Scheduling UET, UCT dags with release dates and deadlines

Jacques Verriet

UU-CS-1995-31
September 1995

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : + 31 - 30 - 531454

Scheduling UET, UCT dags with release dates and deadlines

Jacques Verriet

Technical Report UU-CS-1995-31
September 1995

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

Scheduling UET, UCT dags 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

The problem of scheduling precedence graphs for which every task has to be executed in a non-uniform interval is considered, with interprocessor communication delays. For the following classes of graphs we will present a polynomial time algorithm that finds minimum-lateness schedules.

1. Outforests on two processors.
2. Series-parallel graphs and opposing forests with the least urgent parent property (to be defined) on two processors.
3. Inforests with the least urgent parent property on m processors.

1 Introduction

The problem of scheduling a set of tasks under a precedence relation has been studied for a long time. The objective of most of the examined subproblems is finding the shortest schedule of a precedence graph on a set of fully connected processors. Many of these subproblems have been shown to be NP-complete [15]. Only for very restricted problems efficient algorithms are known: the execution lengths of all tasks are equal, the communication delays for information exchange are neglected, the number of processors is two [4], or special classes of precedence graphs are considered [9, 13, 16].

In parallel architectures large delays occur before the result of the execution of a task on one processor can be used by a task on another processor. If these communication delays are not neglected, the scheduling problems known to be solvable in polynomial time are even more restricted. The objective of these problems is finding a shortest schedule for special classes of graphs on two processors or on an unrestrictively large number of processors such that the communication delays do not exceed the execution time of a task [11, 14, 1].

In this report another constraint is added to the scheduling problem. Every task has to be executed in a non-uniform time interval: each task must be executed after a given release date and must be completed before a given deadline. To our knowledge the problem for precedence graphs with non-uniform deadlines and communication delays has not been considered before.

The problem of scheduling with release dates and deadlines is a generalisation of scheduling to minimise the makespan, that is scheduling with a uniform release date and a uniform deadline. Hence studying scheduling problems with release dates and deadlines increases the insight into other scheduling problems. For instance it is unknown if a minimum-length schedule on two processors for an arbitrary precedence graph with unit execution times and unit communication delays can be constructed in polynomial time. Several problems closely related to this problem are studied in this report.

Furthermore scheduling with release dates and deadlines can be useful when scheduling to minimise the schedule length. Some scheduling problems trying to find a shortest schedule on $m + 1$ processors can be transformed into a scheduling problem with release dates and deadlines

on m processors and vice versa. This can be done by completely filling one processor and translating the starting times and finishing times of the tasks scheduled on this processor into release dates and deadlines of the remaining tasks using the precedence constraints [7]. Release dates and deadlines can be eliminated by adding extra tasks and simulating the release dates and deadlines of the original tasks by adding precedence constraints between the original and the new tasks.

In this report several scheduling algorithms will be presented. These algorithms construct schedules for graphs, in which every task has been assigned a deadline and possibly a release date. The algorithms take communication delays into account. All algorithms have the same global structure. First the deadlines are modified such that they are consistent with the precedence constraints. The modified deadlines determine a priority list containing every task. A list scheduling algorithm assigns a starting time to every task using the priority list.

In the following section two scheduling algorithms are presented. It will be shown that the second algorithm constructs schedules on two processors for outforests in which no release date and no deadline is violated, if such schedules exist. The first algorithm works in the special case of uniform release dates, and is faster. For an outforest this algorithm constructs a schedule meeting every deadline, if such a schedule exists.

In section 3 an extra constraint on the modified deadlines is introduced. This extra constraint is called the least urgent parent property. In section 3 two algorithms for scheduling graphs with the least urgent parent property are presented. The first is an algorithm for scheduling a class of graphs, that is a superclass of the class of series-parallel graphs and of the class of opposing forests, on two processors. This algorithm constructs schedules in which no deadline is violated, if such schedules exist.

The other algorithm that will be presented in section 3, constructs schedules for inforests with the least urgent parent property on an arbitrary number of processors. This algorithm finds a schedule meeting every deadline in polynomial time, if such a schedule exists. Checking whether a schedule on m processors of length at most D exists for an inforest with unit execution times and unit communication delays is shown to be NP-complete by Lenstra, et al [12]. Hence minimising the schedule length for inforests with the least urgent parent property instead of inforests with arbitrary deadlines is a subproblem of this problem that is solvable in polynomial time.

Some preliminary definitions conclude this section. Let G be a graph. Throughout this report G will be used to denote a graph or the set of vertices of a graph. This will be clear from the context. G is a precedence graph, if G is a directed acyclic graph. From now on a graph G is a precedence graph containing n nodes and e edges. The nodes in a graph will also be called tasks. Let u, v be nodes of G. u is called a child of v, if (v, u) is an edge of G. If u is a child of v, v is called a parent of u. The outdegree (indegree) of u is the number of children (parents) of u. u is a child (parent) of a set of nodes V, if u is a child (parent) of a node in V. u is a predecessor of v, if nodes $u = u_0, \ldots, u_{k+1} = v$ $(k \geq 0)$, exist, such that $(u_0, u_1), (u_1, u_2), \ldots, (u_k, u_{k+1})$ are edges of G. In that case v is called a successor of u, which is denoted by $u \prec v$. Also u is a successor (predecessor) of a set of nodes V, if u is a successor (predecessor) of a node in V. Nodes without successors will be called sinks, nodes without predecessors will be called sources.

In this report we consider graphs, in which every node u has been assigned a deadline $D(u)$ and possibly a release date $R(u)$. In the rest of this report release dates are assumed to be nonnegative integers, deadlines are considered to be positive integers. If only deadlines are assigned to the tasks, the release date of each node is considered zero.

Algorithms will be presented for nonpreemptively scheduling with release dates and deadlines on a set of fully connected processors. Duplication of tasks is not allowed. Communication delays are not negligible: if v is a child of u and u and v are not executed on the same processor, then the execution of v can start unit time after the execution of u has finished.

A schedule for a graph G on m processors is a list of subsets of G. These subsets are called time slots. A schedule $S = (S_0, \ldots, S_{l-1})$ is a valid schedule for G, if the following six 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, $0 \leq t \leq l-1$.
4. If $u \prec v$, $u \in S_t$, and $v \in S_{t'}$, then $t < t'$.

The preceding four properties are satisfied for schedules in which communication delays are neglected and in which they are not. The following two properties are satisfied in schedules in which communication delays are taken into account. These need not be satisfied for schedules in which the communication delays are neglected.

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.

These properties do not contain information about which task is executed on which processor. Given a valid schedule on m processors for a graph G finding a correct assignment of a processor to every task takes $O(\min\{mn, n+e\})$ time.

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

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 property.

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

So in a partial schedule S the predecessors of a task that is scheduled in S, are scheduled in S as well.

Let $S = (S_0, \ldots, S_{l-1})$ be a partial schedule of a graph G on m processors. A node u of G, not scheduled in S, 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.

2 Scheduling outforests on two processors

Two algorithms were presented by Garey and Johnson [6, 7] for scheduling arbitrary graphs without communication delays on two processors: one for scheduling graphs to meet deadlines and the other for scheduling with release dates and deadlines. In this section two algorithms will be presented for scheduling on two processors with unit communication times.

2.1 Scheduling with deadlines

A scheduling algorithm will be presented that finds optimal schedules on two processors for outforests. The algorithm is similar to the algorithm by Garey and Johnson [6] for scheduling graphs with deadlines on two processors without communication delays. The algorithm consists of two steps. First the deadlines are modified, such that they are consistent with the precedence constraints. The modified deadlines are used to create a priority list containing all tasks. Using this

priority list a starting time will be assigned to every task.

In any valid schedule for a graph G on m processors a task is scheduled at an earlier time than its successors. Therefore the deadline of a task may be smaller than the deadlines of its successors. Let u be a task having $k \geq 1$ successors having a deadline $\leq d$. Because of communication delays only one of its children can be scheduled immediately after u, so the lower bound on the time to schedule u and its successors is $2 + \lceil \frac{k-1}{m} \rceil$. So u has to be completed at time $d - 1 - \lceil \frac{k-1}{m} \rceil$. The modification of the deadlines when scheduling graphs on m processors is done as follows.

While there are nodes not having a modified deadline, select a node u not having a modified deadline, such that all successors of u have been assigned a modified deadline. Assume $v_1, \ldots, v_k$ are the successors of u, such that $D(v_1) \leq \ldots \leq D(v_k)$. Then

$$D(u) = \min \left\{ D(u), \min_{1 \leq i \leq k} D(v_i) - 1 - \left\lceil \frac{i-1}{m} \right\rceil \right\}.$$

The following lemma shows the consistency of the modified deadlines, that is it will be shown that in every schedule meeting all original deadlines each modified deadline is met as well.

Lemma 2.1. *Let G be a graph in which every task has been assigned a deadline. Let S be a valid schedule for G on m processors. If S meets every original deadline, then S meets every modified deadline.*

Proof. Let G be a graph in which each task has been assigned a deadline. Let S be a valid schedule for G on m processors. Suppose S meets all original deadlines. Let u be a task of G. Induction is used to prove u meets its modified deadline. If u is a sink, $D(u)$ is not changed during the deadline modification, so u clearly meets its modified deadline. So we may assume u is not a sink. Suppose every successor of u meets its modified deadline. Let $v_1, \ldots, v_k$ be the successors of u such that $D(v_1) \leq \ldots \leq D(v_k)$. Let $1 \leq i \leq k$, u has i successors having a deadline $\leq D(v_i)$. Because of communication delays it takes at least $2 + \lceil \frac{i-1}{m} \rceil$ time slots to schedule u and its successors $v_1, \ldots, v_i$. So u must be scheduled at time $\leq D(v_i) - 2 - \lceil \frac{i-1}{m} \rceil$. So u is completed at time $\leq D(v_i) - 1 - \lceil \frac{i-1}{m} \rceil$. So u meets its modified deadline. □

After a modified deadline has been calculated for each task a priority list L is constructed. L contains all tasks ordered by nondecreasing deadlines. Using L every task is assigned a starting time in the following way. Initially S is the empty sequence. For every time slot S_t L is traversed to find as many tasks as possible, however at most m, that are available at time t. These tasks are assigned to S_t.

```
PRIORITY LIST SCHEDULING()
1  let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
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
```

The complexity of the above algorithm is dominated by the deadline modification algorithm. Let G be a graph in which each task has a deadline. By using counting sort $O(n)$ time is needed to create a priority list. Assigning starting times requires quadratic time. The length of the schedule constructed by the above algorithm is at most n. By introducing some extra variables it is possible to check whether a node is unscheduled and available in constant time. For every time slot the priority list is traversed once, so $O(n^2)$ time is used to find available tasks.

