is completed at time $d - 2 - k$. Consequently, (u_1, u_2) meets its modified deadline.

□

We will prove some properties of graphs that have to be scheduled on m processors in which every task and every pair of tasks has a modified deadline. For these graphs the following statements are true.

$$\text{If } N(u,d) + P(u,d) \geq 1, \text{ then } D(u) \leq d - 1 - \lceil \tfrac{1}{m}(N(u,d) + P(u,d) - 1) \rceil. \tag{1}$$

$$\begin{aligned} &\text{If } N(u_1, u_2, d) + P(u_1, u_2, d) = km + 1 \text{ and } D(u_1) = D(u_2) = d - 1 - k \text{ for some } k, \\ &\text{then } D(u_1, u_2) = d - 2 - k. \text{ Otherwise, } D(u_1, u_2) = \min\{D(u_1), D(u_2)\}. \end{aligned} \tag{2}$$

Lemma 4.2. *Let G be a graph in which every pair of tasks has a modified deadline. Let u_1, u_2 be two tasks of G.*

$$\text{If } N(u_1, u_2, d) + P(u_1, u_2, d) \geq km + 1, \text{ then } D(u_1, u_2) \leq d - 2 - k. \tag{3}$$

Proof. Let G be a graph such that every pair of tasks of G has a modified deadline. Let $u_1 \neq u_2$ be tasks of G. Suppose $N(u_1, u_2, d)+P(u_1, u_2, d) \geq km+1$. Clearly, $N(u_1, d), N(u_2, d) \geq N(u_1, u_2, d)$ and $P(u_1, d), P(u_2, d) \geq P(u_1, u_2, d)$. From (1),

$$D(u_1), D(u_2) \leq d - 1 - \left\lceil \frac{1}{m}(N(u_1, u_2, d) + P(u_1, u_2, d) - 1) \right\rceil. \tag{4}$$

Case 1. $N(u_1, u_2, d) + P(u_1, u_2, d) \geq km + 2$. Substitution in (4) yields $D(u_1), D(u_2) \leq d - 2 - k$. Using (2), we obtain $D(u_1, u_2) \leq d - 2 - k$.

Case 2. $N(u_1, u_2, d) + P(u_1, u_2, d) = km + 1$. Using (4) yields $D(u_1), D(u_2) \leq d - 1 - k$. With (2), if $D(u_1)$ or $D(u_2)$ is smaller than $d-1-k$, then $D(u_1, u_2) \leq d-2-k$. Otherwise, from (2), $D(u_1, u_2) = d - 2 - k$.

□

Lemma 4.3. *Let G be a graph in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let V be a set of common successors of u_1 and u_2. If $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V, then $N(u_1, u_2, d) + P(u_1, u_2, d) \geq |V| - 1$.*

Proof. Let G be a graph such that a modified deadline has been computed for every pair of tasks of G. Let u_1, u_2 be tasks of G. Let V be a set of common successors of u_1 and u_2. Suppose $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V. Every task in V has a deadline at most $d + 1$. Define $V_0 = \{v \in V \mid D(v) \leq d\}$ and $V_1 = \{v \in V \mid D(v) = d + 1\}$. Obviously, $N(u_1, u_2, d) \geq |V_0|$ and $P(u_1, u_2, d) \geq |V_1| - 1$. Therefore $N(u_1, u_2, d) + P(u_1, u_2, d) \geq |V_0| + (|V_1| - 1) = |V| - 1$. □

Lemma 4.4. *Let G be a graph in which every pair of tasks has a modified deadline. Let (u_1, u_2) be a pair of tasks such that $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$. Let $v \neq u_1, u_2$. If $D(v) = D(u_1)$ and $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$, then $D(u_1, v) = D(u_2, v) = D(u_1, u_2)$.*

Proof. Let G be a graph. Suppose every pair of tasks of G has a modified deadline. Let (u_1, u_2) be a pair of tasks such that $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$. Let $v \neq u_1, u_2$ be a task such that $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$ and $D(v) = D(u_1)$. $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$, so $N(u_1, u_2, d) + P(u_1, u_2, d) = km + 1$ and $D(u_1, u_2) = d - 2 - k$ for some k and d. Since $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$, $N(u_1, v, d) + P(u_1, v, d), N(u_2, v, d) + P(u_2, v, d) \geq km + 1$. From (3), $D(u_1, v), D(u_2, v) \leq d - 2 - k$. Because $D(u_1, v), D(u_2, v) \geq D(v) - 1$, $D(u_1, v) = D(u_2, v) = D(v) - 1$. Therefore $D(u_1, v) = D(u_2, v) = D(u_1, u_2)$. □

$P(u_1, u_2, d)$ is defined in terms of sets of common successors of u_1 and u_2. Therefore its definition does not allow an efficient method of determining $P(u_1, u_2, d)$. For interval orders, an alternative definition can be used. This formulation allows us to compute $P(u_1, u_2, d)$ in linear time.

Lemma 4.5. *Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. If $P(u_1, u_2, d) \geq 1$, then*

$$P(u_1, u_2, d) = |\{v_1 \mid u_1, u_2 \prec v_1 \ \&\ D(v_1) = d + 1 \ \&\ \exists v_2 \, [D(v_2) = d + 1 \ \&\ D(v_1, v_2) = d]\}| - 1.$$

Proof. Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Suppose $P(u_1, u_2, d) \geq 1$. In that case, $P(u_1, u_2, d) = |V_P| - 1$, where V_P is a largest subset of $Succ(u_1) \cap Succ(u_2)$ such that every task in V_P has deadline $d + 1$ and every pair of different tasks has deadline d. Define $V = \{v_1 \in Succ(u_1) \cap Succ(u_2) \mid D(v_1) = d + 1 \ \&\ \exists v_2[D(v_2) = d + 1 \ \&\ D(v_1, v_2) = d]\}$. $|V_P| \geq 2$, so for every task v_1 in V_P $D(v_1, v_2) = d$

for some v_2 with deadline $d+1$. So $V_P \subseteq V$. Since G is an interval order, we may assume $V_P = \{v_1, \ldots, v_k\}$, such that $Succ(v_1) \subseteq \ldots \subseteq Succ(v_k)$. Suppose $V \not\subseteq V_P$. Let v be a task of $V \backslash V_P$. v is a common successor of u_1 and u_2.

Case 1. $Succ(v_1) \subseteq Succ(v)$. Clearly, $Succ(v_1) \cap Succ(v_i) \subseteq Succ(v)$ for all i, $2 \leq i \leq k$. From Lemma 4.4, $D(v_i, v) = d$ for every i, $1 \leq i \leq k$. So $V_P \cup \{v\}$ is a set of common successors of u_1 and u_2 with deadline $d+1$, such that every pair of different tasks has deadline d. Contradiction.

Case 2. $Succ(v) \subseteq Succ(v_1)$. v is a task of V, so there is a task v' such that $D(v') = d+1$ and $D(v, v') = d$. Assume $v' \neq v_1$. Because $Succ(v) \cap Succ(v') \subseteq Succ(v_1)$, $D(v_1, v) = d$ with Lemma 4.4. Furthermore, $Succ(v_1) \cap Succ(v) \subseteq Succ(v_i)$ for all i, $2 \leq i \leq k$. Applying Lemma 4.4 yields $D(v_i, v) = d$ for all i, $1 \leq i \leq k$. As a result, V_P is not a largest subset $Succ(u_1) \cap Succ(u_2)$ such that every pair of different tasks has deadline d and every task deadline $d+1$. Contradiction.

So $V = V_P$ and $P(u_1, u_2, d) = |V| - 1$. □

Corollary 4.6. *Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. $P(u_1, u_2, d) =$*

$$\max\{0, |\{v_1 \mid u_1, u_2 \prec v_1 \ \&\ D(v_1) = d+1 \ \&\ \exists v_2\, [D(v_2) = d+1 \ \&\ D(v_1, v_2) = d]\}| - 1\}.$$

Proof. Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Define $V = \{v_1 \in Succ(u_1) \cap Succ(u_2) \mid D(v_1) = d+1 \ \&\ \exists v_2[D(v_2) = d+1 \ \&\ D(v_1, v_2) = d]\}$. Using Lemma 4.5, we may assume $P(u_1, u_2, d) = 0$. In that case, $D(v_1, v_2) = d+1$ for all common successors v_1, v_2 of u_1 and u_2 such that $D(v_1) = D(v_2) = d+1$. Hence $|V| \leq 1$. So $P(u_1, u_2, d) = \max\{0, |V| - 1\}$. □

Let G be an interval order. We will assume G is a transitive closure. For every pair of tasks (u_1, u_2) linear time is required to compute $N(u_1, u_2, d)$ for all d, because we may assume $d \leq n$.

