Calculation of latency of real-time system and fixed-parameter tractibility of UET-UCT scheduling problems

Ning Tang

Calculation of latency of real-time system and fixed-parameter tractibility of UET-UCT scheduling problems

Thèse de doctorat de l'Université Sorbonne

Ecole Doctorale Informatique, Télécommunications et Electronique(ED130)

Spécialité de doctorat: théorie de l'informatique

Unité de recherche : LIP6 ALSOC

Thèse présentée et soutenue à l'Université Sorbonne,

le 30/11/2022, par :

NING TANG

Composition du Jury

Alix Munier-Kordon Professeure des universités, Sorbonne Université	Directeur de thèse
Claire Hanen Professeure des universités, Sorbonne Université	Président (e) du jury
Nicod Jean-marc Professeur des universités, Franche-Comté Electronique Mécanique Thermique et Optique – Sciences et Technologies	Examinateur
Pautet Laurent Professeur des universités, LTCI, Telecom Paris	Examinateur
Giroudeau Rodolphe Maître de Conférences (HDR), Laboratoire LIRMM	Rapporteur
T'kindt Vincent Professeur des universités, Université de Tours	Rapporteur

TABLE OF CONTENTS

Page

ABSTRACT

This thesis considers two main problems.

The first problem considers the minimization of the makespan and the maximum lateness for a set of dependent tasks with unit duration, unit communication delays release times and due dates. Each task requires one processor for its execution and duplication is not allowed. Algorithms are given for scheduling on limited and unlimited number of processors. Time windows of tasks are built from upper bounds of the makespan and the minimum maximum lateness respectively. A fixed-parameter algorithm based on a dynamic programming approach is developed to solve this optimization problem. The parameter considered is the pathwidth of the associated interval graph. They are, as far as we know, the first fixed-parameter algorithms for a scheduling problem with communication delays and a limited number of processors.

The second problem considers real-time systems. Automotive and avionics embedded systems are usually composed by several tasks submitted to complex timing constraints. In this context, LET paradigm was introduced to improve the determinism of a system of tasks that communicate data through shared variables. The age latency corresponds to the maximum time for the propagation of a data in these systems. Its precise evaluation is an important challenging question for the design of these systems. We considered a set of multi-periodic tasks that communicate data following the LET paradigm. Our main contribution is the development of mathematical and algorithm tools to model precisely the dependence between tasks executions. These tools will be considered to experiment an original methodology for computing the age latency of the system. They allow to handle the whole graph instead of particular chains and to extract automatically the critical parts of the graph. Experiments on random generated graphs proved that systems with up to 90 periodic tasks with an hyper-period bounded by 100 can be handled within a reasonable time.

ACKNOWLEDGMENTS

Countless people supported my effort on this essay. First of all, I would like to thank my supervisor, Alix Munier-Kordon, for her guidance throughout my thesis. she has provided invaluable feedback on my analysis and writing, from time to time responding to emails and correcting late at night and early in the morning. I learned what is passion for research. I learned to work hard. We have argued and disagreed, but they will never effect my sincere love and gratefulness to her.

My parents have given me priceless and unconditional support for these three years. In their eyes, I am always the best daughter in the world. They gave me the confidence and strength without which I could not finish my PhD.

My boyfriend accompanied me everyday through this hard journey. He saw all my emotional ups and downs, he knows all the positive and negative thoughts, he also laughed all my laughs and wept all my tears. I can not imagine a day without him at my side.

I am very lucky to have all the PhDs and interns in the same office as me. Their random Greek jokes and french complains motivated me to go to my office everyday.

Last but not least, I would like to thank Sorbonne University and Lip6 lab for providing me the fund and help for my PhD research.

1. Introduction

This thesis has two parts. The first part is to calculate the latency of real time systems; the second part is to give fixed-parameter tractable algorithms for fundamental scheduling optimization problems.

1.1 Fixed-parameter tractable algorithms for fundamental scheduling optimization problem

The concept of scheduling dates back to ancient times: Sun Tzu [85] mentioned about scheduling strategy for military management 2500 years ago. The development of scheduling tools on computers such as Critical Path Method (CPM) can be traced back to mid 1956 with 'UNIVAC1' computer, one of the first computers used by commercial business. After 1970s, the wave of convenient and cheap personal computers (PC) spawned dozens (if not hundreds) of PC based scheduling systems. The evolution of scheduling tools closely tracked the development of computers and it will continue to do so.