If G is a transitive closure, the deadline modification takes $O(e)$ time. For each node u the successors can be ordered by nondecreasing deadlines in $O(outdegree(u))$ time. The modification of the deadline of a task u takes constant time for each successor of u, so the modification of $D(u)$ requires $O(outdegree(u))$ time. Therefore the deadline modification takes $O(e)$ time.

If however G is not a transitive closure, the transitive closure of G has to be computed first. It takes $O(n^\alpha)$ time to build the transitive closure of G for some α. Coppersmith and Winograd [5] have bounded α by 2.376. Goralčíkova and Koubek [8] have shown that the transitive closure of G can be computed in $O(n + e + ne^-)$ time, where e^- is the number of edges in the transitive reduct of G. Clearly $e^- \leq e$. Since a forest has $O(n)$ edges, the transitive closure of an outforest can be computed in $O(n^2)$ time.

So the algorithm uses $O(n^2)$ time to schedule forests and transitive closures, and $O(n^\alpha)$ time for other graphs.

The following theorem shows that the above algorithm constructs a 0-optimal schedule on two processors for an outforest, if a 0-optimal schedule exists.

Theorem 2.2. *Let F be an outforest in which each task has been assigned a modified deadline. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. If a 0-optimal schedule for F on two processors exists, the schedule constructed by the above algorithm using L is 0-optimal.*

Proof. Let F be an outforest in which every task has been assigned a modified deadline. Let L be a priority list containing all tasks of F ordered by nondecreasing deadlines. Suppose a 0-optimal schedule for F on two processors exists. Let S be the schedule on two processors constructed by the algorithm using L. Suppose S is not 0-optimal. Assume t is the first time at which a task is scheduled that violates its deadline. Suppose $u \in S_t$ and $D(u) \leq t$. Since a 0-optimal schedule exists, there is a time slot before S_t containing less than 2 tasks having a deadline $\leq D(u)$. Let $S_{t'-1}$ be the last time slot before S_t that contains at most 1 task having a deadline $\leq D(u)$. Define $F' = \bigcup_{i=t'}^{t-1} S_i \cup \{u\}$. F' contains $2(t-t')+1$ tasks having a deadline $\leq D(u)$. At time $t'-1$ there are unscheduled tasks having a deadline $\leq D(u)$. Since every task has at most one parent, $S_{t'-1}$ contains exactly one task u' such that $D(u') \leq D(u)$.

Case 1. u' is a predecessor of all nodes in F'. Because of communication delays at most one child of u' can be scheduled at time t'. This implies $t = t'$ and u is a child of u'. So u' has at least one successor having a deadline $\leq D(u)$. So $D(u') \leq D(u) - 1 \leq t - 1 = t' - 1$. Since u' is scheduled at time $t'-1$, it violates its deadline. Contradiction.

Case 2. F' contains a task which is not a successor of u'. Let v be a node of F' that is not a successor of u'. Assume F' does not contain any predecessors of v. v was rejected at time $t'-1$. So v was not available at time $t'-1$ after u' had been scheduled. This implies u' and v have the same parent u'', which is scheduled at time $t'-2$. Therefore every task in F' is a successor of u''. So u'' has at least $2(t-t')+2$ successors having a deadline $\leq D(u)$. Therefore

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

Because u'' is scheduled at time $t'-2$, it violates its deadline. Contradiction.

□

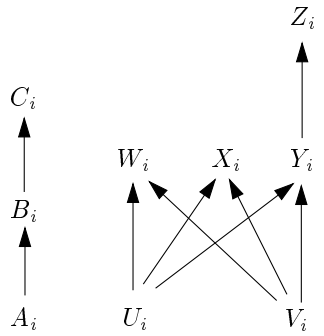
Let F be an outforest, in which every task u has been assigned a deadline $D_0(u)$. Let $D_1(u)$ denote the modified deadline of u. Let S be an optimal schedule for F on two processors. Suppose the lateness of S is l. Define $D_0'(u) = D_0(u) + l$ for every task u in F. Let F' denote the outforest F in which every task u has deadline $D_0'(u)$. In S every task meets deadline $D_0'(u)$. So the above algorithm finds a schedule for F' in which every task u is completed at time $D_0'(u)$.

First the algorithm computes a modified deadline $D_1'(u)$ for every task u. It is easy to see that $D_1'(u) = D_1(u) + l$ for every task u in F. So each priority list ordered by nondecreasing deadlines $D_1'(u)$ is also ordered by nondecreasing deadlines $D_1(u)$. The availability of a task does not depend on its deadline, so the schedule constructed by the above algorithm only depends on the priority list. So in the schedule for F constructed by the above algorithm on two processors every task u is completed at time $D_0'(u) = D_0(u) + l$. So this schedule is optimal.

Corollary 2.3. *Let F be an outforest in which every task has been assigned a deadline. The schedule for F constructed by the above algorithm on two processors is optimal.*

Proof. Obvious. □

The algorithm does not find optimal schedules for arbitrary graphs on 2 processors. Something stronger can be shown: for all k there are graphs such that a 0-optimal schedule on two processors exists and the above algorithm finds a schedule with lateness k. Figure 1 shows a graph in which all tasks have been assigned a modified deadline. This graph will be called G_i.



$$D(A_i) = 5i + 2, D(B_i) = 5i + 3, D(C_i) = 5i + 4$$

$$D(U_i) = 5i + 1, D(V_i) = 5i + 2$$

$$D(W_i) = 5i + 4, D(X_i) = 5i + 4, D(Y_i) = 5i + 4, D(Z_i) = 5i + 5$$

Figure 1: The graph G_i

U_0	V_0	W_0	X_0		U_1	V_1	W_1	X_1					U_k	V_k	W_k	X_k	
A_0	B_0	C_0	Y_0	Z_0	A_1	B_1	C_1	Y_1	Z_1				A_k	B_k	C_k	Y_k	Z_k

Figure 2: A 0-optimal schedule for G

U_0	A_0	B_0	C_0			U_1	A_1	B_1	C_1					U_k	A_k	B_k	C_k		
V_0		W_0	X_0	Y_0	Z_0	V_1		W_1	X_1	Y_1	Z_1			V_k		W_k	X_k	Y_k	Z_k

Figure 3: A schedule for G constructed by the above algorithm

Now we will consider the graph $G = G_0 \cup G_1 \cup \ldots \cup G_k$. Some extra edges are added to G: $(Y_i, U_{i+1}), (Y_i, V_{i+1}), (Y_i, A_{i+1})$ are edges of G for all i, $0 \leq i \leq k-1$. A schedule on two processors for G is shown in figure 2. This schedule is 0-optimal. Figure 3 shows a schedule constructed by the above algorithm using priority list $L = L_0, \ldots, L_n$, where $L_i = (U_i, V_i, A_i, B_i, C_i, W_i, X_i, Y_i, Z_i)$. This schedule is not 0-optimal: Y_i is scheduled at time $6i + 4$. Therefore the lateness of Y_i is

$6i + 5 - (5i + 4) = i + 1$. So the lateness of this schedule is $k + 1$.

It is however possible to prove an upper bound on the lateness of a task. This will be done by introducing an approximation algorithm finding schedules in which no task is scheduled before the time it is scheduled by the above algorithm.

Let G be a graph in which every task has been assigned a modified deadline. Let $N(t)$ denote the number of tasks scheduled at time t. Define $V(d) = \{u \in G \mid D(u) = d\}$ for all d, $1 \leq d \leq D$, where $D = \max_u D(u)$. The sets $V(d)$ will be called levels. The algorithm schedules the tasks such that every task of $V(d+1)$ is scheduled after every task of $V(d)$ for all d, $1 \leq d \leq D-1$. Every task u with deadline d has at most one child with deadline $d+1$. However a task having deadline $d+1$ can have more than one parent having deadline d. So if $|V(d)| \leq 1$, then the tasks of $V(d+1)$ can be executed in the time slot immediately after the last time slot containing a task of $V(d)$. If $|V(d)| \geq 2$, then a time slot is left empty between the last time slot containing a task of $V(d)$ and the first time slot containing a task of $V(d+1)$.

```
LEVEL SCHEDULING()
 1  let (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
 2  t = 0
 3  d = 1
 4  for i = 1 to n
 5      do while D(u_i) = d
 6             do if N(t) = m
 7                   then t = t + 1
 8                schedule u_i at time t
 9                N(t) = N(t) + 1
10         if |V(d)| ≤ 1
11           then t = t + 1
12           else t = t + 2
13         d = d + 1
```

Let G be a graph. Suppose an optimal schedule for G has lateness l. Let $L = (u_1, \ldots, u_n)$ be a list containing all tasks of G ordered by nondecreasing deadlines. It is easy to see that in the schedule constructed by the approximation algorithm using L no task is executed at an earlier time than in the schedule constructed by the list scheduling algorithm. Let S^1 be the schedule constructed by the list scheduling algorithm using L and S^2 the schedule constructed by the approximation algorithm using L. Let u be a task having deadline d. u is scheduled before all tasks having a deadline larger than d. Obviously