When the deadline modification algorithm considers a pair of tasks (u_1, u_2), the modified deadlines of all successors and pairs of successors of u_1 and u_2 have already been calculated. So, from that time on, $P(u_1, u_2, d)$ does not change anymore. Hence, from Corollary 4.6, we conclude $P(u_1, u_2, d) = \max\{0, |V| - 1\}$, where $V = \{v \in Succ(u_1) \cap Succ(u_2) \mid D(v) = d+1 \ \&\ D_{min}(v) = d\}$. So $P(u_1, u_2, d)$ can be computed in linear time for all d. Note that this does not hold for arbitrary graphs.

Computing the modified deadlines of the individual tasks clearly takes $O(n^2)$ time. The calculation of a deadline of a pair of tasks takes linear time for each pair, so $O(n^3)$ time for all pairs. Coppersmith and Winograd [4] showed that computing the transitive closure of a directed acyclic graph takes $O(n^{2.376})$ time. Therefore the modified deadlines are computed in $O(n^3)$ time.

To define the list scheduling algorithm, a notion of priority is required. Let u_1 and u_2 be two tasks of an interval order G. u_1 has a higher priority than u_2, if $D(u_1) < D(u_2)$ or $D(u_1) = D(u_2)$ and $Succ(u_1) \supsetneqq Succ(u_2)$. A list of tasks ordered by non-increasing priority will be called a priority list.

The list scheduling part does not consider pairs of tasks. It is shown in Figure 3.

The priority list used by the list scheduling algorithm has length n. It is traversed once for each time slot. The length of the schedule for G is at most n, so this takes $O(n^2)$ time. Therefore the list scheduling algorithm can be implemented such that the availability of a task can be checked in constant time. Hence it constructs a schedule in $O(n^2)$ time.

The list scheduling algorithm constructs a 0-optimal schedule for an interval order with deadlines if a 0-optimal schedule exists.

```
LIST SCHEDULING ALGORITHM()
1  assume L = (u_1, ..., u_n) is a priority list
2  t = 0
3  while L contains unscheduled tasks
4  do for i = 1 to n
5     do if u_i is unscheduled and available at time t
6           then schedule u_i at time t
7     t = t + 1
```

Figure 3: The list scheduling algorithm

Theorem 4.7. *Let G be an interval order in which every pair of tasks has a modified deadline. Let L be a priority list. If a 0-optimal schedule for G on m processors exists, then in the schedule for G on m processors constructed by the list scheduling algorithm using L every pair of tasks meets its deadline.*

Proof. Let G be an interval order such that every pair of tasks of G has a modified deadline. Let L be a priority list. Suppose there is a 0-optimal schedule for G on m processors. Let S be the schedule for G on m processors constructed by the list scheduling algorithm using L. Suppose in S not every pair of tasks meets its deadline. Assume S_t is the first time slot containing a task in a pair violating its deadline. Let u_1 be this task and let (u_1, u_2) be this pair. (u_1, u_2) violates its deadline. So $D(u_1, u_2) \leq t$. 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 a 0-optimal schedule for G exists, there are at most mt tasks with deadline $\leq t$. Hence there is a time slot before S_t that is idle or contains a task with deadline $\geq t + 1$. Let $S_{t'-1}$ be the last such time slot. Define $G_1 = \bigcup_{i=t'}^{t-1} S_i \cup \{u_1\}$. G_1 contains $m(t - t') + 1$ tasks with deadline at most t. No task of G_1 was available at time $t'-1$. Let u be a source of G_1. u was not available at time $t'-1$, because one of the following statements is true.

1. $S_{t'-1}$ contains a parent of u.
2. $S_{t'-2}$ contains two parents of u.
3. $S_{t'-2}$ contains a parent u' of u and $S_{t'-1}$ contains another child of u'.

Hence every task in G_1 has a predecessor in $S_{t'-2} \cup S_{t'-1}$.

Case 1.1. Every task in G_1 is a successor of $S_{t'-1}$. Define $Q = S_{t'-1} \cap \{v \in G \mid D(v) \leq t\}$. Because every task in G_1 has deadline $\leq t$, every task in G_1 is a successor of Q. Because of communication delays, at most $|Q|$ successors of Q are scheduled at time t'. Since Q contains less than m tasks, $t = t'$. As a result, Q contains a parent w of u_1. Hence $N(w, t) \geq 1$. From (1), $D(w) \leq t-1 = t'-1$. So w is not completed before its deadline. Contradiction.

Case 1.2. Not every task of G_1 has a predecessor in $S_{t'-1}$. Define $V = \{w \in S_{t'-2} \cup S_{t'-1} \mid w \text{ is a parent of } G_1\}$. From Proposition 3.3, V contains a task w_1 such that $G_1 \subseteq Succ(w_1)$. Define $V' = V \backslash \{w_1\}$.

Case 1.2.1. Every task in G_1 has a predecessor in V'. With Proposition 3.3, V' contains a task w_2 such that $G_1 \subseteq Succ(w_2)$. Hence w_1 and w_2 have $\geq m(t-t')+1$ common successors with deadlines $\leq t$. So $N(w_1, w_2, t)+P(w_1, w_2, t) \geq m(t-t')+1$. From (3), $D(w_1, w_2) \leq t - 2 - (t - t') = t' - 2$. So (w_1, w_2) violates its deadline. Contradiction.

Case 1.2.2. Not every task in G_1 has a predecessor in V'. Let v be a task in G_1 without a predecessor in V'. Assume v is a source of G_1. V contains a parent of v. V', however, does not. So w_1 is a parent of v. Not every task in G_1 is a successor of $S_{t'-1}$, so $w_1 \in S_{t'-2}$. Because $S_{t'-2}$ does not contain another parent of v, $S_{t'-1}$ contains another child v' of w_1. v' is scheduled before v, so v' occurs before v in L. Thus $D(v') \leq D(v)$. As a result, $N(w_1, t) \geq m(t-t')+2$. From (1),

$$\begin{aligned} D(w_1) &\leq t - 1 - \left\lceil \tfrac{1}{m}(m(t-t')+1) \right\rceil \\ &= t - 1 - t + t' - 1 \\ &\leq t' - 2. \end{aligned}$$

So w_1 is late. Contradiction.

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

Case 3. $D(u_1) = D(u_2) = t+1$ and $D(u_1, u_2) = t$. Assume $Succ(u_1) \subseteq Succ(u_2)$. Let U be the set of tasks with priority as least as high as u_1. $|U| \geq 2$, since $u_1, u_2 \in U$. Let v_1, v_2 be two tasks of U. Clearly, $D(v_1), D(v_2) \leq D(u_1) = t+1$. If $D(v_1) \leq t$ or $D(v_2) \leq t$, then, from (2), $D(v_1, v_2) \leq t$. We will assume $D(v_1) = D(v_2) = t+1$. Since the priority of v_1 and v_2 is at least as high as that of u_1, $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v_1), Succ(v_2)$. By applying Lemma 4.4 twice, we obtain $D(v_1, v_2) = t$. So, in order to meet every deadline, at most one task of U can be scheduled at time $\geq t$. Since a 0-optimal schedule for G exists, U contains at most $mt+1$ tasks. Therefore there is a time slot before S_t that contains at most $m-1$ tasks with priority as least as high as u_1. Let $t'-1$ be the last such time before t. Define $G_2 = \bigcup_{i=t'}^{t-1} S_i \cup \{u_1, u_2\} \cup \{v \in \bigcup_{i \geq t} S_i \mid v \prec u_2\}$. G_2 contains at least $m(t-t')+2$ tasks and every pair of different tasks of G_2 has deadline $\leq t$. No task of G_2 was available at time $t'-1$. Let u be a source of G_2. u was not available at time $t'-1$, because one of the following three conditions is satisfied.

1. $S_{t'-1}$ contains a parent of u.
2. $S_{t'-2}$ contains two parents of u.
3. $S_{t'-2}$ contains a parent u' of u and $S_{t'-1}$ contains another child of u'.

Thus every task in G_2 has a predecessor in $S_{t'-2}$ or $S_{t'-1}$.

Case 3.1. Every task in G_2 is a successor of $S_{t'-1}$. Define $Q = S_{t'-1} \cap \{v \in G \mid D(v) \leq t\}$. Since every task in G_2 has deadline $\leq t+1$, every task in G_2 is a successor of Q. S is a valid schedule, so at most $|Q| \leq m-1$ successors of Q are executed at time t'. Therefore $t = t'$. From Proposition 3.3, Q contains a predecessor w of both u_1 and u_2. Hence $N(w, t+1) \geq 2$. With (1), $D(w) \leq (t+1) - 2 = t - 1 = t' - 1$. So w violates its deadline. Contradiction.