The obstacle of scheduling problem is to find the optimal solution in a reasonable time. Fixed-parameter tractable algorithm is a tool to solve hard problems in polynomial time when some parameters in the problem are bounded.

In this thesis, we attempted to use fixed-parameter tractable algorithm and dynamic programming tools to find optimal solutions in polynomial time with a fixed parameter in fundamental scheduling problems.

The model we considered is unit execution time and unit communication time (UET-UCT) scheduling model, which is one of the most researched model in scheduling theory field. We considered this model with two different criteria, makespan and maximum lateness.

First of all, in chapter 2, we gave a mathematical definition of the model we considered, briefly introduced parameter complexity theory and the parameter we considered. Then in chapter 3, we

introduced two kernelization of this model: modification of release dates and deadlines of tasks and active schedules. In chapter 4, we gave an executable algorithm for makespan optimization on UET-UCT model on unlimited number of machines, proved the correctness of this algorithm and calculated the complexity which shows that this is a fixed parameter tractable algorithm. After, we extended the algorithm to optimize maximum lateness on limited number of machines in chapter 5.

We proved in this thesis that the scheduling problem $P|r_i, prec, p_i = 1, c_{ij} = 1|L_{max}$ and $\overline{P}|r_i, prec, p_i = 1, c_{ij} = 1|C_{max}$ are fixed-parameter tractable parameterized with pathwidth of the interval graph of time windows. We extended a previous approach [72] to tackle problems with communications delay and a limited number of machine, also to optimize the maximum lateness. We limit our enumeration to active schedules, which decreases the worst-case complexity of the method. These are the first two fixed parameter tractable algorithm on scheduling with communication delay problems.

1.2 Calculation of age latency in real time systems

Automotive and avionics embedded systems are usually composed by several tasks submitted to complex timing constraints. In this context, safety is one of the most important features. In LET paradigm, real-time tasks communicate through shared buffers, the time of reading input and writing output are determined regardless of the execution time of tasks. LET is motivated by the observation that the relevant behavior of real-time systems is determined when input is read and output is written and not when programs execute. The age latency of real-time systems corresponds to the maximum time for the propagation of data. The precise evaluation of the upper and lower bounds of age latency is an important challenging question for the design of these systems.

We consider in this thesis a set of multi-periodic tasks that communicate data following the LET paradigm. In chapter 6, we formally defined the real-time model our characterisation of the dependence between tasks executions are presented in Section 6.2. Section 6.3 and 6.4 explain the construction of a partial expanded graph of a real-time system. Section 6.5 presents our algorithm

on the partial expanded graph which leading to an optimum upper bound of the age latency. This algorithm is experimented in Section 6.6 on a the case study ROSACE [76] and in Section 6.7 on randomly generated graphs.

Our main contribution is the development of mathematical and algorithm tools to model precisely the dependence between tasks executions. These tools will be considered to experiment an original methodology for computing the age latency of the system. They allow to handle the whole graph instead of particular chains and to extract automatically the critical parts of the graph. Experiments on random generated graphs proved that systems with up to 90 periodic tasks with an hyper-period bounded by 100 can be handled within a reasonable time.

2. Preliminary notions and context

Scheduling theory is a theory about the allocation of resources to activities over time. Resources can be computation processors, manufacturing machines, workers, teachers, etc. Activities can be program tasks in computer operating systems, steps of a complex project, operations in a production process, lessons in schools, etc. Scheduling theory can be applied in various fields like theory of computer science, management science, etc. For machine scheduling problems, algorithms have been widely investigated since the introduction of operation systems (OS) in 1950s [21, 37, 46, 87]. In this thesis, we focused on machine (processor) scheduling problems with specific conditions and applications.

The aim of this chapter is to give a brief view of the concepts we applied for in optimizing schedules with communication delays respect to different criteria. First we will give an overview on the classic scheme of machine scheduling models, then we will focus on the models with communication delays, after we will discuss common criteria considered in scheduling optimization problems. At last, we will introduce mainly researched scheduling strategies and complexities, especially on active schedules and fixed parameter complexity.

