Scheduling Trees with Communication Delays

Andreas Jakoby
Rüdiger Reischuk
Med. Universität zu Lübeck*

October 1997

Abstract

Several variants of scheduling task graphs with interprocessor communication delays have been shown to be $\mathcal{NP}$-complete. This paper completely characterizes the complexity of this problem under the restriction that the task graphs are trees and that each task has unit execution time. It is shown that the problem remains $\mathcal{NP}$-complete for binary trees with uniform communication delays. The same holds for complete binary trees, but varying communication delays. On the other hand, a polynomial time algorithm is presented and analyzed that constructs an optimal schedule for complete k-ary trees with uniform communication delays. Finally, the parallel complexity of scheduling trees and dags with constant communication delay is estimated by proving inclusion in NC, resp. P-completeness results.

1. Introduction

A directed acyclic graph can be used to represent the dependencies among elementary computation steps in a given algorithm. We will call such a graph a **task graph**. Let us assume that each node of the task graph has the same execution time, which will be taken as the unit. If one wants to implement an algorithm as fast as possible on a synchronous multiprocessor architecture one simply has to schedule the tasks on the available set of processors.

In general, there will be a delay due to the necessary communication between processors. Typically, a single communication step exceeds the time of an elementary computation step by far. If processor P_a has computed an intermediate result z in the t-th step and this is needed by some other processor P_b as input for its computation one cannot assume that z is available for P_b already at step $t+1$. This leads to the following scheduling problem.

Definition 1 A **task graph** $(G; \text{ffi})$ is a DAG $G = (V; A)$, which represents a set of **tasks V** and their dependencies as a set of directed edges A, together with a delay function $\text{ffi} : A \, \Gamma \, \mathbb{N}$. Let $P = \text{f}P_1; P_2; \colon\colon\colon \text{g}$ be the set of processors of the parallel machine, which is going to execute G. A **schedule S** for $(G; \text{ffi})$ and P is a set of triples $S \, \Gamma \, V \, \Gamma \, P \, \Gamma \, \mathbb{N}$ such that the following conditions are fulfilled:

1. $8x \; 2 \; V \quad 9P_i \; 2 \; P$ and $t \; 2 \; \mathbb{N} \; : \; (x; P_i; t) \; 2 \; S$;
   for each task $x \; 2 \; V$ there is at least one processor P_i and a time t such that P_i processes x at t.

[Γ] Institut für Theoretische Informatik, WallstraΓe 40, D-23560 Lübeck, Germany
email: jakobyΓreischuk Γinformatik.mu-luebeck.de

2. $(x;P_i;t)\ 2\ S \quad =) \quad 8\ y\ 6{=}\ x \quad (y;P_i;t)\ 62\ S$;
   each processor does not process several tasks at the same time.

3. $(x;P_i;t)\ 2\ S \quad \Gamma \quad (y;x)\ 2\ A \qquad =)$
   $9\ t^0 \Gamma\ t$ such that $(y;P_i;t^0)\ 2\ S$ or
   $9\ P_j\ 2\ P$ and $t^{00} \Gamma\ t\ \Gamma\ \text{ffi}(x;y)$ such that $(y;P_j;t^{00})\ 2\ S$:

   If x is executed at time t by processor P_i and y is a direct predecessor of x then either
   (a) y is processed by P_i by time $t\,\Gamma\,1$ or
   (b) y is processed by some other processor P_j by time $t\,\Gamma\,1\,\Gamma\,\text{ffi}(y;x)$.

Note that a schedule may start at time 0 . If the last step is executed at time t then the schedule is finished by time $t+1$. Let S_i denote the **schedule S** restricted to processor P_i :

$$S_i \ := \ f(x;t)\ j\ (x;P_i;t)\ 2\ Sg$$

and $S_i(t)$ the task that is executed by P_i at step t . The finishing time of P_i is given by

$$T(S_i) \ := \ 1 + \text{maxft}\ j\ S_i(t) \text{ is defined } g$$

and the **time span** or **length** of S by

$$T(S) \ := \ \max T(S_i) \ :$$

A schedule S is called **optimal** for $(G;\text{ffi})$ and P , if $T(S)$ is minimal for all schedules S^0 for $(G;\text{ffi})$ and P . Let T_{opt} be the optimal schedule length:

$$T_{opt}((G;\text{ffi});P) \ := \ \text{minf}T(S)jS \text{ is a schedule for } (G;\text{ffi}) \text{ and } P g \ :$$

In the following the most interesting case of maximal parallelism is considered, that means the set of processors P can be chosen arbitrarily large. We therefore simply denote the minimal scheduling time by $T_{opt}(G;\text{ffi})$. Depending on the type of communication delays we will distinguish between the following variants of this scheduling problem.

DELAY SCHEDULING (DS):
Given a task graph $(G;\text{ffi})$ and a deadline $T^\Gamma\ 2\ \mathbb{N}$ decide:
Is there a schedule S with $T(S)\ \Gamma\ T^\Gamma$, that means $T_{opt}(G;\text{ffi})\ \Gamma\ T^\Gamma$?

UNIFORM DELAY SCHEDULING (UDS)[1]:
DS restricted to inputs $(G;\text{ffi})$ with $\text{ffi}\ \Gamma\ \Gamma$ for some $\Gamma\ 2\ \mathbb{N}$ that may depend on G and may increase with the number of nodes of the task graphs.

CONSTANT DELAY SCHEDULING (CDS):
DS restricted to inputs $(G;\text{ffi})$ with $\text{ffi}\ \Gamma\ c$ for a universal constant c independent of the graph G .
ff

The UDS problem is defined by Papadimitriou and Yannakakis in [PY88]. In their paper it is shown

Theorem: UDS is NP -complete.

They also describe a simple algorithm that approximates the optimal schedule length within a factor of 2. The accuracy of the approximation follows from a simple lower bound that can be derived for the minimal schedule length. An extension of the algorithm can be given that also solves the more general DS-problem within the same factor.

$\infty\!\!\Omega$

In [JKS89] Jung, Kirousis and Spirakis describe a strategy based on dynamic programming for computing a schedule of minimal length. The runtime grows polynomial in the size of the DAGs, but exponential in the communication delay. The authors claim a time bound $O(n^{\Gamma+1})$, but there seems to be a gap in the analysis. However a time bound $O(n^{\Gamma+2})$ is achievable, where n is the number of nodes of the task graph $(G; \text{ffi})$ and $\text{ffi}\ \Gamma\ \Gamma$ the uniform communication delay. This result implies that the CDS problem can be solved in deterministic polynomial time. Furthermore, the communication delay necessarily has to grow substantially with the number of nodes for any NP-reduction to UDS (unless $P = NP$).