Case 3.2. Not every source of G_2 has a parent in $S_{t'-1}$. Define $V = \{w \in S_{t'-2} \cup S_{t'-1} \mid w \text{ is a parent of } G_2\}$. With Proposition 3.3, V contains a task w_1 such that every task of G_2 is a successor of w_1. Let $V' = V \backslash \{w_1\}$.

Case 3.2.1. Every task in G_2 is a successor of V'. From Proposition 3.3, V' contains a task w_2 such that $G_2 \subseteq Succ(w_2)$. Hence every task in G_2 is a common successor of w_1 and w_2. It is easy to see that $D(v_1, v_2) \leq t$ for all $v_1 \neq v_2$ in G_2. Applying Lemma 4.3, we get $N(w_1, w_2, t) + P(w_1, w_2, t) \geq m(t-t')+1$. From (3), $D(w_1, w_2) \leq t - 2 - (t - t') = t' - 2$. So (w_1, w_2) violates its deadline. Contradiction.

Case 3.2.2. Not every task in G_2 is a successor of V'. Let v be a task of G_2 that has no predecessor in V'. Assume v is a source of G_2. V contains a parent of v. V', however, does not. So w_1 is a parent of v. Because v is a not successor of $S_{t'-1}$, w_1 is scheduled at time $t'-2$. Since v was not available at time $t'-1$ and $S_{t'-2}$ contains only one parent of v, $S_{t'-1}$ contains another child v' of w_1. v' is scheduled before v, so v' occurs before v in L. Therefore $D(v') < D(v)$ or

$D(v') = D(v)$ and $Succ(v') \supseteq Succ(v)$. Every pair consisting of two different tasks of $G_2 \cup \{v'\}$ has deadline $\leq t$. Applying Lemma 4.3 to w_1 and $G_2 \cup \{v'\}$ yields $N(w_1, t) + P(w_1, t) \geq m(t - t') + 2$. From (1),

$$\begin{aligned} D(w_1) &\leq t - 1 - \left\lceil \tfrac{1}{m}(m(t-t')+1) \right\rceil \\ &= t - 1 - t + t' - 1 \\ &\leq t' - 2. \end{aligned}$$

So w_1 is not completed before its deadline. Contradiction.

□

Corollary 4.8. *Let G be an interval order for which a 0-optimal schedule exists on m processors. The above-mentioned algorithm finds a 0-optimal schedule for G on m processors.*

Proof. Obvious. □

Let G be an interval order in which every task u has an original deadline $D_0(u)$. Let $D_1(u)$ and $D_1(u_1, u_2)$ denote the deadlines of u and (u_1, u_2) computed by the deadline modification algorithm. Let S be an optimal schedule for G on m processors. Suppose its lateness is l. Define $D_0'(u) = D_0(u) + l$ for every task u in G. Let G' denote the interval order G in which every task has deadline $D_0'(u)$. In S every task u meets its modified deadline $D_0'(u)$. So the above-mentioned algorithm finds a schedule for G' in which every task u is completed at time $D_0'(u)$. First it computes modified deadlines $D_1'(u)$ and $D_1'(u_1, u_2)$. It is easy to see that $D_1'(u) = D_1(u) + l$ for every task u in G and $D_1'(u_1, u_2) = D_1(u_1, u_2) + l$ for every pair of tasks (u_1, u_2). So every priority list used to construct a 0-optimal schedule is a priority list with respect to deadlines $D_1(u)$ as well. The availability of a task is independent of its deadline, so the schedule constructed by the list scheduling algorithm only depends on the precedence constraints and the priority list. Hence in the schedule for G constructed by the algorithm above on m processors every task u is completed at time $D_0'(u) = D_0(u) + l$. So this schedule is optimal.

Corollary 4.9. *Let G be an interval order in which every task has been assigned a deadline. The schedule for G on m processors constructed by the above-mentioned algorithm is optimal.*

Proof. Obvious. □

5 Scheduling with release dates and deadlines

Scheduling with release dates and deadlines can be done in a similar manner to scheduling with only deadlines. The algorithm presented in this section also consists of two parts. The first modifies every release date and every deadline, the second does the actual scheduling.

The modification of the release dates is very simple. In every valid schedule for a graph a task is scheduled after all its predecessors. Hence the release date of a task may exceed those of its predecessors. Therefore the release dates can be modified as follows.

As long as there are nodes without a modified release date, select such a node u whose parents have been assigned a modified release date. Assume $v_1, \ldots, v_k$ are the parents of u. Then

$$R(u) = \max \left\{ R(u), \max_{1 \leq i \leq k} R(v_i) + 1 \right\}.$$

It is obvious that in every valid schedule not violating any original release date no task is scheduled before its modified release date.

The modification of the deadlines is more involved. Three definitions are needed. Let G be a graph in which every task has a deadline and a (modified) release date. For all nodes u in G and integers r, d such that $R(u) \leq r \leq D(u) \leq d$

$$G(u, r, d) = \{v \in G \mid D(v) \leq d \;\&\; (v \in Succ(u) \vee R(v) \geq r) \;\&\; u \neq v\}.$$

For all nodes u in G and integers r, d such that $R(u) \leq r \leq D(u) \leq d$ and $d > r + 1$

$$H(u, r, d) = \{v \in G \mid D(v) \leq d \ \&\ (v \in Succ(u) \vee R(v) > r + 1)\}.$$

For all nodes u_1, u_2 in G and integers r, d such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$

$$G(u_1, u_2, r, d) = G(u_1, r, d) \cap G(u_2, r, d).$$

Note that $G(u, r, d) = G(u, u, r, d)$. The sets $G(u, r, d)$, $H(u, r, d)$ and $G(u_1, u_2, r, d)$ will be used to define the deadline modification. A set $G(u, r, d)$ contains tasks that have to be completed before time d and cannot start execution until time r or after the execution u is completed. Therefore if $G(u, r, d)$ is sufficiently large, u has to be executed before all tasks of $G(u, r, d)$ in order to meet its deadline. Something similar holds for $H(u, r, d)$ and $G(u_1, u_2, r, d)$.

To be able to consider pairs of tasks in the deadline modification algorithm, $P(u_1, u_2, r, d)$ is defined. Let r, d be integers such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$. $P(u_1, u_2, r, d) = \max\{0, |V| - 1\}$, where V is a set of tasks satisfying the following properties and V contains at least as many tasks as any set satisfying these properties.

1. $V \subseteq G(u_1, u_2, r, d + 1) \backslash G(u_1, u_2, r, d)$.
2. $D(v_1, v_2) = d$ for all $v_1 \neq v_2$ in V.

In order to to meet every deadline, at most one task of V can be executed at time $\geq d$. Like in the previous section we use shorthand notations: $P(u, r, d) = P(u, u, r, d)$.

The deadline modification algorithm has a loop structure similar to the deadline modification part of the algorithm by Verriet [12] for scheduling with non-uniform release dates. Let $D = \max_u D(u)$ and $R = \max_u R(u)$. The following notation is introduced: $D_{min}(u_1)$ denotes the smallest deadline $D(u_1, u_2)$ for a task u_2 such that $D(u_2) = D(u_1)$. The deadline modification algorithm is shown in Figure 4.

The following lemma shows the consistency of the modified deadlines.

Lemma 5.1. *Let G be a graph. Let S be a valid schedule for G. If no task of G violates its original deadline, then every task and every pair of tasks meets its modified deadline.*

Proof. Let G be a graph. Let S be a valid schedule for G. Suppose in S every task meets its original deadline. During the execution of the deadline modification algorithm, a number of deadlines get changed. Let l be the number of times a deadline is decreased. Let $D_i(u)$ and $D_i(u_1, u_2)$ denote the deadline of u and (u_1, u_2) after the i^{th} modification. Let $D_0(u)$ denote the original deadline of u and $D_0(u_1, u_2) = \min\{D_0(u_1), D_0(u_2)\}$. Let $G_i(u, r, d)$, $H_i(u, r, d)$, $G_i(u_1, u_2, r, d)$ and $P_i(u_1, u_2, r, d)$ denote $G(u, r, d)$, $H(u, r, d)$, $G_i(u_1, u_2, r, d)$ and $P(u_1, u_2, r, d)$ after the i^{th} modification. Clearly, every deadline $D_0(u)$ and $D_0(u_1, u_2)$ is met. Suppose no deadline $D_i(u)$ or $D_i(u_1, u_2)$ is violated. Exactly one deadline is decreased by the $i + 1^{\text{th}}$ modification. This deadline is changed, because Line 6, 10, 16, 17, 20 or 21 of the deadline modification algorithm is executed.