2.1 Homogeneous scheduling models with communication delays

The study of scheduling problems with interprocessor communication delays are rapidly growing resulting from the development of parallel and distributed memory systems. This phenomenon occurred due to a large range of applications including data mining, multimedia and bio-computing. Despite that machine scheduling theory has been intensively study since 1950's as we mentioned before, the study on theories concerning communication delays is much younger and still has a big room for expansions [23, 37, 44, 87].

In this thesis, we considered homogeneous scheduling models with communication delays, where we have parallel identical machines fully connected and the communication between ma-

chines are not negligible. If two tasks are communicating and they are allocated on different machines, a communication delay has to be considered between them. This means the higher for the level of parallel, the higher for the time cost for communication. Therefore, we need to find a compromise between parallel executions and sequential executions, which is main challenging part of this problem.

In the main research field, we separate homogeneous scheduling delay models into three different categories according to the ratio between execution time and communication time as follows:

1. Small communication time (SCT) scheduling models have tasks with communication time smaller than execution time. SCT models have some important applications such as parallel matrix multiplication on metacomputing platforms [57].

2. Unit execution time and unit communication time (UET-UCT) scheduling models contains tasks with the same communication time and execution time. This model is the fundamentally considered in scheduling theory.

3. Large communication time (LCT) scheduling models considers tasks with larger communication time than execution time. As the parallel and distributed systems becomes larger and larger, this model get more and more attentions recently with application in cloud computing [60].

2.1.1 Problem definition and notation

An instance of a homogenerous scheduling problem with communication delays is specified by: a set $\mathcal{T} = \{1, 2, \ldots, n\}$ of n tasks is to be executed on an unlimited number of machines (sometimes also called as processors). Each machine can process at most one task at a time. Tasks have a execution processing time p_i and are partially ordered by a directed acyclic graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$, also called precedence graph. Let t_i be the starting time of the task i. An arc $(i, j) \in \mathcal{A}$ is called a precedence relation between task i and task j. For any arc $(i, j) \in \mathcal{A}$, task i must finish its execution before the task j starts executing, i.e. $t_i + p_i \leq t_j$. If tasks i and j are

assigned to different processors, a communication delay with duration c_{ij} must be added after the execution of task i, to send data to task j and thus $t_i + p_i + c_{ij} \leq t_j$.

In scheduling theory, the objective functions for optimization problems are generally limited to regular criteria, which are defined as follows [8, 25].

Definition 2.1.1. Let $C_i(\sigma)$ and $G(\sigma)$ denote respectively the completion time of task i and the value of criterion G for schedule σ. If $C_i(\sigma) \leq C_i(\sigma'), \forall i \in \{1, 2, \ldots, n\}$ implies $G(\sigma) \leq G(\sigma')$ for two schedules σ and σ', the criterion G is said to be a regular criterion.

Some common researched criteria are listed below:

1. Makespan $C_{max}(\sigma) = \max\limits_{i \in \{1,2,\ldots,n\}} C_i(\sigma)$.
2. Total weighted completion time $C(\sigma) = \sum\limits_{i=1}^{n} w_i C_i(\sigma)$.
3. Total weighted tardiness with due dates $\sum\limits_{i=1}^{n} w_i T_i(\sigma)$ where $T_i(\sigma) = max\{C_i(\sigma) - d_i, 0\}$.
4. Maximum lateness with deadlines $L_{max}(\sigma) = \max\limits_{i \in \{1,2,\ldots,n\}} (C_i(\sigma) - d_i)$.
5. Total number of tardy tasks with due dates $U(\sigma) = \sum\limits_{i=1}^{n} U_i(\sigma)$,

   where $U_i(\sigma) = \begin{cases} 1 & \text{if } C_i(\sigma) > d_i \ , \\ 0 & otherwise, \end{cases}$

where w_i is the weight of task i and d_i is the due date of task i, for all $i \in \{1, 2, \ldots, n\}$.