Jung et al. also improve the lower bound of [PY88] for the minimal schedule length for graph that are complete binary trees, namely from about $\log n \Delta \Gamma = \log \Gamma$ to about $2 \log n \Delta \Gamma = \log \Gamma$. They conclude that this matches the upper bound given in [PY88] at the constant level. Does this mean that the algorithm of [PY88] constructs optimal schedules when restricted to complete binary trees? The answer must be no. In general, an optimal schedule is shorter and does not have the simple symmetric structure common to all schedules constructed by this algorithm. Since the lower and the upper bound as stated are a little floppy – they treat Γ and $\Gamma + 1$ as equal and do not differentiate between rounding up or rounding down $\log \Gamma$ – the bound $2 \log n\, \Delta\, \Gamma = \log \Gamma$ does not even give the correct constant factor, by which $\log n$ has to be multiplied. For example, for $\Gamma = 11$ the bound claims a factor $6{:}359 :::$, but the correct factor in the asymptotic limit is 5, which will follow from our results below.

Let us finally mention that the DS problem can also be solved in polynomial time for any delay function ffi that takes arbitrary values in the real interval $[0; 1]$ as shown by Chrétienne and Picouleau [CP91]. However, the assumption that all occurring delays are smaller than an elementary processing step is a severe and for most applications unrealistic restriction.

2. New Results

The delay scheduling problem is of great practical importance. Therefore it seems useful to fully understand the influence of the different parameters like type of task graphs and type of delays. In particular, which properties give rise to NP-completeness? For example as already mentioned, an efficient approximation of an optimal schedule within a factor 2 is possible, but we do not know of any polynomial time scheme that for arbitrary graphs guarantees a factor $\text{ff}\ \Gamma\ 2$.

We continue this line of research for the delay scheduling problem and concentrate on the underlying task graphs. It has been shown for many NP-complete optimization on graphs that very fast solutions are possible if one restricts to special types of graphs like series-parallel or bounded tree-width graphs (for an overview see [R91]). In particular, almost every problem becomes simple on trees. The well studied unit-execution time multiprocessor scheduling problem without delay, i.e. $\text{ffi}\ \Gamma\ 0$, is known to be NP-complete for arbitrary graphs, but can be solved in polynomial time on forests [GJ79]. For this version the number of available processors is a parameter of the problem instance and hence bounded. Otherwise, the problem becomes trivial with minimal schedule length equal to the depth of the DAG.

If one extends this scheduling problem such that each task has its individual deadline the situation is as follows. NP-complete even for DAGs that are trees, but in P for reversed trees [GJ79]. In the following, tree always means an intree, that is all edges run in direction from the leaves to the root of the tree. In a reversed or outtree the orientation is switched. In this paper we want to estimate the complexity of the delay scheduling problem when restricted to trees of bounded degree. The NP-completeness reduction of UDS given in [PY88] generates DAGs of unbounded indegree, more precisely, the degree may become close to the size of the graph. Our results improve this situation considerably. (1,2)-trees and complete binary trees are the simplest types of such trees. The main result of this paper is that the delay scheduling problem even restricted to such simple graphs does not become easier.

Theorem 1 ––

The remaining part of this paper is organized as follows. Section 3 contains the NP-completeness proof for $(1;2)$-trees, section 4 the proof for complete binary trees with varying delays. In section 5 we will present an algorithm for complete trees with uniform delays and prove its correctness. Then a recurrence for the optimal schedule length and some numerical values for binary, ternary and 4-ary trees are given. We close with an overview on the known complexity results for delay scheduling. An extended abstract of these results has been presented at 3. SWAT 1992 [JR92], some of these results can also be found in [J91].

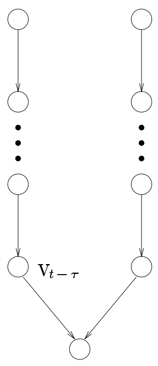
3. The $(\Sigma$-Completeness Proofs for AA, AA-Trees

Let us first observe that the DS problem for a graph $G = (V; A)$ with delay ffi and deadline T^{Γ} can be solved by a polynomial time bounded NTM as follows:

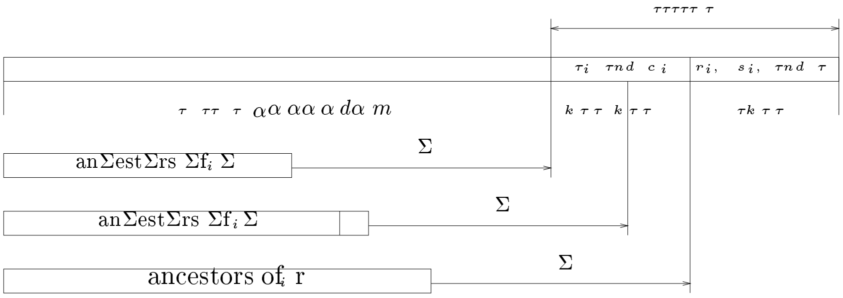
1. Guess a set $S \, \Gamma \, V \, \Gamma \, P \, \Gamma \, \mathbb{N}$ of triples of size at most jVj^2 where P is a set of jVj processors.
2. Verify that S is a schedule for G and ffi.
3. Verify that $T(S)$ is not larger then T^{Γ}.

The correctness follows from the fact that for a task graph of size n there are always optimal schedules that use at most n processors with the total number of processor steps being bounded by n^2.