Case 1. Line 6 is executed. For some task u $D_{i+1}(u) = d - \lceil \frac{1}{m}(|G_i(u, r, d)| + P_i(u, r, d)) \rceil$ such that $|G_i(u, r, d)| + P_i(u, r, d) \geq m(d - r)$. If $P_i(u, r, d) = 0$, let $G' = G_i(u, r, d)$. Otherwise, $P_i(u, r, d) = |V| - 1$, where V is a largest subset of $G_i(u, r, d + 1) \backslash G_i(u, r, d)$ such that $D_i(v_1, v_2) = d$ for all $v_1 \neq v_2$ in V. Let $G' = G_i(u, r, d) \cup V'$, where V' is the subset of V containing the tasks that are completed at time d. Because every task v of V is completed at time $D_i(v)$ and either v_1 or v_2 is completed at time $D_i(v_1, v_2)$ for all v_1, v_2 in V, $|V'| = |V| - 1$. Hence $|G'| = |G_i(u, r, d)| + P_i(u, r, d)$.

Case 1.1. $|G_i(u, r, d)| + P_i(u, r, d) > m(d - r)$. G' contains $\geq m(d - r) + 1$ tasks that are completed at time d. One of them is scheduled at time $\leq d - \lceil \frac{1}{m}(|G_i(u, r, d)| + P_i(u, r, d)) \rceil \leq d - (d - r + 1) = r - 1$. This is a successor of u. So u is completed at time $\leq d - \lceil \frac{1}{m}(|G_i(u, r, d)| + P_i(u, r, d)) \rceil$. Hence u is finished at time $D_{i+1}(u)$.

```
DEADLINE MODIFICATION()
 1  assume (u_1, ..., u_n) contains all tasks ordered by non-decreasing modified release dates
 2  for d = D downto 1
 3  do for i = 1 to n such that D(u_i) ≤ d
 4     do for r = R(u_i) to R
 5        do if |G(u_i, r, d)| + P(u_i, r, d) ≥ m(d − r)
 6              then D(u_i) = min{ D(u_i), d − ⌈(1/m)(|G(u_i, r, d)| + P(u_i, r, d))⌉ }
 7                   D(u_i, u_i) = D(u_i)
 8                   D_min(u_i) = min{D_min(u_i), D(u_i)}
 9           if d > r + 1 and |H(u_i, r, d)| + P(u_i, r + 2, d) ≥ m(d − (r + 2)) + 2
10              then D(u_i) = min{ D(u_i), d − 1 − ⌈(1/m)(|H(u_i, r, d)| + P(u_i, r + 2, d) − 1)⌉ }
11                   D_min(u_i) = min{D_min(u_i), D(u_i)}
12           for j = 1 to n
13           do if R(u_j) ≤ r and D(u_j) ≤ d and
14                 |G(u_i, u_j, r, d)| + P(u_i, u_j, r, d) ≥ m(d − r) + 1 and
15                 D(u_i) = D(u_j) = d − ⌈(1/m)(|G(u_i, u_j, r, d)| + P(u_i, u_j, r, d))⌉
16              then D(u_i, u_j) = D(u_i) − 1
17                   D(u_j, u_i) = D(u_i) − 1
18                   D_min(u_i) = D(u_i) − 1
19                   D_min(u_j) = D(u_j) − 1
20              else D(u_i, u_j) = min{D(u_i), D(u_j)}
21                   D(u_j, u_i) = min{D(u_i), D(u_j)}
```

Figure 4: The deadline modification algorithm

Case 1.2. $|G_i(u,r,d)| + P_i(u,r,d) = m(d-r)$. Let $G'' = G' \cup \{u\}$. G'' contains $m(d-r)+1$ tasks, all of which are completed at time d. So one of these tasks is scheduled at time $\leq d - \lceil \frac{1}{m} |G''| \rceil \leq r-1$. This is either u or a successor of u. So u is executed before time $D_{i+1}(u) = r$.

Case 2. Line 10 of the deadline modification algorithm is executed. There is a task u such that $D_{i+1}(u) = d - 1 - \lceil \frac{1}{m}(|H_i(u,r,d)| + P_i(u,r+2,d) - 1) \rceil$, $d > r+1$ and $|H_i(u,r,d)| + P_i(u,r+2,d) \geq m(d-(r+2))+2$. Every task of $H_i(u,r,d)$ is scheduled at time $\leq d-1$. If $P_i(u,r+2,d) = 0$, let $H = H_i(u,r,d)$. Otherwise, $P_i(u,r+2,d) = |V| - 1$ for some $V \subseteq G_i(u,r+2,d+1) \backslash G_i(u,r+2,d)$ such that every pair of different tasks of V has deadline d. Let $H = H_i(u,r,d) \cup V'$ where V' contains every task of V that is completed at time d. $|V'| = |V| - 1$, so $|H| = |H_i(u,r,d)| + P_i(u,r+2,d)$. Hence H contains $\geq m(d-(r+2))+2$ tasks that are completed at time d. Let v_1, v_2 be two tasks of H scheduled at time t_1 and t_2 such that $t_1 \leq t_2$ and all other tasks of H are scheduled at time $\geq t_2$. Define $H' = H \backslash \{v_1\}$. It takes at least $\lceil \frac{1}{m} |H'| \rceil \geq d - r - 1$ time slots to schedule every task of H'. v_2 is a task of H' which is scheduled in the earliest time slot containing a task of H'. So $t_2 \leq d - \lceil \frac{1}{m} |H'| \rceil \leq r+1$. So v_2 is a successor of u. Since $t_1 \leq t_2$, v_1 is also a successor of u. Because of communication delays, u is scheduled at time $\leq t_2 - 2 \leq d - 2 - \lceil \frac{1}{m}(|H|-1) \rceil$. Since $|H| = |H_i(u,r,d)| + P_i(u,r+2,d)$, u is completed at time $d - 1 - \lceil \frac{1}{m}(|H_i(u,r,d)| + P_i(u,r+2,d) - 1) \rceil$. So u does not violate deadline $D_{i+1}(u)$.

Case 3. Line 16 is executed. $D_{i+1}(u_1,u_2) = D_i(u_1) - 1$, $|G_i(u_1,u_2,r,d)| + P_i(u_1,u_2,r,d) \geq m(d-r)+1$ and $D_i(u_1) = D_i(u_2) = d - \lceil \frac{1}{m}(|G_i(u_1,u_2,r,d)| + P_i(u_1,u_2,r,d)) \rceil$ for some pair of tasks (u_1,u_2). If $P_i(u_1,u_2,r,d) = 0$, let $G' = G_i(u_1,u_2,r,d)$. Otherwise, let $G' = G_i(u_1,u_2,r,d) \cup V'$, where V is a largest subset of $G_i(u_1,u_2,r,d+1) \backslash G_i(u_1,u_2,r,d)$ such that $D_i(v_1,v_2) = d$ for all $v_1 \neq v_2$ and V' contains every task of V that is scheduled at time $\leq d-1$. Since every task v in V is scheduled before time $D_i(v)$, $P_i(u_1,u_2,r,d) = |V| - 1 = |V'|$. G' contains $|G_i(u_1,u_2,r,d)| + P_i(u_1,u_2,r,d) \geq m(d-r)+1$ tasks that are completed at

time d. So the first task that is completed is scheduled at time $\leq d - \lceil \frac{1}{m}(|G_i(u_1, u_2, r, d)| + P_i(u_1, u_2, r, d))\rceil \leq r-1$. This is a common successor of u_1 and u_2. So u_1 or u_2 is scheduled at time $\leq d-1-\lceil \frac{1}{m}(|G_i(u_1, u_2, r, d)|+P_i(u_1, u_2, r, d))\rceil$. So (u_1, u_2) meets deadline $D_{i+1}(u_1, u_2)$.

Case 4. Line 17 is executed. Similar to Case 3.

Case 5. Line 20 is executed. Obvious.

Case 6. Line 21 is executed. Obvious.