Among these, makespan is the most basic and fundamental criterion used in scheduling research. Makespan indicates the length of time that takes from the start of tasks to the end. One may consider other objective functions $f(C_1(\sigma), \ldots, C_n(\sigma))$ depending on the completion time of tasks such as the total weighted completion time. In manufacturing environments with due dates constraints, criteria like maximum lateness and total weighted tardiness are more appropriate as objective functions. As you can see, all these objective functions are monotone non-decreasing functions in completion time $C_i(\sigma)$, so they are all regular criteria. Besides, many reductions exist

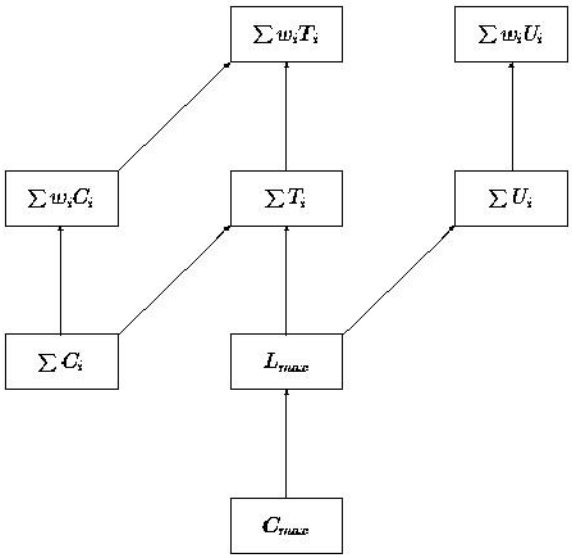


Figure 2.1: Reductions for objective functions [91].

between scheduling problems with different objective functions, see in Figure 2.1. We have $\sum_i T_i$ and $\sum_i U_i$ reduce to $\sum_i w_i T_i$ and $\sum_i w_i U_i$ by setting $w_i = 1$ for all i. Furthermore, we have C_{max}, $\sum_i C_i$ and $\sum_i w_i C_i$ reduce to L_{max}, $\sum_i L_i$ and $\sum_i w_i L_i$ by setting $d_i = 0$ for all i.

Finally, for decision problems and any threshold value y, for all $i \in \{1, 2, \ldots n\}$, we have

$$\begin{aligned} L_{max} \leq y &\Leftrightarrow C_i - d_i \leq y \\ &\Leftrightarrow C_i - (d_i + y) \leq 0 \\ &\Leftrightarrow \max\{0, C_i - (d_i + y)\} \leq 0 \\ &\Leftrightarrow \sum_{i=0}^{n} T_i = \sum_{i=0}^{n} \max\{0, C_i - (d_i + y)\} \leq 0 \\ &\Leftrightarrow \sum_{i=0}^{n} U_i \leq 0. \end{aligned}$$

Among all these objective functions, C_{max} and L_{max} are the most basic ones in optimization

scheduling problems. This thesis focus on optimization of makespan and maximum lateness.

To systematize scheduling models, a three field of classification $\alpha|\beta|\gamma$ was introduced by Graham et al. [46]. The first entry α specifies the processor environment, β indicates job characteristics, and γ denotes objective functions, also called optimization criteria.

For the first field $\alpha = \alpha_1\alpha_2$ specifying the machine environment, we have α_1 indicates the types of machines and α_2 indicates the number of machines. Let $\circ$ donates the empty symbol. All possible values of α_1 are characterized as follows:

$\alpha_1 = \circ$: single machine;

$\alpha_1 = P$: identical parallel machines. Tasks have the same processing time on different machines;

$\alpha_1 = Q$: uniform parallel machines. A processing speed is specified for each machine;

$\alpha_1 = R$: unrelated parallel machines;

$\alpha_1 = O$: open shop;

$\alpha_1 = F$: flow shop;

$\alpha_1 = J$: job shop;

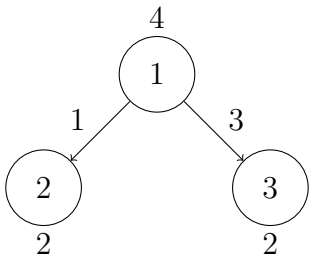
If α_2 is a positive number, then there is a fixed number of machines. If $\alpha_2 = \circ$, then the number of machines is unavailable.