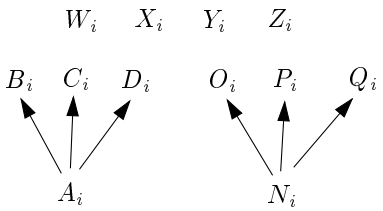
$$t \leq \sum_{i=1}^{d} \left\lceil \frac{1}{m} |V(i)| \right\rceil + \sum_{\substack{i=1 \\ |V(i)| \geq 2}}^{d-1} 1 - 1.$$

It is not difficult to see that t is maximal, if $|V(i)| = 2$ for all i, $1 \leq i \leq d-1$, G has $m(d+l)$ tasks having a deadline $\leq d$, and S^2_t is the last time slot of S^2 containing a task having deadline d. In that case

$$\begin{aligned} t &= d - 1 + \left\lceil \tfrac{1}{m}(m(d+l) - 2(d-1)) \right\rceil + d - 1 - 1 \\ &= 2d - 3 + \left\lceil d + l - \tfrac{2}{m}(d-1) \right\rceil \\ &= 3d + l - 3 - \left\lfloor \tfrac{2}{m}(d-1) \right\rfloor. \end{aligned}$$

So u is scheduled at time $\leq 3d + l - 3 - \left\lfloor \frac{2}{m}(d-1) \right\rfloor$. Therefore its lateness is at most $2d + l - 2 - \left\lfloor \frac{2}{m}(d-1) \right\rfloor$. The lateness of a task in S^2 is at least its lateness in S^1. So the lateness of a task with deadline d scheduled in S^1 is $\leq 2d + l - 2 - \left\lfloor \frac{2}{m}(d-1) \right\rfloor$. So the lateness of S^1 is at most $2D + l - 2 - \left\lfloor \frac{2}{m}(D-1) \right\rfloor$, where D is the maximum deadline of G.

Also the algorithm does not find optimal schedules when scheduling outforests on more than two processors. As was the case for scheduling arbitrary graphs on two processors, the lateness for schedules for outforests on more than two processors constructed by the above algorithm is unbounded. This also holds for schedules of outforests for which a 0-optimal schedule exists. Figure 4 shows an outforest, that will be called F_i. We consider the outforest $F = F_0 \cup \ldots \cup F_k$. In this graph we add some edges: $(Q_i, A_{i+1}), (Q_i, N_{i+1}), (Q_i, W_{i+1}), (Q_i, X_{i+1}), (Q_i, X_{i+1}), (Q_i, Y_{i+1})$, (Q_i, Z_{i+1}) are edges of F for all i, $0 \leq i \leq k-1$. Figure 5 shows a 0-optimal schedule on three processors. In figure 6 another schedule for F is shown. This schedule is constructed by the above algorithm, using priority list $L = L_0, \ldots, L_k$, where $L_i = (W_i, X_i, Y_i, Z_i, A_i, N_i, B_i, C_i, D_i, O_i, P_i, Q_i)$. In this schedule Q_i is executed at time $6i+4$ for every i. So the lateness of Q_i is $6i+5-(5i+4) = i+1$ for each i. So the lateness of this schedule is $k+1$.



$$D(A_i) = 5i+2, D(B_i) = 5i+4, D(C_i) = 5i+4, D(D_i) = 5i+4$$
$$D(N_i) = 5i+2, D(O_i) = 5i+4, D(P_i) = 5i+4, D(Q_i) = 5i+4$$
$$D(W_i) = 5i+1, D(X_i) = 5i+2, D(Y_i) = 5i+2, D(Z_i) = 5i+2$$

Figure 4: An outforest for which a 0-optimal schedule exists

W_0	X_0	B_0	O_0		A_1	B_1	C_1	P_1			A_k	B_k	C_k	P_k
A_0	Y_0	C_0	P_0		N_1	Y_1	D_1	Q_1		W_k	N_k	Y_k	D_k	Q_k
N_0	Z_0	D_0	Q_0	W_1	X_1	Z_1	O_1				X_k	Z_k	O_k	

Figure 5: A 0-optimal schedule for F

W_0	Z_0		C_0	Q_0	W_1	X_1	A_1	B_1	C_1	Q_1				W_k	X_k	A_k	B_k	C_k	Q_k
X_0	A_0	B_0	D_0			Y_1	N_1	O_1	D_1						Y_k	N_k	O_k	D_k	
Y_0	N_0	O_0	P_0			Z_1			P_1						Z_k			P_k	

Figure 6: A schedule for F constructed by the above algorithm

A similar construction can be used to prove that the algorithm by Garey and Johnson constructs schedules on three processors in which the lateness is unbounded, even for outforests for which a 0-optimal schedule exists.

An upper bound of $2d + l - 2 - \lceil \frac{2}{m}(d-1) \rceil$ on the lateness of a task having deadline d was proved before, where l is the lateness of an optimal schedule. However for outforests a better upper bound on the lateness can be proved. In an outforest every task has at most one predecessor, so the tasks having deadline $d+1$ can be executed immediately after the tasks having deadline d. Another approximation algorithm will be presented. This algorithm finds schedules in which the

levels are executed immediately after each other.

```
OUTFOREST LEVEL SCHEDULING()
 1  let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
 2  t = 0
 3  d = 1
 4  for i = 1 to n
 5     do if D(u_i) = d
 6           then while D(u_i) = d
 7                    do if N(t) = m
 8                          then t = t + 1
 9                       schedule u_i at time t
10                       N(t) = N(t) + 1
11                 t = t + 1
12        d = d + 1
```

Let F be an outforest for which an optimal schedule on m processors has lateness l. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. Let S^1 be the schedule for F constructed by the above algorithm using L. Let S^2 be the schedule created by the outforest level scheduling algorithm using L. It is not difficult to see that no task of F is executed at an earlier time in S^2 than in S^1. Let u be a task of F having deadline d. u is executed at time t after all tasks having a deadline $\leq d-1$. So

$$t \leq \sum_{i=1}^{d} \left\lceil \frac{1}{m} |V(i)| \right\rceil - 1.$$

It is easy to see that t is maximal, if $|V(i)| = 1$ for all i, $1 \leq i \leq d-1$, $|V(d)| = m(d+l) - (d-1)$ and t is the last time at which a task having deadline d is scheduled. In that case

$$\begin{aligned} t &= d - 1 + \left\lceil \tfrac{1}{m}(m(d+l) - (d-1)) \right\rceil - 1 \\ &= d - 2 + \left\lceil d + l - \tfrac{1}{m}(d-1) \right\rceil \\ &= 2d + l - 2 - \left\lfloor \tfrac{1}{m}(d-1) \right\rfloor . \end{aligned}$$

So its lateness is at most $d+l-1-\left\lfloor \frac{1}{m}(d-1) \right\rfloor$. The lateness of a task in S^2 is at least the lateness of this task in S^1, so the lateness of a task with deadline d scheduled in S^1 is $\leq d+l-1-\left\lfloor \frac{1}{m}(d-1) \right\rfloor$. So the lateness of S^1 is at most $D+l-1-\left\lfloor \frac{1}{m}(D-1) \right\rfloor$, where D is the maximum deadline of F.

So the lateness of a schedule for a graph G on m processors constructed by the above algorithm is at most $D+l-1-\left\lfloor \frac{1}{m}(D-1) \right\rfloor$, if G is an outforest, where $D = \max_u D(u)$ and l is the lateness of an optimal schedule for G on m processors. If G is not an outforest, the lateness of the schedule is at most $2D + l - 2 - \left\lfloor \frac{2}{m}(D-1) \right\rfloor$.

2.2 Scheduling with release dates and deadlines

Scheduling graphs in which every task has been assigned a deadline and a release date 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 part modifies all release dates and deadlines, the second part does the actual scheduling. The algorithm is similar to the algorithm of Garey and Johnson [7] for scheduling arbitrary graphs on two processors with release dates and deadlines without communication delays.

Because in every valid schedule a task u is scheduled after all its predecessors, the release date of u may exceed the release dates of all its predecessors. Therefore the release dates can be modified as follows.

While there are nodes not having a modified release date, select a node u not having a modified release date such that all parents $v_1, \ldots, v_k$ of u have been assigned a modified release date. Then

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

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. Two definitions are needed to define the deadline modification. Let G be a graph in which all tasks have deadlines and (modified) release dates. For all nodes u in G and integers r, d such that $R(u) \le r \le D(u) \le d$ define

$$G(u, r, d) = \{ v \in G \mid D(v) \le d \ \& \ (u \prec v \vee R(v) \ge r) \ \& \ u \ne v \}.$$

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

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

The sets $G(u, r, d)$ and $H(u, r, d)$ 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 can start execution not before r or after u has been scheduled. 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 $H(u, r, d)$. The following shows the deadline modifications due to $G(u, r, d)$ and $H(u, r, d)$.

Let u be a task and let r, d be integers such that $R(u) \le r \le D(u) \le d$.

1. If $|G(u, r, d)| \ge m(d - r)$, then

$$D(u) = \min \left\{ D(u), \ d - \left\lceil \frac{1}{m} |G(u, r, d)| \right\rceil \right\}.$$

2. If $d > r + 1$ and $|H(u, r, d)| \ge m(d - (r + 2)) + 2$, then

$$D(u) = \min \left\{ D(u), \ d - 1 - \left\lceil \frac{1}{m} (|H(u, r, d)| - 1) \right\rceil \right\}.$$

Let G be a graph. When a deadline modification occurs the sets $G(u, r, d)$ and $H(u, r, d)$ may get changed. This might cause some triples (u, r, d) to be considered more than once. However by carefully considering all triples, no triple needs to be considered twice and only $O(n^3)$ triples have to be taken into account.