So all deadlines $D_{i+1}(u)$ and $D_{i+1}(u_1, u_2)$ are met. Therefore no modified deadline $D_l(u)$ or $D_l(u_1, u_2)$ is violated. □

In the following lemmas we prove some properties of graphs in which every pair of tasks has a modified deadline that have to be scheduled on m processors. For these graphs the following statements are true.

$$\text{If } |G(u, r, d)| + P(u, r, d) \geq m(d - r), \text{ then } D(u) \leq d - \lceil \tfrac{1}{m}(|G(u, r, d)| + P(u, r, d))\rceil. \tag{5}$$

$$\begin{gathered}\text{If } |H(u, r, d)| + P(u, r + 2, d) \geq m(d - (r + 2)) + 2,\\ \text{then } D(u) \leq d - 1 - \lceil \tfrac{1}{m}(|H(u, r, d)| + P(u, r, d) - 1)\rceil.\end{gathered} \tag{6}$$

$$\begin{gathered}\text{If } |G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq m(d - r) + 1 \text{ and}\\ D(u_1) = D(u_2) = d - \lceil \tfrac{1}{m}(|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d))\rceil,\\ \text{then } D(u_1, u_2) \leq D(u_1) - 1. \text{ Otherwise, } D(u_1, u_2) = \min\{D(u_1), D(u_2)\}.\end{gathered} \tag{7}$$

Lemma 5.2. *Let G be a graph in which every pair of tasks has a modified deadline. Let u_1, u_2 be two tasks of G.*

$$\textit{If } |G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq m(d - r) + 1, \textit{ then } D(u_1, u_2) \leq r - 2. \tag{8}$$

Proof. Let G be a graph in which every pair of tasks has a modified deadline. Let u_1 and u_2 be different tasks of G. Suppose $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq m(d - r) + 1$. Obviously, $|G(u_1, r, d)| + P(u_1, r, d), |G(u_2, r, d)| + P(u_2, r, d) \geq |G(u_1, u_2, r, d)| + P(u_1, u_2, r, d)$. So $D(u_1), D(u_2) \leq d - \lceil \frac{1}{m}(|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d))\rceil \leq r - 1$.

Case 1. $D(u_1) = D(u_2) = d - \lceil \frac{1}{m}(|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d))\rceil$. From (7), $D(u_1, u_2) \leq D(u_1) - 1$. So $D(u_1, u_2) \leq r - 2$.

Case 2. $\min\{D(u_1), D(u_2)\} < d-\lceil \frac{1}{m}(|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d))\rceil$. Applying (7) yields $D(u_1, u_2) = \min\{D(u_1), D(u_2)\}$. Hence $D(u_1, u_2) \leq r - 2$.

□

Lemma 5.3. *Let G be a graph in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let V be a set containing tasks with release date $\geq r$ and common successors of u_1 and u_2. If $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V, then $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq |V| - 1$.*

Proof. Let G be a graph such that every pair of tasks of G has a modified deadline. Let u_1, u_2 be tasks of G. Let V be a set containing tasks with release date $\geq r$ and common successors of u_1 and u_2. Suppose $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V. Every task in V has a deadline $\leq d+1$. Define $V_0 = \{v \in V \mid D(v) \leq d\}$ and $V_1 = \{v \in V \mid D(v) = d + 1\}$. Obviously, $V_0 \subseteq G(u_1, u_2, r, d)$ and $P(u_1, u_2, r, d) \geq |V_1| - 1$. Hence $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq |V_0| + (|V_1| - 1) = |V| - 1$. □

Lemma 5.4. *Let G be a graph in which every pair of tasks has a modified deadline. Let u be a task of G. Let V be a set containing tasks with release date $\geq r + 2$ and successors of u. If $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V, then $|H(u, r, d)| + P(u, r + 2, d) \geq |V| - 1$.*

Proof. Let G be a graph in which every pair of tasks has a modified deadline. Let u be a task of G. Let V be a set containing tasks with release date $\geq r+2$ and successors of u. Suppose $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in V. From (7), $D(v) \leq d+1$ for all v in V. Define $V_0 = \{v \in V \mid D(v) \leq d\}$ and $V_1 = \{v \in V \mid D(v) = d+1\}$. Clearly, $V_0 \subseteq H(u, r, d)$ and $P(u, r+2, d) \geq |V_1| - 1$. Hence $|H(u, r, d)| + P(u, r+2, d) \geq |V_0| + (|V_1| - 1) = |V| - 1$. □

Lemma 5.5. *Let G be a graph in which every pair of tasks has a modified deadline. Let (u_1, u_2) be a pair of tasks such that $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$. Let $v \neq u_1, u_2$. If $D(v) = D(u_1)$ and $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$, then $D(u_1, v) = D(u_2, v) = D(u_1, u_2)$.*

Proof. Let G be a graph. Suppose that every pair of tasks of G has a modified deadline. Let (u_1, u_2) be a pair of tasks such that $D(u_1, u_2) = D(u_1) - 1 = D(u_2) - 1$. Let v be a task of G different from u_1 and u_2, such that $D(v) = D(u_1)$ and $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$. $D(u_1, u_2) = D(u_1) - 1$, so $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq m(d-r)+1$ for some r and d. Because $Succ(u_1) \cap Succ(u_2) \subseteq Succ(v)$, also $|G(u_1, v, r, d)| + P(u_1, v, r, d), |G(u_2, v, r, d)| + P(u_2, v, r, d) \geq m(d-r)+1$. From (8), $D(u_1, v), D(u_2, v) \leq r-2$. Since $D(u_1, v), D(u_2, v) \geq D(v) - 1$, $D(u_1, v) = D(u_2, v) = D(u_1, u_2)$. □

Because $P(u_1, u_2, r, d)$ is defined in terms of subsets of $G(u_1, u_2, r, d+1) \backslash G(u_1, u_2, r, d)$, its definition does not allow an efficient method of calculating $P(u_1, u_2, r, d)$. For interval orders, we will derive a new formulation which allows us to determine $P(u_1, u_2, r, d)$ in linear time.

Lemma 5.6. *Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let r, d be integers such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$. If $P(u_1, u_2, r, d) \geq 1$, then $P(u_1, u_2, r, d) =$*

$$|\{v_1 \in G(u_1, u_2, r, d+1) \mid D(v_1) = d+1 \ \& \ \exists v_2 \, [D(v_2) = d+1 \ \& \ D(v_1, v_2) = d]\}| - 1.$$

Proof. Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let r, d be integers such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$. Define $V = \{v_1 \in G(u_1, u_2, r, d+1) \mid D(v_1) = d+1 \ \& \ \exists v_2 \, [D(v_2) = d+1 \ \& \ D(v_1, v_2) = d]\}$. Suppose $P(u_1, u_2, r, d) \geq 1$. $P(u_1, u_2, r, d) = |V_P| - 1$, where V_P is a largest subset of $G(u_1, u_2, r, d+1) \backslash G(u_1, u_2, r, d)$ such that every pair of tasks of V_P have deadline d. Obviously, $V_P \subseteq V$. Because G is an interval order, we may assume $V_P = \{v_1, \ldots, v_k\}$ such that $Succ(v_1) \subseteq \ldots \subseteq Succ(v_k)$. Suppose V is not a subset of V_P. Let v be a task of $V \backslash V_P$.

Case 1. $Succ(v_1) \subseteq Succ(v)$. $Succ(v_1) \cap Succ(v_i) \subseteq Succ(v)$ for all i, $2 \leq i \leq k$, since $Succ(v_1) \subseteq Succ(v_i)$. With Lemma 5.5, $D(v_i, v) = d$ for all i, $1 \leq i \leq k$. As a result, $V_P \cup \{v\}$ is a subset of $G(u_1, u_2, r, d+1) \backslash G(u_1, u_2, r, d)$ such that every pair of tasks has deadline d. Contradiction.

Case 2. $Succ(v) \subseteq Succ(v_1)$. Since $v \in V$, $D(v, v') = d$ for some v' such that $D(v') = d+1$. Assume $v' \neq v_1$. $Succ(v) \cap Succ(v') \subseteq Succ(v_1)$, so, from Lemma 5.5, $D(v_1, v) = d$. Furthermore, $Succ(v_1) \cap Succ(v) \subseteq Succ(v_i)$ for all i, $2 \leq i \leq k$. Using Lemma 5.5, we obtain $D(v_i, v) = d$ for all i, $1 \leq i \leq k$. So $V_P \cup \{v\}$ is a subset of $G(u_1, u_2, r, d+1) \backslash G(u_1, u_2, r, d)$ such that every pair of tasks has deadline d that is larger than V_P. Contradiction.