Lemma 1 ––– ––


$v_{t-\tau}$

Lemma 2 $T_{opt}(\Delta_t; \Gamma) = t + 1$ –


ττττττ τ
τi τnd c i
ri, si, τnd τ
τ ττ τ αα αα α dα m
k τ τ k τ τ
τk τ τ
anΣestΣrs Σfi Σ
Σ
anΣestΣrs Σfi Σ
Σ
ancestors of i r
Σ

This contradicts property a1.

a3 If a processor P^0 executes the first part of a u_j-chain $(u_j; : : : ; u_{k+1})$ and P_W executes the second part of the u_j-chain $(u_{k\Gamma 1}; : : : ; u_1)$ P_W cannot execute x_1 of this subtree before

$$\begin{aligned} ff_j + (j \, \Gamma \, k) + \Gamma + k + d \, \Gamma \, 1 \quad & \Gamma \; ff_n + n + d + \Gamma \, \Gamma \, 1 \\ & = ff_n + n + d + (m \, \Gamma \, 1)(d+3) + m + k \, \Gamma \, 1 \\ & = ff_n + k + 4m + m(d+3) \, \Gamma \, 4 \\ & \Gamma \; ff_0 + m(d+3) + 2m \\ & = fl + 2m: \end{aligned}$$

This contradicts property a1.

a4 If a processor P^0 executes a x_i-chain P_W cannot execute c_i before $ff_0 + d + 3 + \Gamma = fl + m + k$.

a5 $\mathbf{Pfi}$ cannot start the execution of F or a node r_i before $fi = fl + 2k$ (follows from the construction of Δ_{fi}).

a6 $\mathbf{Pfi}$ executes at most $k \, \Gamma \, m$ c_i between $fl + m + k$ and fi. So it has to execute at least m x_i-chains (follows from property a4).

a7 If the schedule obeys the deadline T^{Γ} P_W has to use the subschedule of figure 7 (follows from property a1, a4, and a5).

a8 $\mathbf{Pfi}$ cannot start the execution of a x_i-chain before ff_0. So P_W executes at most m x_i-chains (follows from property a1).

a9 $\mathbf{Pfi}$ executes exactly m x_i-chains (follows from property a6 and a8) and $3m$ u_j-chains (follows from the construction of D_i-trees and property a3). ∎

Proof of Fact B: If there exists m D_i-subtrees with $3m$ U_j-subtrees which represent different elements of M P_W executes the u_j-chains $j := 3m : : : 1$ before ff_0 (like in figure 9) and the m x_i-chains before fl.

$$ff_0 = \Gamma + \Gamma + \sum^{1}$$

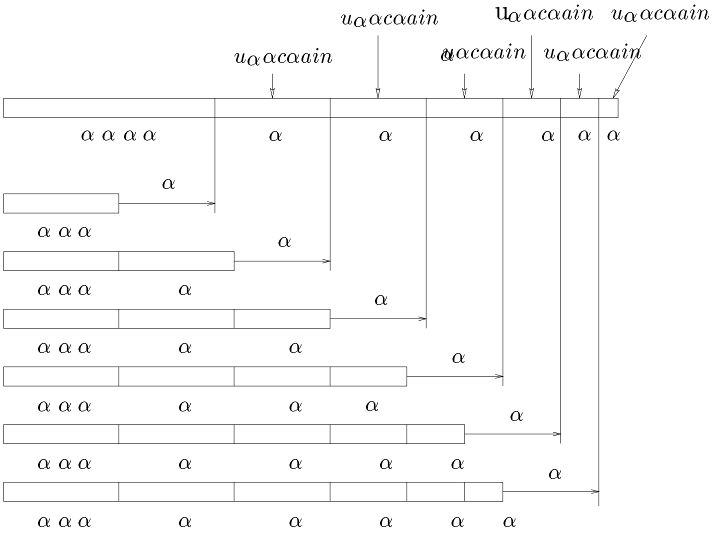
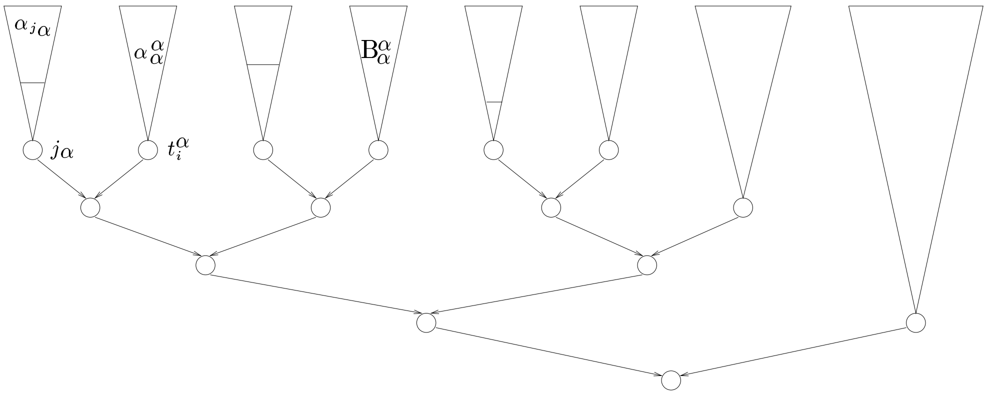


Figure 9: This diagram shows a small example with $3m = 6$ that describes a schedule for 6 different u_j -chains.

4. The Reduction for Complete Binary Trees

The **proof** for complete binary trees goes along the same lines as the one for $(1;2)$ -trees. Technically, it is more involved in order to construct a tree of depth $O(\log n)$, where $n = jMj$ and M and K are the inputs of the X3C problem.

The construction is illustrated by figure 10. Each subset $K_i = fj_1;j_2;j_3g\ 2\ K$ is represented by a subtree D_i . The subtree with root j_1 consisting of the pieces E_{j_α} and C_{j_α} represents the element j_1 .



D_i can be constructed by the following strategy:

1. Build a tree D_i^0 with subgraphs $E_{j\alpha}^0, C_{j\alpha}^0, E_{j\alpha}^0, C_{j\alpha}^0$, $E_{j\alpha}^0, C_{j\alpha}^0, B_2^{10}, B_2^{20}, B_2^{30}, B_1^0$ and B_0^0 where all leafs of D_i^0 have the same depth. The subtrees of D_i^0 have the following number of nodes

$$N(E_{j_i}^0) := ji \qquad\qquad N(B_2^{10}) := \sum$$

Proof: Let P_w be the processor that executes the root w of B.

If a node w_i is executed by a processor P^0 ($P^0 6= P_w$) P_w has to wait at least $N(B^0_0) + 1 + N(B) \Gamma$ $N(B) \Gamma T^{\Gamma}$ steps before P_w is able to execute a successor of w_i. Thus, P_w has to execute all w_i and all there successors. ∎

Lemma 7 ––

b5 **P_{fi}** cannot start the execution of F or a node r_i before $N(B^0_3) = N(B^0_0) + k$ (follows from the construction of B).

b6 **P_{fi}** executes at most $k \Gamma m$ w_i between $N(B^0_0) + m$ and $N(B^0_3)$. So it has to execute at least m u^6_i (follows from property b4).

b7 **P_{fi}** cannot start the execution of a u^5_i before $N(B^0_1) = N(B^0_0) \Gamma 2m$. So P_W executes at most m u^6_i's (and m u^5_i's) (follows from property b1).

b8 **P_{fi}** executes exactly m u^6_i's (follows from property b6 and b7) and 3m j 's (follows from the construction of D_i -trees and property b3). ∎

Proof of Fact E: If there exist m D_i -subtrees with 3m E_j -subtrees which represent different elements of M P_W executes the E^0_j with $j := 3m ::: 1$ before $N(B^{10}_2)$ (like in figure 9), the m u^1_i's, the m u^2_i's, and the m u^3_i's before $N(B^{30}_2)$, the m u^4_i's before $N(B^0_1)$, and the m u^5_i's and the m u^6_i's before $N(B^0_0)$.