The deadline modification algorithm consists of three nested loops. It is similar to the deadline modification algorithm by Garey and Johnson [7]. The outer loop selects values of d in decreasing order. For each value of d the middle loop selects nodes u, for which $D(u) \le d$, in order of increasing release dates. For fixed u, d the inner loop selects values r in increasing order. The inner loop modifies the deadline of u, if a modification condition holds. The deadline modification is done by the following algorithm.

```
DEADLINE MODIFICATION()
 1  let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing release dates
 2  D = max_i D(u_i)
 3  R = max_i R(u_i)
 4  for d = D downto 1
 5     do for i = 1 to n
 6           do if D(u_i) ≤ d
 7                 then for r = R(u_i) to R
```

8 **do if** $|G(u_i, r, d)| \geq m(d - r)$
9 **then** $D(u_i) = \min\left\{D(u_i), d - \left\lceil \frac{1}{m} |G(u_i, r, d)| \right\rceil\right\}$
10 **if** $d > r + 1$ **and** $|H(u_i, r, d)| \geq m(d - (r + 2)) + 2$
11 **then** $D(u_i) = \min\left\{D(u_i), d - 1 - \left\lceil \frac{1}{m}(|H(u_i, r, d)| - 1)\right\rceil\right\}$

It is not difficult to see that this loop structure allows every triple (u, r, d) to be considered only once. The values of $|G(u, r, d)|$ and $|H(u, r, d)|$ and the modified deadline imposed on u do not depend on the (original) deadline of u. So as long as $|G(u, r, d)|$ and $|H(u, r, d)|$ remain unchanged, an extra consideration of the triple (u, r, d) does not result in a modification, that has not already occurred in the first consideration of the triple. $|G(u, r, d)|$ and $|H(u, r, d)|$ can only change, if the deadline of some node v is modified, such that the (original) deadline of v is greater than d and the modified deadline is at most d. Since the outer loop considers the values of d in decreasing order, the modifications causing $|G(u, r, d)|$ or $|H(u, r, d)|$ to change have already occurred. So no triple (u, r, d) needs to be considered more than once.

Some deadlines and release dates may be quite large. So the above algorithm may need to consider a large number of triples. It is however possible to bound the number of values of r and d that need to be considered.

For fixed u, d at most $n + 2$ values of r need to be taken into account. It can be shown, that the values, that need to be considered are the release dates of the nodes v in the graph, for which $R(u) \leq R(v) \leq D(u)$, and $D(u)$ and $D(u) - 2$. Suppose r is a value, not one of the at most $n + 2$ indicated values, which causes $D(u)$ to be changed. In that case $d \geq r$ and $|G(u, r, d)| \geq m(d - r)$ or $d > r + 1$ and $|H(u, r, d)| \geq m(d - (r + 2)) + 2$. Suppose $d \geq r$ and $|G(u, r, d)| \geq m(d - r)$. Let r' be the smallest release date exceeding r or $D(u)$ whichever is the smallest. Since $r' > r$, $|G(u, r', d)| = |G(u, r, d)| \geq m(d - r) > m(d - r')$. So the same modification will occur when considering (u, r', d).

Otherwise suppose $d > r + 1$ and $|H(u, r, d)| \geq m(d - (r + 2)) + 2$ 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 $|H(u, r', d)| = |H(u, r, d)| \geq m(d - (r + 2)) + 2 > m(d - (r' + 2)) + 2$. So the same modification occurs when considering a value from the restricted set of values.

It is more complicated to prove that only $O(n)$ values of d need to be considered. The values d that need to be considered are the modified deadlines of all nodes u. To show this some preprocessing is required. The deadlines have to be changed, such that for all nodes u, v if $u \prec v$, then $D(u) \leq D(v)$. This property is easy to maintain and it does not violate the existence of 0-optimal schedules.

The next value of d to be considered is the largest current deadline less than the previously considered value of d. Obviously this assures all deadline modifications to occur. Suppose for some d no task having deadline d remains after considering all triples (u, r, d). It can be proved that no 0-optimal schedule exists. Let u be a task having deadline d, such that when its deadline is changed, no task having deadline d remains. Due to the constraint on the deadlines u has no successors with a deadline $\leq d$. $D(u)$ was modified, so either $|G(u, r, d)| \geq m(d - r)$ and $r \leq d$ or $|H(u, r, d)| \geq m(d - (r + 2)) + 2$ and $d > r + 1$ for some r. If $|G(u, r, d)| \geq m(d - r)$ and $r \leq d$, then $G(u, r, d)$ contains at least $m(d - r)$ tasks having a release date $\geq r$ and a deadline $\leq d$. Since u was the last task having a deadline d, every task in $G(u, r, d)$ has a deadline $\leq d - 1$. $|G(u, r, d)| \geq m(d - r) > m(d - 1 - r)$. In the interval $[r, d - 1]$ only $m(d - 1 - r)$ tasks can be executed. So no 0-optimal schedule exists.

If $|H(u, r, d)| \geq m(d - (r + 2)) + 2$ and $d > r + 1$, then $H(u, r, d)$ contains $\geq m(d - (r + 2)) + 2$ tasks having a release date $\geq r + 2$ and a deadline $\leq d$. Only $m(d - (r + 2))$ tasks can be executed in the interval $[r + 2, d]$, so no 0-optimal schedule exists.

Therefore no 0-optimal schedules are lost, if the deadline modification algorithm is terminated, when after considering d no tasks with deadline d remain. So only $O(n)$ values of d need to be considered.

So to modify the deadlines in a graph consisting of n nodes $O(n)$ values of r have to be considered in the inner loop of the algorithm and $O(n)$ values of d need to be selected in the outer loop. So only $O(n^3)$ triples need to be considered.

The following lemma shows the consistency of the modified deadlines.

Lemma 2.4. *Let G be a graph in which each task has been assigned a release date and a deadline. Let S be a valid schedule for G on m processors. If S meets all original deadlines, then each node meets its modified deadline.*

Proof. Let G be a graph in which each task has been assigned a release date and a deadline. Let S be a schedule for G on m processors. Suppose S meets all original deadlines. Let u be a task of G and let r, d be integers such that $R(u) \leq r \leq D(u) \leq d$. Suppose $D(u)$ was changed when (u, r, d) was considered, otherwise u obviously meets the deadline computed when the triple (u, r, d) was considered.

1. $|G(u, r, d)| \geq m(d - r)$.

   Case 1. $|G(u, r, d)| > m(d-r)$. Scheduling all tasks of $G(u, r, d)$ on m processors requires at least $\lceil \frac{1}{m} |G(u, r, d)| \rceil \geq d - r + 1$ time slots. So there is a task u' of $G(u, r, d)$ scheduled at time $\leq d - \lceil \frac{1}{m} |G(u, r, d)| \rceil \leq d - (d - r + 1) = r - 1$. Clearly u' is a successor of u, so u is completed at time $d - \lceil \frac{1}{m} |G(u, r, d)| \rceil$.

   Case 2. $|G(u, r, d)| = m(d - r)$. Define $G' = G(u, r, d) \cup \{u\}$. $|G'| = m(d - r) + 1$, so scheduling the tasks of G' takes at least $d - r + 1$ time slots. Hence G' contains a node u' scheduled at time $\leq r - 1$. u' is either a successor of u or u itself. In both cases u is completed at time $r = d - \lceil \frac{1}{m} |G(u, r, d)| \rceil$.

2. $d > r+1$ and $|H(u, r, d)| \geq m(d-(r+2))+2$. Let v_1, v_2 be two tasks of $H(u, r, d)$ scheduled at time t_1 and t_2 such that $t_1 \leq t_2$ and all other tasks of $H(u, r, d)$ are scheduled at time $\geq t_2$. Define $H' = H(u, r, d) \setminus \{v_1\}$. Then $|H'| = |H(u, r, d)|-1 \geq m(d-(r+2))+1$. It takes at least $\lceil \frac{1}{m} |H'| \rceil$ time slots to schedule all tasks of H'. v_2 is a task of H' which is an element of the first time slot containing a task of H'. So $t_2 \leq d - \lceil \frac{1}{m} |H'| \rceil \leq d - \lceil \frac{1}{m}(m(d - (r + 2)) + 1) \rceil \leq r+1$. So v_2 is a successor of u. Since $t_1 \leq t_2$, v_1 is also a successor of u. Due to communication delays u is scheduled at time $\leq t_2 - 2 \leq d-2 - \lceil \frac{1}{m}(|H(u, r, d)| - 1) \rceil$. Therefore u is completed at time $d - 1 - \lceil \frac{1}{m}(|H(u, r, d)| - 1) \rceil$.

So u meets its modified deadline computed when the triple (u, r, d) was considered. □

The assignment of a starting time to every task is done by an algorithm similar to the one presented in section 2.1 using a priority list. This is possible because of the way the availability of a task is defined. R_0 denotes the smallest release date of an unscheduled task.

```
PRIORITY LIST SCHEDULING()
 1  let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
 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  if u_i is unscheduled
 9                          then R_0 = min{R_0, R(u_i)}
10        t = max{t + 1, R_0}
```

The complexity of the above algorithm is dominated by the deadline modification. The other steps of the algorithm run in quadratic time. Clearly the release date modification takes $O(e)$ time and the creation of the priority list requires $O(n)$ time.

The algorithm assigning starting times is similar to the algorithm for scheduling graphs with only deadlines. Because some release dates might be quite large, we may not assume that t is at most n, if S_t is a non-empty time slot of the schedule S constructed for G. To obtain an algorithm with a quadratic running time R_0 was introduced. Obviously 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 $O(n)$ values of t are considered. 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 deadline modification algorithm has to consider all triples (u, r, d). Let G be a graph, assume G is a transitive closure. As was explained above only $O(n^3)$ triples have to be taken into account in each step of the deadline modification algorithm.

It can also be shown that only $O(n^2)$ deadline modifications occur during the execution of the deadline modification algorithm: at most one deadline modification occurs for fixed u and d. Let r_0 be the smallest value of r such that $|G(u, r, d)| \geq m(d - r)$ or $|H(u, r, d)| \geq m(d - (r + 2)) + 2$ and $d > r + 1$. If $|G(u, r_0, d)| \geq m(d - r_0)$, then $D(u) = d - \left\lceil \frac{1}{m} |G(u, r_0, d)| \right\rceil \leq r_0$. Since the values of r are selected in increasing order, all values of r such that $|G(u, r, d)|$ causes $D(u)$ to be modified (these values are at most r_0), have already been considered.