Consequently, $V = V_P$ and $P(u_1, u_2, r, d) = |V| - 1$. □

Corollary 5.7. *Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let r, d be integers such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$. $P(u_1, u_2, r, d) =$*

$$\max\{0, |\{v_1 \in G(u_1, u_2, r, d+1) \mid D(v_1) = d+1 \ \& \ \exists v_2 \, [D(v_2) = d+1 \ \& \ D(v_1, v_2) = d]\}| - 1\}.$$

Proof. Let G be an interval order in which every pair of tasks has a modified deadline. Let u_1, u_2 be tasks of G. Let r, d be integers such that $R(u_1), R(u_2) \leq r \leq D(u_1), D(u_2) \leq d$. Define $V = \{v_1 \in G(u_1, u_2, r, d+1) \mid D(v_1) = d+1 \ \& \ \exists v_2[D(v_2) = d+1 \ \& \ D(v_1, v_2) = d]\}$. From Lemma 5.6, we may assume $P(u_1, u_2, r, d) = 0$. In that case, $D(v_1, v_2) = d+1$ for all tasks $v_1, v_2 \in G(u_1, u_2, r, d+1) \backslash G(u_1, u_2, r, d)$. Hence $|V| \leq 1$. So $P(u_1, u_2, r, d) = \max\{0, |V| - 1\}$. □

Now we will analyse the complexity of the deadline modification algorithm. Let G be a graph in which every task has been assigned a deadline. If every release date is zero, we may assume that the maximum deadline is bounded by n. In general, this is not the case, so some release dates and deadlines may be quite large. As a result, the deadline modification algorithm has to consider a lot of triples (u, r, d) and quadruples (u_1, u_2, r, d). Furthermore, the sets $G(u, r, d)$, $H(u, r, d)$ and $G(u_1, u_2, r, d)$ might change during the deadline modification. So some triples or quadruples need to be considered more than once. However, we will show that no triple and no quadruple has to be taken into account twice and that only $O(n)$ values of r and d need to be considered.

It is not difficult to see that this loop structure allows every triple (u, r, d) and every quadruple (u_1, u_2, r, d) to be considered only once. The values of $|G(u, r, d)| + P(u, r, d)$ and $|H(u, r, d)| + P(u, r+2, d)$ and the modified deadline imposed on u do not depend on the original deadline of u. In addition, $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d)$ and the deadlines computed for (u_1, u_2) and (u_2, u_1) are independent of the (original) deadlines of u_1, u_2 and (u_1, u_2). So, as long as $|G(u, r, d)| + P(u, r, d)$, $|H(u, r, d)| + P(u, r+2, d)$ and $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d)$ remain unchanged, an extra consideration of the triple (u, r, d) or the quadruple (u_1, u_2, r, d) does not result in a modification that has not already occurred in the first consideration.

$|G(u, r, d)|$, $|H(u, r, d)|$ and $|G(u_1, u_2, r, d)|$ can only change if a node w exists whose original deadline is greater than d and whose modified deadline is at most d. $P(u, r, d)$, $P(u, r+2, d)$ and $P(u_1, u_2, r, d)$ only get modified if the deadline of a pair of tasks (w_1, w_2) is decreased, such that its modified deadline is at most d. Since the outer loop considers the values of d in decreasing order, the modifications causing $|G(u, r, d)| + P(u, r, d)$, $|H(u, r, d)| + P(u, r+2, d)$ or $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d)$ to change have already occurred. So no triple or quadruple needs to be considered more than once.

Furthermore, it is not difficult to see that only $O(n)$ values of r have to be considered. For fixed u and d at most $n+2$ values of r need to be taken into account, namely $D(u)$, $D(u) - 2$ and the release dates of the nodes v for which $R(u) \leq R(v) \leq D(u)$. Suppose r is a value, not one of the at most $n+2$ indicated ones, which causes $D(u)$ to be modified. In that case, $d \geq r$ and $|G(u, r, d)| + P(u, r, d) \geq m(d-r)$ or $d > r+1$ and $|H(u, r, d)| + P(u, r+2, d) \geq m(d-(r+2))+2$. Suppose $d \geq r$ and $|G(u, r, d)| + P(u, r, d) \geq m(d-r)$. Let r' be the smallest release date exceeding r or $D(u)$, whichever is the smallest. Since $r' > r$, $|G(u, r', d)| + P(u, r', d) = |G(u, r, d)| + P(u, r, d) \geq m(d-r) > m(d-r')$. So the same modification will occur when (u, r', d) is considered.

Otherwise, suppose $d > r+1$ and $|H(u, r, d)| + P(u, r+2, d) \geq m(d-(r+2))+2$ for some r that is not one of the restricted set of values. Let r' be the smallest of the smallest release date exceeding r and $D(u) - 2$. $r' > r$, so $|H(u, r', d)| + P(u, r'+2, d) = |H(u, r, d)| + P(u, r+2, d) \geq m(d-(r+2))+2 > m(d-(r'+2))+2$. So the same modification occurs when considering one of the indicated values.

If we assume that initially $D(u_1, u_2) = \min\{D(u_1), D(u_2)\}$ for all tasks u_1 and u_2, the deadline of a pair of tasks (u_1, u_2) has to be modified only when the deadline of u_1 or u_2 has changed. So only $O(n)$ values of r need to be considered during the deadline modification.

It is more complicated to show that only $O(n)$ values of d have to be considered. Three extra constraints are introduced to prove this. For all tasks u_1, u_2 (1) If u_1 is a parent of u_2, then $D(u_1) \leq D(u_2)$; (2) $D(u_1, u_2) \leq \min\{D(u_1), D(u_2)\}$; and (3) initially, $D(u_1, u_2) = \min\{D(u_1), D(u_2)\}$. The next value of d that will be considered is the largest deadline (of a task or a pair of tasks) that has not been considered earlier. Suppose no tasks and pairs of tasks with deadline d remain

after considering all triples (u, r, d) and all quadruples (u_1, u_2, r, d). Some task or pair of tasks with deadline d was the last to get its deadline changed.

Suppose (u_1, u_2) had deadline d and when its deadline was changed, no other pair and no task had deadline d. $|G(u_1, u_2, r, d)| + P(u_1, u_2, r, d) \geq m(d-r)+1$ for some r. Every common successor of u_1 and u_2 has a deadline $> d$, so $G(u_1, u_2, r, d)$ does not contain any common successors of u_1 and u_2. (u_1, u_2) is the only pair having deadline d. Consequently, $P(u_1, u_2, r, d) = 0$. Therefore $G(u_1, u_2, r, d)$ contains at least $m(d - r) + 1$ tasks with release dates $\geq r$ and deadlines $\leq d - 1$. These have to be executed in the interval $[r, d - 1]$, which is impossible. Hence no 0-optimal schedule exists.

Otherwise, assume a task u with deadline d was the last to get its deadline modified. When $D(u)$ was decreased, no other task or pair of tasks had deadline d. $D(u)$ is changed, because $|G(u, r, d)| + P(u, r, d) \geq m(d-r)$ or $d > r+1$ and $|H(u, r, d)| + P(u, r+2, d) \geq m(d-(r+2))+2$ for some r. Suppose $|G(u, r, d)| + P(u, r, d) \geq m(d - r)$. Since u has deadline d and there are no other tasks or pairs of tasks with deadline d, $G(u, r, d)$ does not contain any successors of u and $P(u, r, d) = 0$. So $G(u, r, d)$ contains $\geq m(d - r) > m(d - r - 1)$ tasks with release dates $\geq r$ and deadlines $\leq d - 1$. Only $m(d - r - 1)$ tasks can be executed in the interval $[r, d - 1]$. As a result, no valid schedule for G is 0-optimal.

Suppose $d > r + 1$ and $|H(u, r, d)| + P(u, r + 2, d) \geq m(d - (r + 2)) + 2$. There are no pairs of tasks with deadline d, so $P(u, r + 2, d) = 0$. Because every successor of u has deadline $\geq d + 1$, $H(u, r, d)$ contains at least $m(d-(r+2))+2$ tasks with release dates $\geq r+2$ and deadlines $\leq d-1$. These must be executed in the interval $[r+2, d-1]$, which can contain at most $m((d-1)-(r+2))$ tasks. Consequently, no 0-optimal schedule exists.

So the deadline modification can be terminated if after considering all triples (u, r, d) and all quadruples (u_1, u_2, r, d) for some d, no task or pair of tasks with deadline d remains. Since $D(u_1, u_2) = \min\{D(u_1), D(u_2)\}$ or $D(u_1, u_2) = \min\{D(u_1), D(u_2)\} - 1$ for all tasks u_1 and u_2, only $O(n)$ values of d have to be considered by the deadline modification algorithm.

Because of the extra constraints, the deadline of the pair (u_1, u_2) has to be changed only when either $D(u_1)$ or $D(u_2)$ is changed. It is not difficult to see that at most one deadline of a task gets modified for fixed u, d. Let r_0 be the smallest value of r such that $|G(u, r, d)| + P(u, r, d) \geq m(d-r)$ or $|H(u, r, d)| + P(u, r + 2, d) \geq m(d - (r + 2)) + 2$ and $d > r + 1$. If $|G(u, r_0, d)| + P(u, r_0, d) \geq m(d - r_0)$, then $D(u) = d - \left\lceil \frac{1}{m}(|G(u, r_0, d)| + P(u, r_0, d)) \right\rceil \leq r_0$. So $D(u) \leq r_0$. Therefore $D(u)$ can only be changed when considering a value $r \leq r_0$. Since the values of r are selected in increasing order, no other modification of $D(u)$ occurs.