If P_W executes m u^6_i's before $N(B^0_0)$ it has to execute 3m E^0_j subtrees before $N(B^{10}_2)$ (see property b8 and the construction of the D_i).

Each E^0_j cannot be executed before $N(C^{00}_j)$. So we can order the E^0_j subtrees according to their earliest execution time. Therefore we have the following order (see figure 9):

$$E^0_{3m}, E_{3m \Gamma 1}, \ldots, E_1$$

Assume that P_W executes some E^0_j subtrees where the E_j -subtrees represent the same element of M. Let $g : M \Gamma \mathbb{N}$ be a function, that give the number of E^0_j which are executed by P_W and represent one element m_i 2 M .

We say that there is a **hole** at m_j in the ordered sequence if P_W executes no E^0_j.

For each m_j with $g(m_j) \Gamma 1$ exist $g(m_j) \Gamma 1$ holes of m_k with $k \Gamma j$ (a hole with $j \Gamma k$ cannot be used to execute one E^0_j, because $N(C^{00}_k) + k \Gamma j \Gamma N(C^{00}_j)$, and P_W cannot execute a E^0_j subtree before $N(C^{00}_j)$). But a hole with $k \Gamma j$ cannot be used to execute a complete E^0_j subtree (because $k \Gamma j$).

So if P_W has to execute 3m E^0_j before $N(B^0_0)$ it has to execute exactly one E^0_j for each element m_j 2 M . ∎

Fact F directly follows from the construction of the D_i and E_j . ∎

These lemmata prove Theorem 2. ∎

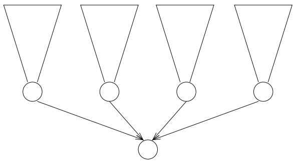
5. A Polynomial Time Algorithm for Complete Trees

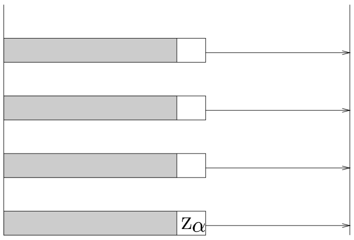
In this section an algorithm will be described that generates an optimal schedule for complete k -ary trees assuming a fixed communication delay. The proof of optimality goes indirectly and is rather involved.

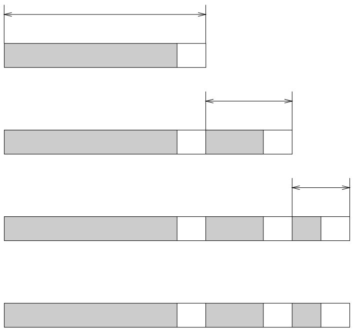
Proof of Theorem 3: Let $(G; \Gamma)$ with $G = (V; A)$ and Γ 2 $\mathbb{N}$ be the inputs of UDS where G is a complete k-ary tree. For x 2 V let G(x) denote the maximal subtree of G with root x .

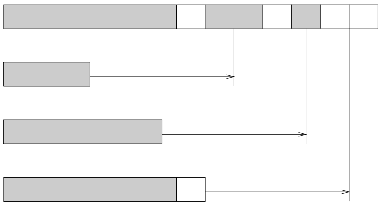
For each x 2 V we choose a processor P_x and construct a partial schedule S[x] for G(x) consisting of all operations executed by P_x. This will be done by a recursive algorithm **Schedule[fifi Pfi 0]**.

S[x] can be generated as soon as for all direct predecessors z_i of x S[z_i] has been constructed by calling **Schedule**[




z_α





2. For two direct predecessors z_i 6= z_j of x holds $G(z_i)\,\Gamma\,G(z_j)$ = ; . Thus, Fact 2 follows directly from the observation above.

3. The proof of Fact 3 goes along the same line as the proof of Fact 2.

4. If $t = t_k$ the claim is trivial.

   If $t\,\Gamma\,t_k$ P executes a direct predecessor z_i of x at step $t_k\,\Gamma\,1$. Thus, z_i is not executed by another processor P_i before $t_k\,\Gamma\,1\,\Gamma\,\Gamma$. If a direct predecessor z_j of x is not executed by P before t^0 and no other processor P^0 has executed z_j before $t^0\Gamma\,\Gamma$ then P cannot execute x at step t^0 and no other processor P^{00}can execute x before $t^0\Gamma\,\Gamma + 1$.

   Thus, Fact 4 is proved. ∎

The three conditions for a correct schedule follows directly from Fact 1, Fact 3, and the structure of $S[G]$. The normal form condition A1 follows directly from Fact 1 and Fact 2. Condition A2 follows directly from Fact 3, Fact 4, and from the construction of $S[x]$. The normal form condition A3 follows directly from Fact 1 and the number of triples of the subschedule **Schedule**[**fifi Rfi t**]. ∎

We can show that this normal form condition is no restriction in case of complete trees. This property is nontrivial since it does not hold for arbitrary trees, even for binary trees. Figure 13 shows a (1,2)-tree and Figure 14 an optimal schedule for this tree. It can easily be seen that there does not exist a schedule in normal form of the same length.

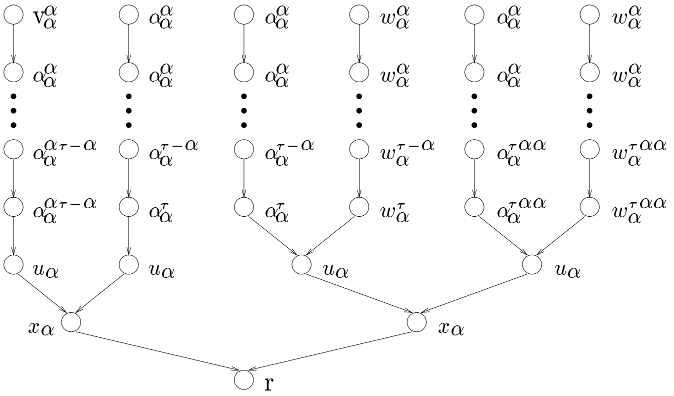
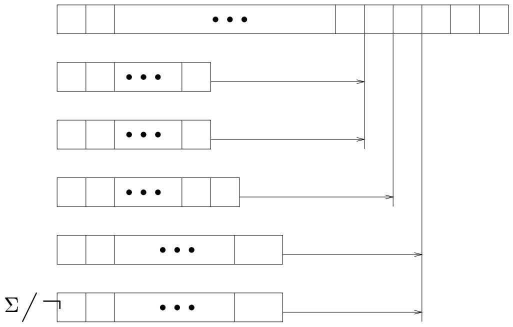


Figure 13: A tree T_B that does not have an optimal schedule in normal form.