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

Furthermore $O(n^3)$ time is required to compute and update $|G(u, r, d)|$ and $|H(u, r, d)|$ and to maintain the extra constraint on the deadlines. The maintenance of the constraint requires linear time after each deadline modification, so this can be done in $O(n^3)$ time. The computation and updating of $|G(u, r, d)|$ and $|H(u, r, d)|$ is done by the deadline modification algorithm. This is only described for $|G(u, r, d)|$, computation and updating of $|H(u, r, d)|$ can be done in a similar way.

Assume $u_1, \ldots, u_n$ is the list of tasks sorted by nondecreasing modified release dates. For fixed u and d the middle loop computes $C = |G(u, R(u), d)|$ and $i_0 = \min \{i \mid R(u_i) = R(u)\}$. Set $R = R(u)$.

After each execution of the inner loop i_0 is incremented by 1 until either $i_0 > n$ or $R(u_{i_0}) > R$, if $R < D(u)$. Subtract 1 from C for each node u_j such that $i'_0 \leq j < i_0$, where i'_0 is the old value of i_0 and $u \not\prec u_j$, $D(u_j) \leq d$, and $R(u_j) = R$. Now $C = |G(u, R', d)|$, where R' is the smallest of d and the smallest release date exceeding R. Set $R = R'$ and continue.

This way $O(n^2)$ values $|G(u, r, d)|$ and $|H(u, r, d)|$ are computed. These computations each require linear time. The computation of i_0 and R also takes $O(n)$ time. In the inner loop i_0, R and C are updated by adding or subtracting 1. For fixed u and d at most n additions and subtractions occur. Therefore the algorithm computing the modified deadlines requires $O(n^3)$ time.

The following theorem shows the algorithm for scheduling outforests with release dates and deadlines constructs 0-optimal schedules on two processors, if these exist.

Theorem 2.5. *Let F be an outforest in which every task has a modified release date and a modified deadline. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. If a 0-optimal*

schedule for F on two processors exists, the schedule constructed by the above algorithm using L is 0-optimal.

Proof. Let F be an outforest in which all tasks have a modified release date and a modified deadline. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. Suppose a 0-optimal schedule on two processors exists. Let S be the schedule on two processors constructed by the above algorithm using L. Suppose not every task meets its deadline in S. Let t be the first time at which a task violating its deadline is scheduled. Assume $u \in S_t$ and $D(u) \leq t$. Let $t' - 1 < t$ be the last time at which at most one task having a deadline $\leq D(u)$ is scheduled. Define $F' = \bigcup_{i=t'}^{t-1} S_i \cup \{u\}$. $|F'| = 2(t-t')+1$ and every task in F' has a deadline $\leq D(u)$. Define $P = S_{t'-1} \cap \{v \in F \mid D(v) \leq D(u)\}$.

Case 1. $|P| = 0$. Every task in F' was rejected at time $t' - 1$. Since every task has at most one parent, all tasks of F' have a release date larger than $t' - 1$. So F contains $\geq 2(t-t')+1$ tasks that have to be executed in the interval $[t', t]$. In this interval only $2(t-t')$ tasks can be executed, so no 0-optimal schedule exists. Contradiction.

Case 2. $|P| = 1$. Suppose $u' \in P$. All tasks of F' were rejected at time $t' - 1$. So none of the tasks of F' was available at time $t' - 1$ after u' was scheduled.

Case 2.1. Every task in F' has a release date $\geq t'$ or is a successor of u'. Clearly $F' \subseteq G(u', t', D(u))$. So $|G(u', t', D(u))| \geq |F'| \geq 2(D(u) - t') + 1$. Therefore

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

Since u' is scheduled at time $t' - 1$, u' violates its deadline. Contradiction.

Case 2.2. F' contains a node that is no successor of u' or has a release date $\leq t' - 1$. Let v be such a node. We may assume that v has a release date $\leq t' - 1$. v was rejected by the algorithm at time $t' - 1$. Because $R(v) \leq t' - 1$, u' and v have a common parent u'', which is scheduled at time $t' - 2$. Clearly F' is a subset of $H(u'', t' - 2, D(u)) \backslash \{u'\}$. Therefore $|H(u'', t' - 2, D(u))| \geq |F'| + 1$. So $H(u'', t' - 2, D(u))$ contains at least $2(D(u) - t') + 2$ tasks. So

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

Because u'' is scheduled at time $t' - 2$, it violates its deadline. Contradiction.

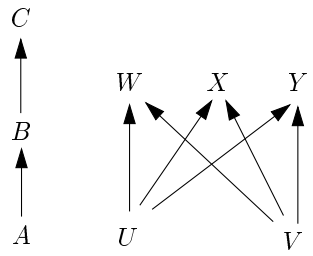
□

This algorithm does not find optimal schedules for outforests. Let F be an outforest, in which every task u has been assigned a deadline $D_0(u)$. Suppose l is the lateness of an optimal schedule for F on two processors. Let $D'_0(u) = D_0(u) + l$ for every task u. Let F' be the outforest F in which each task u has deadline $D'_0(u)$. Let $D_1(u)$ and $D'_1(u)$ be the modified deadlines of task u computed by the deadline modification algorithm for F and F', respectively. In general $D'_1(u) \neq D_1(u)$, since the deadline modification algorithm only modifies a deadline $D(u)$, if $R(u) \leq D(u)$.

It is however possible to define an algorithm that finds optimal schedules. Let F be an outforest containing n tasks. Define $l_0 = \max\{0, \max_u R(u) - D(u) + 1\}$. Any valid schedule for F has lateness at least l_0. In an optimal schedule a task u is scheduled at time $\leq R(u) + n - 1$. So the the lateness of u in an optimal schedule is $\leq R(u) + n - 1 + 1 - D(u) \leq l_0 + n - 1$. Therefore the lateness of an optimal schedule for F on two processors is at most $l_0 + n - 1$. By adding l to every deadline and applying the above algorithm to the resulting outforest a schedule with lateness at most l is constructed, if such a schedule exists. So using binary search an optimal schedule can

be found. The algorithm constructing an optimal schedule takes $O(n^3 \log n)$ time.

The above algorithm does not construct 0-optimal schedules for arbitrary graphs. This can be seen in figures 7, 8 and 9: figure 7 shows a graph and a 0-optimal schedule for this graph is shown in figure 8. Using priority list L the above algorithm finds the schedule shown in figure 9 that is not 0-optimal.



$$R(A) = 0, R(B) = 1, R(C) = 2$$

$$R(U) = 0, R(V) = 0, R(W) = 1, R(X) = 1, R(Y) = 1$$

$$D(A) = 2, D(B) = 3, D(C) = 4$$

$$D(U) = 1, D(V) = 2, D(W) = 4, D(X) = 4, D(Y) = 4$$

Figure 7: A graph in which all tasks have modified release dates and deadlines

U	V	W	X		
A	B	C	Y		

Figure 8: A 0-optimal schedule

U	A	B	C	Y	
V		W	X		

$$L = (U, V, A, B, C, W, X, Y)$$

Figure 9: The schedule constructed by the algorithm using list L

By assigning release date zero to every task in the graph shown in figure 1 a graph is constructed for which a schedule meeting every deadline exists. By joining k of these graphs the way this was done in section 2.1 a graph is constructed for which the lateness is at least k.

Let G be a graph, in which every task has a release date and a deadline. Suppose an optimal schedule for G on m processors is 0-optimal. In that case $R(u) < D(u)$ for every node u in G. So no task in the schedule constructed by the level scheduling approximation algorithm, which is presented in section 2.1, violates its release date. By using this approximation algorithm it is easy to see that the lateness of a task u in G in the schedule for G on m processors constructed by the above algorithm is at most $2D - 2 - \left\lfloor \frac{2}{m}(D-1) \right\rfloor$, where $D = \max_u D(u)$.

Again it is possible to prove a better upper bound for outforests. The outforest level scheduling algorithm presented in section 2.1 constructs schedules that might not be valid schedules, if release dates are considered. Let F be an outforest, in which every task has been assigned a deadline and a release date. Suppose a 0-optimal schedule for G on m processors exists. Let S be the schedule for G on m processors constructed by the above algorithm. By defining a level algorithm similar to

the level algorithm for outforests presented in section 2.1 it is easy to see that the lateness of a task in S having deadline d is at most $d-1$. So the lateness of S is at most $D-1$, where $D = \max_u D(u)$.

Also for inforests the above algorithm does not construct optimal schedules. The algorithm can however be used to define an algorithm for scheduling inforests on two processors. This algorithm uses the above algorithm. The algorithm for scheduling inforests first transforms the inforest into an outforest by exchanging deadlines and release dates and by reversing all edges.

Let F be an inforest. Let $D = \max_u D(u)$. We may assume $D > \max_u R(u)$. An outforest F' will be constructed. $D'(u), R'(u)$ will denote the deadlines and release date of task u in F', respectively. (u, v) is an edge of F', if and only if (v, u) is an edge of F. Let u be a task of F. We define

$$D'(u) = D - R(u) \text{ and } R'(u) = D - D(u).$$

It is easy to see that a 0-optimal schedule for F exists, if and only if a 0-optimal schedule for F' exists.

The above algorithm will be applied on F'. The schedule for F can be constructed from the schedule for F' by considering the first time slot to be the last and vice versa: if a task is scheduled at time t in the schedule for F', then in the schedule for F it is scheduled at time $D - t - 1$.

The transformation of an inforest consisting of n nodes into an outforest and the transformation of the schedule for the outforest into a schedule for the inforest both require $O(n)$ time. So this algorithm finds a schedule for an inforest in $O(n^2)$ time. It finds 0-optimal schedules on two processors, if such a schedule exists.