Otherwise, if $d > r_0 + 1$ and $|H(u, r_0, d)| + P(u, r_0 + 2, d) \geq m(d - (r_0 + 2)) + 2$, then $D(u) = d - 1 - \left\lceil \frac{1}{m} |H(u, r_0, d)| + P(u, r_0 + 2, d) - 1 \right\rceil \leq r_0$. Therefore all other values of r for which the algorithm modifies the deadline of u have been considered earlier.

So for fixed u and d at most one modification of a deadline of an individual task occurs. Consequently, $O(n)$ deadlines of pairs get modified for every u and d. Hence $O(n^3)$ times a deadline is changed.

When the deadline modification part considers a triple (u, r, d) or a quadruple (u_1, u_2, r, d), no task with deadline $\geq d + 1$ will be taken into account afterward. The same holds for pairs consisting of two tasks with deadlines $\geq d + 1$. Therefore the deadline of a pair of two tasks whose deadline is $d + 1$ remains unchanged. So $P(u, r, d)$ and $P(u_1, u_2, r, d)$ do not change after the deadline modification algorithm considers the first task or pair of tasks with deadline d.

From Corollary 5.7, if G is an interval order, then $P(u_1, u_2, r, d) = \max\{0, |V| - 1\}$, where V is the set of tasks v that are common successors of u_1 and u_2 or have release date $\geq r$ such that $D(v) = d + 1$ and $D_{min}(v) = d$. As a result, $P(u_1, u_2, r, d)$ can be computed in $O(n)$ time. The same, obviously, holds for the sets $G(u, r, d)$, $H(u, r, d)$ and $G(u_1, u_2, r, d)$. Since $O(n^3)$ deadline modifications occur, the deadline modification algorithm determines the modified deadlines in $O(n^4)$ time.

Because of the general definition of availability, it is possible to use an algorithm similar to

the algorithm for scheduling to meet deadlines only. We use the same definition of priority as in the previous section. One extra notation is introduced: R_0 denotes the smallest release date of an unscheduled task. Figure 5 shows the list scheduling algorithm.

```
LIST SCHEDULING ALGORITHM()
1  assume L = (u_1, ..., u_n) is a priority list
2  t = 0
3  while L contains unscheduled tasks
4  do R_0 = ∞
5     for i = 1 to n
6     do if u_i is unscheduled and available at time t
7           then schedule u_i at time t
8           else  R_0 = min{R_0, R(u_i)}
9     t = max{t + 1, R_0}
```

Figure 5: The list scheduling algorithm

For every time slot the priority list is traversed to find available tasks. This requires $O(n)$ time for every time t. So the algorithm uses $O(nT)$ time to assign a starting time to every task, where T is the number of values of t considered by the algorithm.

If, during the execution of the algorithm, t never increases by more than 1, the constructed schedule has length $O(n)$, so $T = O(n)$. Suppose $t_1, \ldots, t_k$ and $t'_1, \ldots, t'_k$ are values of t considered during the assignment of starting times, such that t'_i is the last value considered before t_i and $t'_i \leq t_i - 2$ for all i, $1 \leq i \leq k$. Define $V_i = \{u \in G \mid t_{i-1} \leq R(u) < t_i\}$, where $t_0 = 0$. Only $O(|V_i|)$ values of t are considered during the assignment of starting times to the tasks of V_i. So the total number of values of t considered by the algorithm is $O(n)$. So assigning starting times takes $O(n^2)$ time.

The list scheduling algorithm constructs 0-optimal schedules for interval orders if such schedules exist.

Theorem 5.8. *Let G be an interval order in which every pair of tasks has a modified deadline. Let L be a priority list. If a 0-optimal schedule for G exists on m processors, then every pair of tasks meets its deadline in the schedule on m processors for G constructed by the list scheduling algorithm using L.*

Proof. Let G be an interval order in which every pair of tasks has a modified deadline. Let L be a priority list. Assume a 0-optimal schedule for G exists on m processors. Let S be the schedule on m processors for G constructed by the list scheduling algorithm using L. Suppose in S not every pair meets its deadline. Let S_t be the first time slot containing a task in a pair violating its deadline. Assume (u_1, u_2) is a pair violating its deadline such that $u_1 \in S_t$ and u_2 is scheduled at time $\geq t$. In that case, $D(u_1, u_2) \leq t$. There are three possibilities: $D(u_1) \leq t$, $D(u_2) \leq t$ or $D(u_1) = D(u_2) = t + 1$ and $D(u_1, u_2) = t$.

Case 1. $D(u_1) \leq t$. Since a 0-optimal schedule for G exists, there are at most mt tasks with deadline $\leq t$. Therefore there is a time slot before S_t that is idle or contains a task with deadline $\geq t + 1$. Let $S_{t'-1}$ be the last such time slot. Define $G_1 = \bigcup_{i=t'}^{t-1} S_i \cup \{u_1\}$. All tasks in G_1 have a deadline at most t. No task of G_1 was available at time $t' - 1$. Let u be a source of G_1. u was not available at time $t' - 1$, because one of the following conditions is satisfied.

1. u has a release date $\geq t'$.
2. $S_{t'-1}$ contains a parent of u.

3. $S_{t'-2}$ contains two parents of u.
4. $S_{t'-2}$ contains a parent u' of u and $S_{t'-1}$ contains another child of u'.

Hence every task in G_1 has a release date $\geq t'$ or is a successor of $S_{t'-2} \cup S_{t'-1}$.

Case 1.1. Every task in G_1 with release date $\leq t'-1$ is a successor of $S_{t'-1}$. Define $V = \{w \in S_{t'-1} \mid w \text{ is a parent of } G_1\}$. If $V = \emptyset$, then every task in G_1 has release date $\geq t'$. In that case, G_1 contains $m(t-t')+1$ tasks that have to be scheduled in the interval $[t', t]$, which is impossible. So we may assume $V \neq \emptyset$. From Proposition 3.3, V contains a task w_1 such that $\{w \in G_1 \mid R(w) \leq t'-1\} \subseteq Succ(w_1)$. Clearly, $G_1 \subseteq G(w_1, t', t)$. Therefore $|G(w_1, t', t)| \geq m(t-t')+1$. With (5),

$$\begin{aligned} D(w_1) &\leq t - \lceil \tfrac{1}{m}(m(t-t')+1) \rceil \\ &= t - (t-t'+1) \\ &= t'-1. \end{aligned}$$

So w_1 is not completed before its deadline. Contradiction.

Case 1.2. G_1 contains a task that is no successor of $S_{t'-1}$ and has a release date $\leq t'-1$. Define $V = \{w \in S_{t'-2} \cup S_{t'-1} \mid w \text{ is a parent of } G_1\}$. Proposition 3.3 states that V contains a task w_1 that is a predecessor of every task in G_1 with release date $\leq t'-1$. Define $V' = V \setminus \{w_1\}$.

Case 1.2.1. Every task in G_1 with release date $\leq t'-1$ is a successor of V'. Using Proposition 3.3, we find that V' contains a task w_2 that is a predecessor of every task in G_1 with release date $\leq t'-1$. Therefore $G_1 \subseteq G(w_2, t', t)$. Also $G_1 \subseteq G(w_1, t', t)$, so $G_1 \subseteq G(w_1, w_2, t', t)$. Hence $|G(w_1, w_2, t', t)| + P(w_1, w_2, t', t) \geq m(t-t')+1$. From (8), $D(w_1, w_2) \leq t'-2$. So (w_1, w_2) violates its deadline. Contradiction.

Case 1.2.2. G_1 contains a task with release date $\leq t'-1$ that is not a successor of V'. Let v be such a task. Assume v is a source of G_1. V' does not contain a parent of v. V, however, does. So w_1 is a parent of v. Since G_1 contains a task that has a release date $\leq t'-1$ and is no successor of $S_{t'-1}$, w_1 is scheduled at time $t'-2$. $S_{t'-2}$ does not contain another parent of v, so $S_{t'-1}$ contains another child v' of w_1. v' is scheduled before v, so v' occurs before v in L. Hence $D(v') \leq D(v) \leq t$. So $G_1 \cup \{v'\} \subseteq H(w_1, t'-2, t)$. So $|H(w_1, t'-2, t)| + P(w_1, t', t) \geq m(t-t')+2$. From (6),

$$\begin{aligned} D(w_1) &\leq t - 1 - \lceil \tfrac{1}{m}(m(t-t')+1) \rceil \\ &= t - 1 - (t-t'+1) \\ &= t'-2. \end{aligned}$$

So w_1 is late. Contradiction.