Proof: Let S^0 be an optimal schedule for $(G;\Gamma)$, so that for all $G(x)$ we can find an optimal schedule $S^x \Gamma S^0$ which is a subschedule of S^0. S^0 can be generated by the union of optimal schedules for all $G(x)$. W.l.o.g. we assume that if a processor P_q executes a node u, which is not the last in its schedule, it also executes the direct successor of u. If a node $u = S_q(j)$ violates this assumption it is possible to execute u by a separate processor and all tasks of $G(u)$ in S_q can be deleted. Thus, each processor P_y executes only nodes of a connected subtree with root $y = S_y(T(S_y))$.

It is possible that S^0 violates the three conditions A1, A2, and A3, but we can modify S^0 into a new schedule that fulfills the three conditions within three steps. After the first step the modified schedule fulfills A1, after the second step it fulfills A2, and after the third step it also fulfills A3.

Assume that S^0 violates condition A1. For each such processor P_q we will construct a sequence of schedules:

$$S_q^0 = S_q^{[0]} \Gamma S_q^{[1]} ::: \Gamma S_q^{[T(S_q \tau\tau} = S_q$$

that ends with a schedule S_q fulfilling A1.

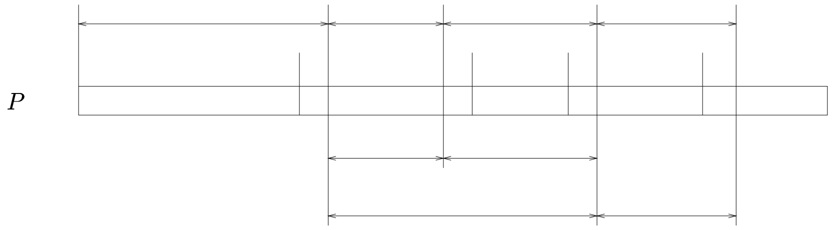
In the i-th phase $S_q^{[i]}$ will be generated from $S_q^{[i\Gamma 1]}$ such that the following properties hold:

P1: A transformation does not increase the finishing time of P_q, that is $T(S_q^{[i]}) = T(S_q^{[i\Gamma 1]})$.

P2: If the subschedule $f\ S_q^{[i\Gamma 1]}(j)\ j\ j\ \Gamma\ i\ \Gamma\ 1\ g$ fulfills A1 then does the longer subschedule $f\ S_q^{[i]}(j)\ j\ j\ \Gamma\ i\ g$.

If $f\ S_q^{[i\Gamma 1]}(j)\ j\ j\ \Gamma\ i\ g$ fulfills A1 then let $S_q^{[i]} := S_q^{[i\Gamma 1]}$. Otherwise $S_q^{[i\Gamma 1]}$ has the structure as described in figure 15 where $ff_0 := i$. Set $S^{00} := S_q^{[i\Gamma 1]}$.

Thus, ff_0 is the minimal time such that $f\ S^{00}(j)\ j\ 0\ \Gamma\ j\ \Gamma\ ff_0\ g$ violates condition A1. Let $a_0 := S^{00}(ff_0)$ be the node that is executed by P_q at step ff_0.



$$\begin{array}{lll} A & := & f\,S^{00}(j)\; j\; 0\,\Gamma\, j\,\Gamma\, ff_1\, g \\ B & := & f\,S^{00}(j)\; j\; ff_1\,\Gamma\, j\,\Gamma\, fi_1\, g \\ C & := & f\,S^{00}(j)\; j\; fi_1\,\Gamma\, j\,\Gamma\, fi_0\, g \\ D & := & f\,S^{00}(j)\; j\; fi_0\,\Gamma\, j\,\Gamma\, ff_0\, g \end{array}$$

The following three operations will be used to modify S^{00}.

Π 1 : Delete all tasks of a subtree G_1.

Let F be the set of executions $(x; P_q; t)$ of S^{00} where x is a node of G_1. The result of this procedure is given by $S^{00} n\, F$.

Π 2 : Switch the execution of two intervals I_1 and I_2 of S^{00} that directly follow each other if the tasks in I_2 can be processed earlier.

After the switch processor P_q first executes I_2 and then I_1.

Let i^f_1 be the first step of I_1, let i^f_2 be the first step of I_2 and i^l_2 be the last step of I_2. The result of this procedure is given by

$$S^{00} n\, (I_1 [I_2)$$
$$[f\,(x; P_q; t\,\Gamma\, i^f_2 + i^f_1)\; j\; (x; P_q; t)\; 2\, I_2\, g$$
$$[f\,(x; P_q; t + (i^l_2 \Gamma\, i^f_2) + 1)\; j\; (x; P_q; t)\; 2\, I_1\, g:$$

Π 3 : Switch the scheduling of two subtrees $G_1 = (V_1; A_1)$ and $G_2 = (V_2; A_2)$ of identical height.

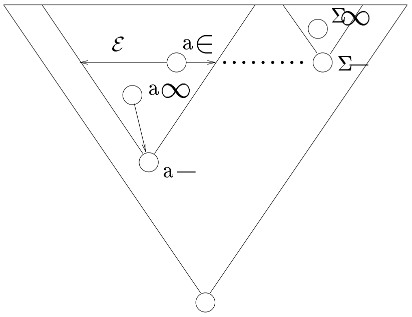
After the switch processor P_q at step t executes a task x_1 of G_1 if it had originally executed the corresponding task $x_2\, 2\, V_2$ at step t and vice versa.

Let S^{000} be the subschedule of S^{00} such that each execution of S^{000} is an execution of a node of G_1 or G_2 and let f be an isomorphism between G_1 and G_2. Then result of this procedure is $S^{00} n\, S^{000} [f\,(f(x); P_q; t)\; j\; (x; P_q; t)\; 2\, S^{000} g$.