This algorithm does not find optimal schedules. It is however possible to find an optimal schedule using binary search. That way finding an optimal schedule takes $O(n^3 \log n)$ time.

3 Scheduling graphs with the least urgent parent property

In the previous section two algorithms were presented that construct optimal schedules on two processors for outforests. It is not surprising that these algorithms are optimal for outforests: the deadline modification algorithms only consider a task and its successors. The predecessors of a task are not taken into account. So in a graph, in which every task has been assigned a modified deadline, every task with deadline d has at most one successor with deadline $d+1$, but every task having deadline $d+1$ can have any number of predecessors having deadline d. Hence a task u has a successor that is most likely to be executed immediately after u, but it has no predecessor that is most likely to be scheduled immediately before u.

In this section an extra constraint on the deadlines is added. In a graph in which the deadlines satisfy this extra constraint, every task has a parent that is most likely to be executed immediately before this task. This parent is called the least urgent parent and the constraint is called the least urgent parent property.

Definition 3.1 (Least urgent parent property). *Let G be a graph, in which every task has been assigned a (modified) deadline. G has the least urgent parent property, if every node u in G that is not a source, has exactly one parent, such that the deadline of this parent exceeds the deadline of the other parents of u. This parent will be called the least urgent parent of u.*

In this section two algorithms for scheduling graphs with the least urgent parent property will be presented: one for scheduling series-parallel graphs on two processors and one for inforests on an arbitrary number of processors.

3.1 Scheduling series-parallel graphs on two processors

We will consider series-parallel graphs with the least urgent parent property. Chrétienne and Picouleau [3] have defined series-parallel graphs as follows. Series-parallel graphs have a single source and a single sink, and a recursive structure. The graph consisting of a single node is the smallest series-parallel graph, this single node is both its source and its sink. Suppose $G_1, \ldots, G_k$

are series-parallel graphs having sources $s_1, \ldots, s_k$ and sinks $t_1, \ldots, t_k$. There are two ways to construct a series-parallel graph from $G_1, \ldots, G_k$: the graph $SER(G_1, \ldots, G_n)$ is constructed by joining $G_1, \ldots, G_k$ and adding edges (t_i, s_{i+1}) for all i, $1 \leq i \leq k-1$. $SER(G_1, \ldots, G_k)$ is shown in figure 10. The graph $PAR(G_1, \ldots, G_k)$ is constructed by joining $G_1, \ldots, G_k$ and adding edges (s, s_i) and (t_i, t) for all i, $1 \leq i \leq k$, where s is the source of $PAR(G_1, \ldots, G_k)$ and t is its sink. $PAR(G_1, \ldots, G_k)$ is shown in figure 11.

Figure 10: $SER(G_1, \ldots, G_k)$

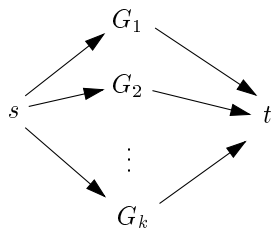


Figure 11: $PAR(G_1, \ldots, G_k)$

This definition is similar to the definitions of two terminal series-parallel graphs by Lawler [10] and edge series-parallel graphs by Valdes, et al [16]. The class of series-parallel graphs, as we will consider them, is almost a subclass of the class of two terminal series-parallel graphs: the graph consisting of a single node is the only series-parallel graph, that is not a two terminal series-parallel graph. Two terminal series-parallel graphs may contain more than one edge between two nodes, this is not allowed in the series-parallel graphs we consider. Furthermore series-parallel graphs do not have transitive edges, that is if (u, v) and (v, w) are edges of a series-parallel graph, (u, w) is not. Two terminal series-parallel graphs may contain transitive edges.

Series-parallel graphs have some nice properties. Using induction it is easy to prove a series-parallel graph satisfies the following properties.

1. Every node has at most one child or at most one parent.
2. If two nodes u and v have a common child w, then w is the only child of u and v.
3. If two nodes u and v have a common parent w, then w is the only parent of u and v.
4. If two nodes u and v have a common child w, then every child of a parent of u is a predecessor of w.
5. If two nodes u and v have a parent w in common, then every parent of a child of u is a successor of w.

In figure 12 a graph is shown for which properties 4 and 5 are not satisfied. Therefore this graph cannot be an induced subgraph of the transitive closure of a series-parallel graph.

To schedule graphs with the least urgent parent property, the algorithm constructs schedules of a special form. These schedules will be said to have the least urgent parent property.

Definition 3.2 (Least urgent parent property). *Let G be a graph with the least urgent parent property. Let S be a schedule for G. S has the least urgent parent property, if for every node u in G that is not a source, the least urgent parent of u is executed after all other parents of u.*

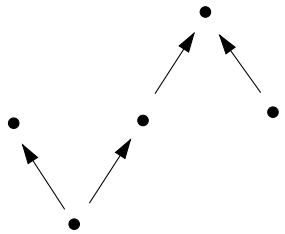


Figure 12: A forbidden subgraph for the transitive closure of a series-parallel graph

The algorithm for scheduling with deadlines presented in section 2.1 will be changed to find schedules with the least urgent parent property for graphs with the least urgent parent property. To be able to define the algorithm a different notion of availability is needed.

Definition 3.3 (Least urgent parent availability). *Let G be a graph with the least urgent parent property. Let S be a partial schedule for G on m processors. A task u is least urgent parent available at time t with respect to S, if u is available in the normal sense and if u is the least urgent parent of v, then all other parents of v are scheduled before time t.*

The modified algorithm constructs schedules with the least urgent parent property on m processors for graphs with the least urgent parent property. The deadline modification is done by the deadline modification algorithm presented in section 2.1. The algorithm to assign starting times to the tasks uses a list containing all tasks ordered by nondecreasing (modified) deadlines. In order to bound the complexity of the algorithm two variables are introduced. $NrParents(u)$ denotes the number of unscheduled parents of u and $LastParent(u)$ denotes the last time at which a parent of u is scheduled. Initially $NrParents(u)$ equals the indegree of u and $LastParent(u) = -1$ for each u.

```
LEAST URGENT PARENT LIST SCHEDULING()
 1  let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
 2  t = 0
 3  while L contains unscheduled tasks
 4     do for i = 1 to n
 5           do if u_i is unscheduled and least urgent parent available at time t
 6                 then for all children v of u_i
 7                         do NrParents(v) = NrParents(v) - 1
 8                            LastParent(v) = t
 9                      schedule u_i at time t
10        t = t + 1
```

The deadline modification requires $O(n^\alpha)$ time for arbitrary graphs. For outforests and series-parallel graphs the deadline modification takes $O(n^2)$ time: from the result by Goralčíkova and Koubek [8] mentioned the section 2.1 it follows that it takes $O(n^2)$ time to compute the transitive closure of a series-parallel graph, since such a graph has at most $2n - 2$ edges as is proved in the following lemma.

Lemma 3.4. *Let G be a series-parallel graph. The number of edges in G is at most $2|G| - 2$.*

Proof. Let G be a series-parallel graph. Let $E(G)$ denote the number of edges of G. Induction is used to prove $E(G) \leq 2|G| - 2$. The lemma obviously holds for the graph consisting of a single node. Let $G_1, \ldots, G_k$ be series-parallel graphs, such that $E(G_i) \leq 2|G_i| - 2$ for all i, $1 \leq i \leq k$.

1. $G = SER(G_1, \ldots, G_k)$.

$$\begin{aligned} E(G) &= \sum_{i=1}^{k} E(G_i) + k - 1 \\ &\leq \sum_{i=1}^{k} (2\,|G_i| - 2) + k - 1 \\ &= 2\,|G| - 2k + k - 1 \\ &= 2\,|G| - k - 1 \\ &\leq 2\,|G| - 2. \end{aligned}$$

2. $G = PAR(G_1, \ldots, G_k)$.

$$\begin{aligned} E(G) &= \sum_{i=1}^{k} E(G_i) + 2k \\ &\leq \sum_{i=1}^{k} (2\,|G_i| - 2) + 2k \\ &= 2(|G| - 2) - 2k + 2k \\ &= 2\,|G| - 4 \\ &\leq 2\,|G| - 2. \end{aligned}$$

□

For each time slot the priority list L is traversed to find least urgent parent available tasks. It is easy to see that a task u is least urgent parent available at time t, if it is available and $NrParents(v) = 1$ and $LastParent(v) < t$ for each child v of u such that u is the least urgent parent of v. Every task, that is not a source, has exactly one least urgent parent, so there are only $O(n)$ edges (u, v) such that u is the least urgent parent of v. For every node u two lists of children are kept: one list contains all children for which u is the least urgent parent, the other children of u are contained in the other list. So to check whether an available node u is least urgent parent available only the children in the first list have to be checked. Suppose u has k_u children for which it is the least urgent parent. $O(k_u)$ time is required to check whether u is least urgent parent available.

Let G be a graph. Since every task has release date zero, the length of a schedule for G constructed by the above algorithm is at most n. Therefore $O(nk_u)$ time is needed to check whether a node u is least urgent parent available throughout the execution of the algorithm. Hence $O(n^2)$ time is required to scan the priority list throughout the algorithm, because $\sum_{u \in G} k_u \leq n$.

Creating the priority list takes $O(n)$ time, as does the initial assignment of $LastParent(u)$ for every node u. The initial assignment of the values $NrParents(u)$ takes $O(indegree(u))$ time for each node u, so $O(e)$ time in total. Constant time is needed to update $NrParents(v)$ and $LastParent(v)$ after a starting time has been assigned to a node u for every child v of u. So after a node u is scheduled updating $NrParents$ and $LastParent$ for its children requires $O(outdegree(u))$ time. So $O(e)$ time is required to update $NrParents$ and $LastParent$. So $O(n^2)$ time is used by the above algorithm to schedule a graph.

So the least urgent parent algorithm uses $O(n^\alpha)$ time to schedule an arbitrary graph. We already know that computing the transitive closure of forests and series-parallel graphs requires only $O(n^2)$ time. So the least urgent parent scheduling algorithm uses $O(n^2)$ time to schedule a forest or a series-parallel graph.