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

Case 3. $D(u_1) = D(u_2) = t+1$ and $D(u_1, u_2) = t$. Assume $Succ(u_1) \subseteq Succ(u_2)$. Let U be the set of tasks containing all tasks whose priority is at least as high as that of u_1. U contains at least two tasks, because $u_1, u_2 \in U$. Let v_1 and v_2 be two tasks of U. If $D(v_1) \leq t$ or $D(v_2) \leq t$, then $D(v_1, v_2) \leq t$. So we will assume $D(v_1) = D(v_2) = t+1$. $Succ(u_1) \subseteq Succ(v_1), Succ(v_2)$, so, using Lemma 5.5 twice, we get $D(v_1, v_2) = t$. So $D(v_1, v_2) \leq t$ for every $v_1 \neq v_2$ in U. In order to meet every deadline, at most one task of U may be executed at time $\geq t$. Since a 0-optimal schedule for G exists, U contains at most $mt+1$ tasks. Consequently, there is a time slot before S_t that contains at most $m-1$ tasks whose priority is at least as high as that of u_1. Let $t'-1$ be the last such time. Define $G_2 = \bigcup_{i=t'}^{t-1} S_i \cup \{u_1, u_2\} \cup \{v \in \bigcup_{i \geq t} S_i \mid v \prec u_2\}$. G_2 contains at least $m(t-t')+2$ tasks whose priority is at least as high as that of u_1. No task of G_2 was available at time $t'-1$. Let u be a source of G_2. u was not available at time $t'-1$, because one of the following conditions is satisfied.

1. u has a release date $\geq t'$.
2. $S_{t'-1}$ contains a parent of u.
3. $S_{t'-2}$ contains two parents of u.
4. $S_{t'-2}$ contains a parent u' of u and $S_{t'-1}$ contains another child of u'.

So every task of G_2 with release date $\leq t'-1$ has a predecessor in $S_{t'-2}$ or $S_{t'-1}$.

Case 3.1. Every task in G_2 with release date $\leq t'-1$ is a successor of $S_{t'-1}$. Define $V = \{w \in S_{t'-1} \mid w \text{ is a parent of } G_2\}$. If $V = \emptyset$, then every task in G_2 has release date $\geq t'$. In that case, G_2 contains $\geq m(t-t')+2$ tasks with priority as least as high as u_1. Applying Lemma 5.5 twice, we obtain $D(v_1, v_2) \leq t$ for every $v_1 \neq v_2$ in G_2. So $m(t-t')+1$ tasks of G_2 have to be executed in the interval $[t', t]$, which is impossible. Hence we may assume $V \neq \emptyset$. From Proposition 3.3, V contains a task w_1 such that $\{w \in G_2 \mid R(w) \leq t'-1\} \subseteq Succ(w_1)$. Using Lemma 5.3, we obtain $|G(w_1, t', t)| + P(w_1, t', t) \geq |G_2| - 1 = m(t-t')+1$. With (5),

$$\begin{aligned} D(w_1) &\leq t - \left\lceil \tfrac{1}{m}(m(t-t')+1) \right\rceil \\ &= t - (t-t'+1) \\ &= t'-1. \end{aligned}$$

So w_1 is not finished before its deadline. Contradiction.

Case 3.2. G_2 contains a task that is no successor of $S_{t'-1}$ and has a release date $\leq t'-1$. Define $V = \{w \in S_{t'-2} \cup S_{t'-1} \mid w \text{ is a parent of } G_2\}$. From Proposition 3.3, V contains a task w_1 that is a predecessor of every task in G_2 with release date $\leq t'-1$. Let $V' = V \setminus \{w_1\}$.

Case 3.2.1. Every task in G_2 with release date $\leq t'-1$ has a predecessor in V'. With Proposition 3.3, V' contains a task w_2 that is a predecessor of every task in G_2 with release date $\leq t'-1$. Since every task in G_2 has priority at least as high as u_1, $D(v_1, v_2) \leq d$ for all $v_1 \neq v_2$ in G_2. Using Lemma 5.3, we obtain $|G(w_1, w_2, t', t)| + P(w_1, w_2, t', t) \geq m(t-t')+1$. From (8), $D(w_1, w_2) \leq t'-2$. So (w_1, w_2) violates its deadline. Contradiction.

Case 3.2.2. G_2 contains a task with release date $\leq t'-1$ that is no successor of V'. Let v be such a task. Assume v is a source of G_2. V' does not contain a parent of v. V, however, does. So w_1 is a parent of v. No parent of v is executed at time $t'-1$, so w_1 is scheduled at time $t'-2$. No other parent of v is scheduled at time $t'-2$. Hence $S_{t'-1}$ contains another child v' of w_1. v' is scheduled before v, so v' occurs before v in L. So $D(v') \leq D(v)$. Define $G_2' = G_2 \cup \{v'\}$. Every task in G_2' has priority as least as high as u_1. By applying Lemma 5.5 twice, we get $D(v_1, v_2) \leq t$ for every $v_1 \neq v_2$ in G_2'. Using Lemma 5.4, we obtain $|H(w_1, t'-2, t)| + P(w_1, t', t) \geq |G_2'| - 1 = m(t-t')+2$. From (6),

$$\begin{aligned} D(w_1) &\leq t - 1 - \left\lceil \tfrac{1}{m}(m(t-t')+1) \right\rceil \\ &= t - 1 - (t-t'+1) \\ &= t'-2. \end{aligned}$$

So w_1 is not completed before its deadline. Contradiction.

□

Corollary 5.9. *Let G be an interval order in which every pair of tasks has a deadline. If a 0-optimal schedule for G exists on m processors, then the schedule on m processors for G constructed by the above-mentioned algorithm is 0-optimal.*

Proof. Obvious. □

The algorithm for scheduling with only deadlines defined in the previous section finds optimal schedules, because its deadline modification part does not depend on the individual deadlines. The deadline modification part of the above-mentioned algorithm does: the deadline of a task u can only be modified if $R(u) \leq D(u)$. Therefore the algorithm above does not find optimal schedules.

It is, however, possible to find optimal schedules using the above-mentioned algorithm. Let G be an interval order. Define $l_0 = \max\{0, \max_u R(u) - D(u) + 1\}$. The lateness of an optimal schedule for G is at least l_0. Let u be a task of G. In an optimal schedule on m processors, u is scheduled at time $\leq R(u) + n - 1$. Hence the lateness of u is at most $R(u) + n - 1 + 1 - D(u) \leq l_0 + n - 1$. So the lateness of an optimal schedule is at least l_0 and at most $l_0 + n - 1$.

If l is added to every deadline and the algorithm is applied to the resulting interval order, a schedule with lateness at most l is constructed, provided that such a schedule exists. So an optimal schedule can be found in $O(n^4 \log n)$ time with bisection search.

Concluding remarks

The deadline modification parts of the algorithms defined by Verriet [12] only consider a task and its successors. The knowledge that at most one parent of a task can be scheduled immediately before this task is not used. As a result, these algorithms find optimal schedules for a small class of graphs.

A solution for this problem presented by Verriet [12] considered a special class of precedence graphs: in these graphs, every task u has a parent, its least urgent parent, which has to be scheduled after every other parent of u.

Another approach to this problem was taken in this report: every pair of tasks is assigned a deadline. This approach can be generalised to other classes of graphs, but, in general, the schedules constructed by the algorithms defined in this report for other classes of graphs are not 0-optimal. The class of the outforests is an exception. However, for outforests the introduction of deadlines for pairs of tasks does not add any information. Besides, more efficient algorithms for scheduling outforests were defined by Verriet [12].

References

[1] H.H. Ali and H. El-Rewini. The time complexity of scheduling interval orders with communication is polynomial. *Parallel Processing Letters*, 3(1):53–58, 1993.

[2] 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.

[3] 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.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via algorithmic progressions. In *Proceedings of the 19th Annual Symposium on the Theory of Computation*, pages 1–6, 1987.

[5] D. Dolev and M.K. Warmuth. Profile scheduling of opposing forests and level orders. *SIAM Journal of Algebraic and Discrete Methods*, 6(4):665–687, October 1985.

[6] M.R. Garey and D.S. Johnson. Scheduling tasks with nonuniform deadlines on two processors. *Journal of the ACM*, 23(6):461–467, July 1976.

[7] M.R. Garey and D.S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6(3):416–426, September 1977.

[8] T.C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9(6):841–848, 1961.

[9] 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.

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

[11] J.D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[12] J. Verriet. Scheduling UET, UCT dags with release dates and deadlines. Technical Report UU-CS-1995-31, Department of Computer Science, Utrecht University, September 1995.