Note that P_q originally does not have to compute all nodes in G_1 or G_2. After the switch P_q computes as many nodes in G_1 as it had originally done in G_2 and vice versa.

For a transformation we have to distinguish the following cases:

1. $\mathrm{height}(G(a_0))\, \Gamma\, \mathrm{height}(G(b_0))$ (see figure 16):



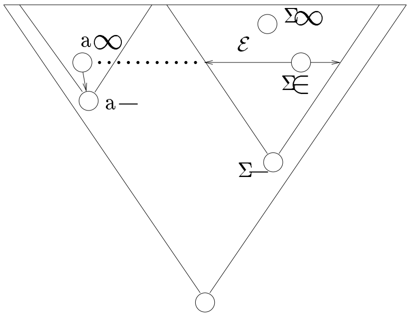


Figure 16: The subtrees $G(a_0)$ and $G(b_0)$ in case $\mathrm{height}(G(a_0))\, \Gamma\, \mathrm{height}(G(b_0))$.

Figure 17: The subtrees $G(a_0)$ and $G(b_0)$ in case $\mathrm{height}(G(a_0))\, \Gamma\, \mathrm{height}(G(b_0))$.

   (a) There exists a processor $P^0 6= P_q$ that executes b_0 at least Γ steps before P_q executes a_0. Thus, b_0 need not be executed by P_q and we can delete all tasks of $G(b_0)$ from S^{00} (operation Π 1).

(b) No processor P^0 executes b_0 Γ Γ steps before P_q executes a_0.

Let E be the set of tasks a of $G(a_0)$ with $depth(a) = depth(b_0)$. Note that all nodes of E are executed by P_q before step ff_0. Let a_2 be the task of E which is executed by P_q at latest.

i. P_q executes a_2 before step fi_0.

Thus, we can switch the execution of the interval where P_q executes the nodes of C with the interval where P_q executes the nodes of D (operation Π 2).

ii. P_q executes a_2 after step fi_0.

Thus, we can switch the scheduling of $G(b_0)$ and $G(a_2)$ within S^{00}, because $G(b_0)$ and $G(a_2)$ are complete k-ary trees of identical height and all successors of b_0 and a_2 are executed after the execution of a_2 (operation Π 3). After that we have to change the interval where P_q executes the tasks of $G(b_0)$ with the interval in which P_q executes the successors of a_2 starting with the direct successor of a_2 and ending with a_0 (operation Π 2).

Let $H(a_0) = A [B [C$ denote the set of nodes that violate condition A1 when scheduling a_0. In each of these cases after the transformation $H(a_0)$ Γ $A [B$, thus the size of the "bad nodes" has been decreased.

2. $height(G(a_0))$ Γ $height(G(b_0))$ (see figure 17):

(a) There exists a processor $P^0 6= P_q$ that executes a_1 at least Γ steps before P_q executes a_0. Thus, a_1 need not be executed by P_q and we can delete all tasks of $G(a_1)$ from S^{00} (operation Π 1).

In this case $H(a_0)$ loses at least the node a_1.

(b) No processor P_p executes a_1 at least Γ steps before P_q executes a_0.

Let E be the set of tasks b of $G(b_0)$ with $depth(b) = depth(a_1)$. Note that all nodes of E are executed by P_q between step fi_1 and step fi_0. Let b_2 be the task of E which is executed by P_q at latest. Thus, we can switch the scheduling of $G(a_1)$ and $G(b_2)$ within S^{00}, because $G(a_1)$ and $G(b_2)$ are complete k-ary trees of identical height and all successors of a_1 and b_2 are executed after the execution of b_2 (operation Π 3). After that we have to switch the interval in which P_q executes the tasks of $G(a_1)$ with the interval where P_q executes the successors of b_2 starting with the direct successor of b_2 and ending with b_0 (operation Π 2).

Since D becomes bigger $H(a_0)$ decreases.

Set S^{00} equal to this modified schedule. If necessary repeat the transformation of S^{00} with modified values for the parameters ff_0, ff_1, fi_0, and fi_1 until $f S^{00}(j) j j$ Γ $i g$ fulfills A1.

Since each transformation reduces $H(a_0)$, the iteration stops after a finite number of transformations, and yields a new subschedule $S_q^{[i]}$ fulfilling condition A1.

Condition A2 can be achieved by deleting all executions $(x; P_q; t)$ where

1. x is already computed by the same processor or
2. x is computed by some other processor P_x at least Γ steps before.

Condition A3 can be achieved as follows:

Let $(x; P_q; t)$ be the first execution where P_q executes no task at step t Γ 1, i.e. P_q is idle at step t Γ 1. If each direct predecessors of x is executed by P_q before t Γ 1 or by another processor before t Γ 1 Γ Γ then replace $(x; P_q; t)$ by $(x; P_q; t$ Γ $1)$. Otherwise there is a direct predecessor u of x

that is not executed by $\mathrm{P_q}$, but by another processor P by step $\mathrm{t\,\Gamma\,1\,\Gamma\,\Gamma}$. Then add $\mathrm{(u;P_q;t\,\Gamma\,1)}$ to the schedule.

Repeat this procedure until the schedule fulfills condition A3.

Each of these changes does not increase the finishing time of a processor. Thus, the resulting schedule is optimal and fulfills condition A1, A2, and A3. Thus, it is in normal from. ∎

The next lemma shows that all schedules in normal form have the same length.

Lemma 11 – –

1. It is easy to see that if the relations between $S_r^0(j\,\Gamma\,1)$, $S_r^0(j)$ and $S_r^{00}(j\,\Gamma\,1)$, $S_r^{00}(j)$ are of the same type then $S_r^0(j) = S_r^{00}(j)$.
    - (a) For sons this is immediately obvious.
    - (b) For brothers it follows from the unique ordering in which the sons of a node are scheduled in both schedules.
    - (c) If the relations between $S_r^0(j\,\Gamma\,1)$, $S_r^0(j)$ and $S_r^{00}(j\,\Gamma\,1)$, $S_r^{00}(j)$ are of type 3 it follows from the normal form condition A1 and the unique ordering in which the sons of a node are scheduled in both schedules.

      Define $k; l_0\ 2\ \mathbb{N}$ such that $S_r^0(j\,\Gamma\,1) = \mathrm{pred}(S_r^0; k; l_0)$. Let $x_0 := \mathrm{pred}(S_r^0; k+1; l_0)$, thus x_0 is a successor of $S_r^0(j)$ and $S_r^{00}(j)$.