Now it is proved that the above algorithm finds a 0-optimal schedule on two processors for a series-parallel graph with the least urgent parent property, if a 0-optimal schedule exists.

Theorem 3.5. *Let G be a series-parallel graph with the least urgent parent property. Let L be a list containing all tasks of G ordered by nondecreasing deadlines. If a 0-optimal schedule for G on two processors exists, the schedule constructed by the least urgent parent scheduling algorithm using L is 0-optimal.*

Proof. Let G be a series-parallel graph with the least urgent parent property. Let L be a list containing all tasks of G ordered by nondecreasing deadlines. Suppose a 0-optimal schedule for G on two processors exists. Let S be the schedule for G on two processors constructed by the least urgent parent scheduling algorithm using L. Suppose S is not 0-optimal. Let t be the first

time at which a task is scheduled that violates its deadline. Assume $u \in S_t$ and $D(u) \leq t$. Let $t'-1 < t$ be the last time at which at most one task having a deadline $\leq D(u)$ is scheduled. Define $G' = \bigcup_{i=t'}^{t-1} S_i \cup \{u\}$. Define $P = S_{t'-1} \cap \{v \in G \mid D(v) \leq D(u)\}$.

Case 1. $|P| = 0$. All tasks of G' were rejected at time $t'-1$. The only tasks in G' that are available, are least urgent parents of nodes for which at least two parents are not scheduled at time $< t'-1$. Let v be a task in G' scheduled at time t'. Two parents of v are scheduled at time $t'-2$, otherwise v would be scheduled at time $t'-1$. S has the least urgent parent property, so the least urgent parent of v is scheduled at time $t'-1$. This is impossible, since $P = \emptyset$.

Case 2. $|P| = 1$. Suppose $u' \in P$. If every task of G' would be a successor of u', t and t' would be equal, since only one child of u' can be scheduled immediately after u'. In that case u' has one successor, u. This implies $D(u') \leq D(u) - 1 \leq t - 1 = t' - 1$. So u' would not meet its deadline. So G' contains a task that is not a successor of u'.

Case 2.1. $S_{t'-2}$ contains a parent u'' of u' and every task of G' is a successor of u''. u'' has at least $2(t-t')+2$ successors with a deadline $\leq D(u)$. So

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

Since $u'' \in S_{t'-2}$, it violates its deadline. Contradiction.

From now on we assume $t \neq t'$. If $t = t'$ and u is neither a child of u' nor a child of a parent of u' scheduled at time $t'-2$, u is the least urgent parent of a child of u'. In that case $D(u') < D(u)$, so u' would violate its deadline.

Case 2.2. $S_{t'-2}$ does not contain a parent of u'. $S_{t'}$ contains two tasks. Because of communication delays one of these tasks is not a successor of u'. Let $v \in S_{t'}$ be a task that is not a successor of u'. v was rejected at time $t'-1$. So v is not least urgent parent available with respect to $S_{t'-1}$.

Case 2.2.1. v is not available at time $t'-1$. Because v is no successor of u', $S_{t'-2}$ contains two parents of v. Since S has the least urgent parent property, the least urgent parent of v is scheduled at time $t'-1$. So u' is the least urgent parent of v. Contradiction.

Case 2.2.2. v is available at time $t'-1$, but v is not least urgent parent available. So v is the least urgent parent of a node w. All other parents of w have a deadline $< D(v) \leq D(u)$. v is scheduled at time t', so u' is a parent of w as well. Since G is a series-parallel graph, w is the only child of u' and v. Let $w' \in S_{t'}$, such that $w' \neq v$. w' is not a successor of u' or v, since G is a series-parallel graph. w' was rejected at time $t'-1$. $S_{t'-2}$ cannot contain two parents of w', otherwise u' would be the least urgent parent of w'. So w' is a least urgent parent of some node. Since w' is scheduled at time t' and G is a series-parallel graph, w' is the least urgent parent of the child of u'. So $v = w'$. Contradiction.

Case 2.3. $S_{t'-2}$ contains a parent u'' of u' and G' contains a node v such that $u'' \not\prec v$. Assume v has no predecessors in G'. v was rejected at time $t'-1$. So v is not least urgent parent available with respect to $S_{t'-1}$.

Case 2.3.1. v is not available at time $t'-1$. v is no successor of u'', so u' is no predecessor of v. Therefore $S_{t'-2}$ contains two parents of v. But in that case u' is the least urgent parent of v. Contradiction.

Case 2.3.2. v is available at time $t'-1$. v was rejected at time $t'-1$. Therefore v is not least urgent parent available. So v is the least urgent parent of a node w. Assume v is scheduled at time t_0 such that S_{t_0} is the first time slot after $S_{t'-1}$ containing a task that is not a successor of u''.

Case 2.3.2.1. $t_0 = t'$. u' is a parent of w, otherwise v would be least urgent parent available at time $t' - 1$. Let $w' \neq v$ be scheduled at time t'. Since G is a series-parallel graph, w' is not a successor of u' or v. $S_{t'-2}$ cannot contain two parents of w', otherwise u' would be the least urgent parent of w'. So w' is the least urgent parent of another node. Since w' is scheduled at time t' and G is a series-parallel graph, w' is the least urgent parent of the child of u'. So $v = w'$. Contradiction.

Case 2.3.2.2. $t_0 \neq t'$. Before time t_0 another parent w' of w is scheduled, otherwise v would be scheduled at time $t' - 1$. w' is a successor of u''. u'' has at least two children, since one of the tasks scheduled at time t' cannot be a child of u'. Let x, y be two children of u'', such that $x = w'$ or $x \prec w'$. Since v is not a successor of u'', the subgraph induced by $\{u'', v, x, y, w\}$ of the transitive closure of G is isomorphic to the graph shown in figure 12. Contradiction.

□

From the preceding theorem the existence of schedules with the least urgent parent property clearly follows.

Corollary 3.6. *Let G be a series-parallel graph with the least urgent parent property. If a 0-optimal schedule for G on two processors exists, a 0-optimal schedule with the least urgent parent property exists.*

Proof. Obvious from theorem 3.5. □

The least urgent parent availability of a task does not depend on its deadline. So the above algorithm finds optimal schedules.

Corollary 3.7. *Let G be a series-parallel graph with the least urgent parent property. The schedule on two processors for G constructed by the above algorithm is optimal.*

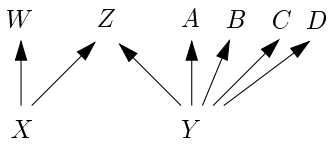
Proof. Obvious. □

Only properties 2, 4, and 5 of series-parallel graphs are used in the proof of theorem 3.5. Hence theorem 3.5 and corollaries 3.6 and 3.7 hold for a larger class of graphs: the class of graphs that satisfy properties 2, 4, and 5 and have the least urgent parent property.

For example theorem 3.5 and corollaries 3.6 and 3.7 also hold, if G is an inforest or an opposing forest (a graph consisting of an inforest and an outforest) with the least urgent parent property.

The above algorithm does not find an optimal schedule for every graph with the least urgent parent property. Figure 13 shows a graph with the least urgent parent property, for which a 0-optimal schedule on two processors exists. Such a schedule is shown in figure 14. It is easy to see that this schedule is the only 0-optimal schedule for this graph, that is every task must be scheduled at the time it is scheduled in this schedule in order to meet every deadline. The schedule shown in figure 14 does not have the least urgent parent property.

Using the outforest level scheduling algorithm presented in section 2.1 an upper bound on the maximum lateness of a schedule constructed by the above algorithm can be proved. Let G be a graph with the least urgent parent property. Every task in G with deadline $d + 1$ has at most parent with deadline d. Therefore the outforest level scheduling algorithm finds a valid schedule for G. Let S be the schedule for G on m processors constructed by the above algorithm. It is not difficult to see that the lateness of task u in S is at most its lateness in the schedule constructed by the approximation algorithm. So if an optimal schedule for G has lateness l and u has deadline d, then the lateness of u is at most $d + l - 1 - \lfloor \frac{1}{m}(d-1) \rfloor$.



$$D(A) = 2, D(B) = 3, D(C) = 3, D(D) = 4$$
$$D(W) = 4, D(X) = 2, D(Y) = 1, D(Z) = 4$$

Figure 13: A graph with the least urgent parent property

Y	A	B	D	
X	W	C	Z	

Figure 14: The only 0-optimal schedule

3.2 Scheduling inforests on m processors

In the section 3.1 it was mentioned that the algorithm constructing 0-optimal schedules for series-parallel graphs also finds optimal schedules for inforests with the least urgent parent property. Now an algorithm will be presented for scheduling inforests with the least urgent parent property on an arbitrary number of processors.

It will be proved that if a 0-optimal schedule exists, a 0-optimal schedule with the least urgent parent exists as well. The existence of 0-optimal schedules with the least urgent parent property will be shown using an algorithm constructing such schedules.

Like all algorithms presented earlier this algorithm first modifies every deadline and assigns starting times using the modified deadlines. The deadline modification for inforests is less complicated than for arbitrary graphs. It is not necessary to compute the transitive closure of an inforest: to modify the deadline of a node only the deadline of its child has to be considered.

While some nodes do not have a modified deadline, select of node u not having a modified deadline, such that its child v has an modified deadline. Then

$$D(u) = \min\{D(u), D(v) - 1\}.$$

Clearly the modified deadlines are consistent with the precedence constraints. It is not difficult to see that the deadlines calculated by the above algorithm equal the deadlines the general deadline modification algorithm would compute.

The assignment of starting times is done by the algorithm presented in section 3.1 without the variables that were introduced to bound the complexity.

```
LEAST URGENT PARENT LIST SCHEDULING()
1  Let L = (u_1, ..., u_n) contain all tasks ordered by nondecreasing deadlines
2  t = 0
3  while L contains unscheduled tasks
4      do for i = 1 to n
5             do if u_i is unscheduled and least urgent parent available at time t
6                    then schedule u_i at time t
7         t = t + 1
```

Now it will be shown that this algorithm constructs 0-optimal schedules, if such schedules exist.