      Now we can define a sequence of nodes $x_0; x_1; :::; x_s = S_r^0(j)$ where $x_i := \mathrm{pred}(S_r^0; 1; l_i)$ and $x_i = S_r^0(l_{i+1})$. It is easy to see that $S_r^{00}(j)$ is also an element of this sequence or $S_r^{00}(j)$ is a predecessor of $S_r^0(j)$. The assumption follows from the normal form condition A2.
2. If the relations between $S_r^0(j\,\Gamma\,1)$, $S_r^0(j)$ and $S_r^{00}(j\,\Gamma\,1)$, $S_r^{00}(j)$ are of different types one node $S_r(j)$ ($S_r\ 2\ fS_r^0; S_r^{00}g$) must be a predecessor of the other node $\overline{S}_r(j)$. So $S_r(j)$ is executed in schedule $\overline{S}$ by a processor $P^0\ 6\!= P$ before $j\,\Gamma\,\Gamma$ contradicting condition A2. ∎

Lemma 10 and Lemma 11 imply that all schedules S in F are optimal. The algorithm described above generates a schedule S in normal form (Lemma 9). Therefore S is an optimal schedule. The time complexity can be estimated as follows.

Lemma 12 ––– –

$f_{\Gamma;k}(h)$:= minimal time to schedule a complete k-ary tree of height h with delay Γ on an unbounded set of processors.

$g_{\Gamma;k}(h)$:= $f_{\Gamma;k}(h) \Gamma f_{\Gamma;k}(h \Gamma 1)$
the time spent by P for the subtrees $B_{Z_{\alpha}, \alpha \alpha \alpha , \alpha}$

Let us remark that with some more effort the parallel time bound can be improved to $O(\log k \Delta \log^2 n)$ which implies that UDS restricted to arbitrary complete trees – even with unbounded indegree – can be solved in NC^3.

The three tables at the end give some initial values of $g_{\Gamma;k}(h)$ for different parameters Γ and k. Note that g becomes periodic after a while. The period is given in bold numbers. The nonperiodic initial sequence increases with the delay Γ. It seems that its length depends on the distance of Γ to the nearest power of $k \Gamma 1$.

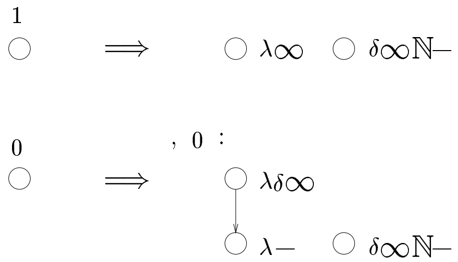
7. CDS-Problem

In [JKS89] Jung, Kirousis and Spirakis describe an $jVj^{O(\Gamma)}$time bounded algorithm for computing a schedule of minimal length. They consider a scheduling model where multiple computation of tasks are allowed. Consequently CDS is in P. In the following we show that this is the best possible characterization by proving completeness.

Theorem 6 ––– –––

1. If c is a Boolean variable x_i or $\overline{x}_i$ replace c by a Boolean constant given by the corresponding input of $\mathrm{ff}_1; : : : ; \mathrm{ff}_n$ of C .

2. If c is a Boolean constant ff the subgraph G_c is given by the graph G_{ff} described in figure 19.



(a) C_c on input $ff_1;:::;ff_n$ outputs 1:
At least one direct predecessors u^0 of c the value 1. Thus, the minimal schedule length for $G_{u'}$ is $2 \Delta d_{u_j} + 1 = 2 \Delta d_c \Gamma 1$. For the other predecessors u^{00} the minimal schedule length for $G_{u''}$ is at most $2 \Delta d_{u_i} + 2 = 2 \Delta d_c 1$. Therefore a processor that executes $\Gamma_{u''}$ can execute Γ_c at step $2 \Delta d_c + 1$.

(b) C_c on input $ff_1;:::;ff_n$ outputs 0:
The values of both direct predecessors u_1 and u_2 are 0. Thus, the minimal schedule length for G_{u_α} and G_{u_α} is $2 \Delta d_{u_\alpha} + 2 = 2 \Delta d_{u_\alpha} + 2 = 2 \Delta d_c$. Consequently, Γ_c cannot be executed before step $2 \Delta d_c + 2$.

2. c is an AND-gate:

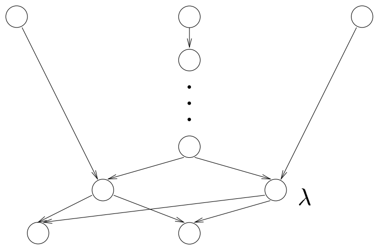
Observe that the subgraph of all ffi nodes of G is a forest F of outtrees since $ffi_{v_\alpha,\alpha}$ may be used for several $ffi_{c'}$-chains. The depth $ffi_{c;1}$ within F is $2 \Delta d_c \Gamma 1$. Since we allow multiple computation of tasks $ffi_{c;1}$ can be executed by a processor $P_{ffi_{c,\alpha}}$ by step $2 \Delta d_c$.

(a) C_c on input $ff_1;:::;ff_n$ outputs 1:
The values of both direct predecessors u_1 and u_2 of c are 1. Thus, the minimal schedule length for G_{u_α} and G_{u_α} is $2 \Delta d_{u_\alpha} + 1 = 2 \Delta d_{u_\alpha} + 1 = 2 \Delta d_c \Gamma 1$. Therefore $P_{ffi_{c,\alpha}}$ can execute Γ_c at step $2 \Delta d_c + 1$.

(b) C_c on input $ff_1;:::;ff_n$ outputs 0:
At least one of a direct predecessor u^0 of c has value 0. Thus, the minimal schedule length for $G_{u'}$ is $2 \Delta d_{u'} + 2 = 2 \Delta d_c$. Consequently, no processor can execute Γ_c at step $2 \Delta d_c + 1$. ∎

This lemma proves theorem 6 by applying it to the output gate of C. ∎

One can even achieve indegree $\Gamma 2$ and an arbitrary delay $\Gamma 2 \mathbb{N}^+$ by using the following graphs for AND (figure 23) and OR gates (figure 22) and setting the deadline to $T^\Gamma := (\Gamma + 3) \Delta \mathrm{depth}(C) + 1$:



with respect to the size of the delays. The following matrix summarizes the known results on the complexity of the delay scheduling problem with respect to the task graphs.

	DAGs	binary trees	complete trees
CDS	P-complete[JKS89] and [Th. 6]	P	NC
UDS	NP-complete[PY88]	NP-complete[Th. 1]	NC[Th. 5]
DS	jj	jj	NP-complete[Th. 2]

References

[C90] P. Chrétienne, Complexity of Tree-Scheduling with Interprocessor Communication Delays, MASI Report 90.5, 1990.

[CP91] P. Chrétienne and C. Picouleau, The Basic Scheduling With Interprocessor Communication Delays, MASI Report 91.6, 1991.

[GHR95] R. Greenlaw, H. J. Hoover and W. L. Ruzzo, Limits to Parallel Computation: P-Completeness Theory, Oxford University Press, 1995.

[GJ79] M. R. Garey and D. S. Johnson, Computers and Intractability, A Guide To the Theory of NP-Completeness, Freeman 1979.

[J91] A. Jakoby, Prozess-Prozessor-Allokationsprobleme für Baumförmige Datenflussgraphen, Diplomarbeit, Technische Hochschule Darmstadt, 1991.

[JKS89] H. Jung, L. Kirousis, P. Spirakis, Lower Bounds and Efficient Algorithms for Multiprocessor Scheduling of DAGs With Communication Delays, Proc. 1. Symposium on Parallel Algorithms and Architectures, 1989, 254 - 264.

[JR92] A. Jakoby, R. Reischuk, The Complexity of Scheduling Problems with Communication Delays for Trees, Proc. 3. Scandinavian Workshop on Algorithm Theory, 1992, 165-177.

[P91] C. Picouleau, Two New NP-complete Scheduling Problems With Communication Delays and Unlimited Number of Processors, MASI Report 91.24,1991.

[PY88] C. Papadimitriou and M. Yannakakis, Towards an Architecture-Independent Analysis of Parallel Algorithms, Proc. 20. ACM Symposium on Theory of Computing, 1988, 510-513, see also SIAM J. Comput. 19, 1990, 322-328.

[R91] R. Reischuk, Graph Theoretical Methods for the Design of Parallel Algorithms, Proc. 8. Symposium on Fundamentals of Computation Theory, 1991, 61 - 67.

[V93] B. Veltman, Multiprocessor Scheduling with Communication Delays, Ph.D. thesis, University of Technology Eindhoven, Department of Computer Science, Eindhoven, The Netherlands, 1993.

gΓ;2(h)	h =																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Γ = 0	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	1	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	1	**2**	**3**	2	3	2	3	2	3	2	3	2	3	2	3	2	3
4	1	**2**	**4**	2	4	2	4	2	4	2	4	2	4	2	4	2	4
5	1	2	**4**	**3**	4	3	4	3	4	3	4	3	4	3	4	3	4
6	1	2	**4**	4	4	4	4	4	4	4	4	4	4	4	4	4	4
7	1	2	4	5	**4**	4	4	4	4	4	4	4	4	4	4	4	4
8	1	2	4	6	**4**	4	4	4	4	4	4	4	4	4	4	4	4
9	1	2	4	6	**4**	**4**	**5**	4	4	5	4	4	5	4	4	5	4
10	1	2	4	7	**4**	**4**	**6**	4	4	6	4	4	6	4	4	6	4
11	1	2	4	8	4	**4**	**6**	**5**	4	6	5	4	6	5	4	6	5
12	1	2	4	8	4	4	7	5	**4**	**6**	**6**	4	6	6	4	6	6
13	1	2	4	8	5	4	7	6	4	**6**	**6**	**5**	6	6	5	6	6
14	1	2	4	8	6	4	7	6	5	**6**	6	6	6	6	6	6	6
15	1	2	4	8	7	4	7	7	5	6	7	**6**	6	6	6	6	6
16	1	2	4	8	8	4	7	8	6	8	**6**	**6**	**7**	6	6	7	6

Table 1: The increases of the optimal schedule length for complete binary trees.

gΓ;3(h)	h =																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Γ = 0	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	1	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	1	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	1	**3**	**4**	3	4	3	4	3	4	3	4	3	4	3	4	3	4
5	1	**3**	**5**	3	5	3	5	3	5	3	5	3	5	3	5	3	5
6	1	**3**	**6**	3	6	3	6	3	6	3	6	3	6	3	6	3	6
7	1	3	**6**	**4**	6	4	6	4	6	4	6	4	6	4	6	4	6
8	1	3	7	4	**6**	**5**	6	5	6	5	6	5	6	5	6	5	6
9	1	3	8	4	7	5	**6**	6	6	6	6	6	6	6	6	6	6
10	1	3	9	4	8	5	7	**6**	6	6	6	6	6	6	6	6	6
11	1	3	9	5	8	**6**	**7**	6	7	6	7	6	7	6	7	6	7
12	1	3	9	**6**	**8**	6	8	6	8	6	8	6	8	6	8	6	8
13	1	3	9	**7**	**8**	7	8	7	8	7	8	7	8	7	8	7	8
14	1	3	9	**8**	8	8	8	8	8	8	8	8	8	8	8	8	8
15	1	3	9	**9**	**8**	9	8	9	8	9	8	9	8	9	8	9	8
16	1	3	**9**	9	9	9	9	9	9	9	9	9	9	9	9	9	9

Table 2: The increases of the optimal schedule length for complete ternary trees.

gΓ;4(h)	h =																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Γ = 0	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	1	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	1	**4**	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4	1	**4**	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	1	**4**	**5**	4	5	4	5	4	5	4	5	4	5	4	5	4	5
6	1	**4**	**6**	4	6	4	6	4	6	4	6	4	6	4	6	4	6
7	1	**4**	**7**	4	7	4	7	4	7	4	7	4	7	4	7	4	7
8	1	**4**	**8**	4	8	4	8	4	8	4	8	4	8	4	8	4	8
9	1	4	**8**	**5**	8	5	8	5	8	5	8	5	8	5	8	5	8
10	1	4	9	5	**8**	**6**	8	6	8	6	8	6	8	6	8	6	8
11	1	4	10	5	9	6	**8**	**7**	8	7	8	7	8	7	8	7	8
12	1	4	11	5	10	6	9	7	**8**	8	8	8	8	8	8	8	8
13	1	4	12	5	11	6	10	7	9	**8**	8	8	8	8	8	8	8
14	1	4	12	6	11	7	10	**8**	**9**	8	9	8	9	8	9	8	9
15	1	4	13	6	12	7	11	**8**	**10**	8	10	8	10	8	10	8	10
16	1	4	14	6	12	**8**	**11**	8	11	8	11	8	11	8	11	8	11

Table 3: The increases of the optimal schedule length for complete 4'ary trees.