Theorem 3.8. *Let F be an inforest with the least urgent parent property. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. If a 0-optimal schedule on m processors exists, the schedule constructed by the above algorithm using L is 0-optimal.*

Proof. Let F be an inforest with the least urgent parent property. Let L be a list containing all tasks of F ordered by nondecreasing deadlines. Suppose a 0-optimal schedule for F on m processors exists. Let S be the schedule constructed by the above algorithm using L. Suppose not every task meets its deadline in S. Let S_t be the earliest time slot containing a task violating its deadline. Assume $u \in S_t$ and $D(u) \leq t$. Let $t' - 1 < t$ be the last time such that $S_{t'-1}$ contains less than m tasks having a deadline $\leq D(u)$. Define $F' = \bigcup_{i=t'}^{t-1} S_i \cup \{u\}$ and $P = S_{t'-1} \cap \{v \in F \mid D(v) \leq D(u)\}$.

Case 1. $P = \emptyset$. Every task of F' is rejected at time $t' - 1$. So every task in F' available at time $t' - 1$ is the least urgent parent of a node, for which at least two parents are not scheduled at time $< t' - 1$. If an available task of F' is not least urgent parent available at time $t' - 1$, then this task is not least urgent parent available at time t'. So $S_{t'}$ contains a task for which two parents are scheduled at time $t' - 2$. Let v be such a task. In that case $S_{t'-1}$ must contain the least urgent parent of v. Contradiction.

Case 2. $P \neq \emptyset$.

Case 2.1. $t = t'$. u is not a child of P, otherwise a task in P would violate its deadline. Since S has the least urgent parent property, $S_{t'-2}$ does not contain two parents of u. So u is available at time $t' - 1$. u is not scheduled at time $t' - 1$, so u is the least urgent parent of some node v such that another parent of v is scheduled at time $\geq t' - 1$. u is scheduled at time $t = t'$, so $S_{t'-1}$ contains another parent w of v. u is the least urgent parent of v, so $D(w) < D(u) \leq t$. So $D(w) \leq t - 1 = t' - 1$. So w violates its deadline. Contradiction.

Case 2.2. $t \neq t'$. Not every task in F' is a successor of P. Define $V_0 = \{w \in S_{t'} \mid w \text{ is a child of } P\}$ and $V_1 = \{w \in S_{t'} \backslash V_0 \mid w \text{ is the least urgent parent of some node } w' \text{ and } P \text{ contains another parent of } w'\}$. Let $V = V_0 \cup V_1$. Since a task has at most one child, $|V| \leq |P| \leq m - 1$. Therefore $S_{t'} \backslash V$ is not empty. Let v be a task in $S_{t'} \backslash V$. v was rejected at time $t' - 1$. v is not the least urgent parent of a task w' such that another parent of w' is scheduled at time $\geq t' - 1$. So v is not available at time $t' - 1$. So $S_{t'-2}$ contains two parents of v. In that case $S_{t'-1}$ contains the least urgent parent of v. Contradiction.

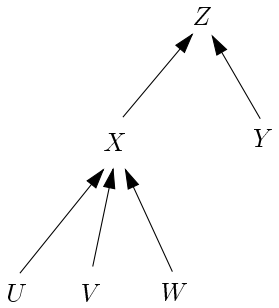
□

From the preceding theorem the existence of schedules with the least urgent parent property clearly follows for every inforest with the least urgent parent property for which a schedule meeting each deadline exists.

Corollary 3.9. *Let F be an inforest with the least urgent parent property. If a 0-optimal schedule for F on m processors exists, a 0-optimal schedule with the least urgent parent property on m exists.*

Proof. Obvious from theorem 3.8. □

This result will be used to define a more efficient algorithm to schedule inforests with the least urgent parent property. Let F be an inforest with the least urgent parent property. F will be transformed into an inforest to be scheduled without communication delays. This transformation is similar to the transformation for outforests defined by Lawler [11]. The transformed inforest will be called a delay-free inforest. For each node u in F having parents $v_1, \ldots, v_k$ such that v_1 is the least urgent parent of u, the delay-free inforest contains the edges (v_1, u) and (v_i, v_1) for all i, $2 \leq i \leq k$. The deadlines remain unchanged. Figure 15 contains an inforest with the least urgent parent property and figure 16 shows the corresponding delay-free inforest.



$$D(U) = 1, D(V) = 1, D(W) = 2$$
$$D(X) = 3, D(Y) = 4, D(Z) = 5$$

Figure 15: An inforest with the least urgent parent property

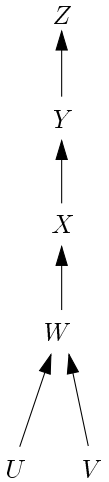


Figure 16: The corresponding delay-free inforest

Clearly a valid schedule for the delay-free inforest F' is a valid schedule for F. Such a schedule has the least urgent parent property. Each valid schedule for F with the least urgent parent property is a valid schedule for F'. So if a 0-optimal schedule for F exists, a 0-optimal schedule for F' exists.

Brucker, et al [2] have presented an algorithm for scheduling delay-free inforests. This algorithm finds 0-optimal schedules, if such schedules exist. This algorithm is described below.

First every deadline is modified. This is done the way it was described for inforests with communication delays. The tasks are put in a list ordered by nondecreasing deadlines. A list scheduling algorithm assigns a starting time to each task. $LastParent(u)$ denotes the last time slot containing a parent of u. Initially $LastParent(u) = -1$ for every node u. $N(t)$ equals the number of tasks scheduled at time t and $First$ is the earliest time slot that is not completely filled, that is $First$ equals the smallest t, such that $N(t) \leq m - 1$. Initially $N(t) = 0$ for each t and $First = 0$. The assignment of starting times is done by the following algorithm.

DELAY-FREE LIST SCHEDULING()
1 let $L = (u_1, \ldots, u_n)$ contain all tasks ordered by nondecreasing deadlines
2 **for** $i = 1$ **to** n
3 **do** $t = \max\{LastParent(u_i) + 1, First\}$

```
4       schedule u_i at time t
5       N(t) = N(t) + 1
6       if N(t) = m
7          then First = First + 1
8       let v be the child of u_i
9       LastParent(v) = max{t, LastParent(v)}
```

Clearly both the deadline modification and the assignment of starting times require linear time. For an inforest with the least urgent parent property (with communication delays) linear time is needed to find the corresponding delay-free inforest.

So the above algorithm algorithm finds schedules in linear time. These schedules are 0-optimal, if a 0-optimal schedule exists.

Also because deadlines do not influence the least urgent parent availability, the above algorithm finds optimal schedules.

Corollary 3.10. *Let F be an inforest with the least urgent parent property. The schedule for F constructed by the above algorithm is optimal.*

Proof. Obvious. □

Concluding remarks

In this report two algorithms were presented that construct optimal schedules for outforests. These algorithms can be used to present an algorithm that finds 0-optimal schedules of minimum length.

Let F be an outforest in which every task has been assigned a release date and a modified deadline. Suppose a 0-optimal schedule for F on two processors exists. Let $R = \max_u R(u)$ and $D = \max_u D(u)$. The length of a shortest 0-optimal schedule for G is at least $R + 1$ and at most $D \leq R + n$. By assigning deadline $D'(u) = \min\{D(u), l\}$ to every task u in G and applying the algorithm presented in subsection 2.2 to G a 0-optimal schedule of length at most l is constructed, if such a schedule exists. By using binary search a 0-optimal schedule of minimum length is found. So after applying this algorithm $O(\log n)$ times a 0-optimal schedule of minimum length is found. So $O(n^3 \log n)$ time is used to find a 0-optimal schedule of minimum length. If uniform release dates are considered only $O(n^2 \log n)$ time is required.

Note that this construction cannot be used for the algorithm that constructs 0-optimal schedules having the least urgent parent property of minimum length, since the least urgent parent property of a graph is violated by this construction.

For several of the graphs we needed an extra constraint on the deadlines to present an algorithm that finds optimal schedules. This is caused by the deadline modification algorithm. This algorithm uses the information that for each node u at most one child of u can be scheduled immediately after u. The knowledge that at most one parent of u can be executed immediately before u is not used. The least urgent parent property is based on this information: the least urgent parent of a task is executed immediately before this task.

In order to present algorithms finding optimal schedules for inforests and series-parallel graphs or other classes of graphs with arbitrary deadlines the extra knowledge has to be used either in the deadline modification algorithm or during the assignment of starting times.

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] P. Brucker, M.R. Garey, and D.S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics of Operations Research*, 2(3):275–284, August 1977.

[3] P. Chrétienne and C. Picouleau. Scheduling with communication delays: a survey. In *Proc. Summer School on Scheduling Theory and its Applications*, Bonas, France, 1992. To appear.

[4] E.G. Coffman, Jr. and R.L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213, 1972.

[5] 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.

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

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

[10] E.L. Lawler. Sequencing problems with series-parallel precedence constraints. Unpublished manuscript, 1978.

[11] E.L. Lawler. Scheduling trees on multiprocessors with unit communication delays. Unpublished manuscript, 1993.

[12] J.K. Lenstra, M. Veldhorst, and B. Veltman. The complexity of scheduling trees with communication delays. In T. Lengauer, editor, *Proc. 1st European Symposium on Algorithms, ESA '93*, volume 726 of *Lecture Notes in Computer Science*, pages 284–294, Berlin, 1993. Springer-Verlag. To appear in Journal of Algorithms.

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

[14] C. Picouleau. New complexity results on the UET-UCT scheduling problem. In *Proc. Summer School on Scheduling Theory and its Applications*, pages 487–502, Bonas, France, 1992. To appear in Discrete Applied Mathematics.

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

[16] J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, May 1982.