For the second field β donating a set of tasks characteristics, we have some popular definitions listed below:

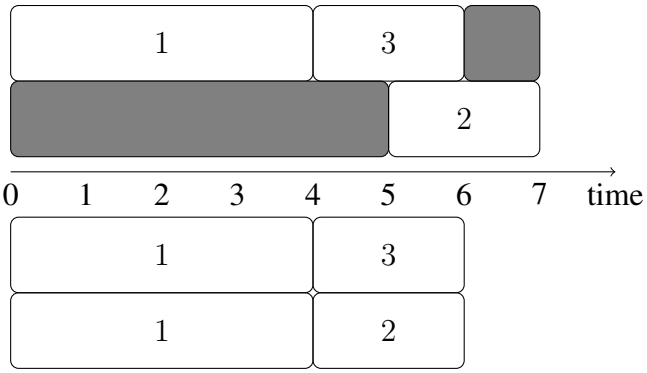
1. pmtn: Preemption is allowed, if not specified, then preemption is not allowed.

2. prec, tree, ...: A precedence relation between jobs is specified. An acyclic directed graph $\mathcal{G}$ is used to present the precedence relation, called precedence graph. One can specify the type of precedence graph such as tree and so on. If not mentioned, then no precedence relation is specified.

3. r_i: Release dates may differ and are specified per job. If not mentioned, we assume that release dates are all equal to zero.



(a) The precedence graph to be scheduled.



(b) The best schedule without duplication above the time axe and the best schedule with duplication below.

Figure 2.2: Example of task duplication.

4. $p_{ij} = 1$: Each job has unit processing time.

For the third field γ, it refers to the optimization criteria.

For example, the problem to minimize the maximum lateness on a single machine subject to a general precedence constraints, it can be written as $1|prec|L_{max}$. UET-UCT scheduling problem on unlimited number of processors can be noted using three-field notation as $\overline{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$.

In a schedule, if duplication is allowed, tasks can be executed more than once. Duplication of tasks can yield a smaller duration of a schedule, see example in Figure 2.2. Besides, when duplication is allowed, the scheduling problems are easier. For example, when SCT assumptions are met, the $\overline{P}|prec, p_i, c_{ij}, dup|C_{max}$ problem is proved by Colin and Chretienne [24] to be polynomial solvable. In this thesis, we only consider the problems without duplication.

When duplication is not allowed, each task must be processed only once. So a schedule can be entirely defined by the starting time of each task i and a assigned processor π_i. Since processors are identical, a task can be randomly assigned to a processor if there are communication delays between this task and all its predecessors, otherwise, the task has to be assigned to the same processor as its predecessor between which there is no communication delay. Therefore, a schedule $\sigma(\mathcal{G})$ can be sufficiently represented as a vector $\sigma(\mathcal{G}) = (t_1, t_2, \ldots, t_n)$ where t_i is the starting time

of task $i \in \mathcal{T}$.

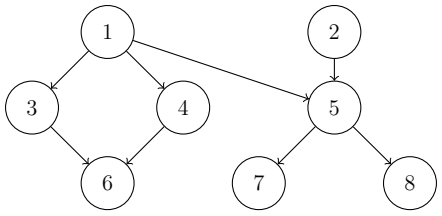
For each task $i \in \mathcal{T}$, let $\Gamma^+(i)$ (resp. $\Gamma^-(i)$) be the set of successors (resp. predecessors) of i, i.e. $\Gamma^+(i) = \{j \in \mathcal{T}, (i,j) \in \mathcal{A}\}$ and $\Gamma^-(i) = \{j \in \mathcal{T}, (j,i) \in \mathcal{A}\}$. The problem $\overline{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$ can be modelled by an integer linear program (P) defined below. For any arc $e = (i,j) \in \mathcal{A}$, we note x_{ij} as the signal of communication delay between the tasks i and j. We set $x_{ij} = 0$ if the task j is executed just after the task i on the same processor; in this case, there is no communication delay between them. Otherwise, $x_{ij} = 1$.

$$(P) \begin{cases} Objective : \min C & \\ \forall e = (i,j) \in \mathcal{A}, t_i + p_i + x_{ij}c_{ij} \leq t_j & (1) \\ \forall i \in \mathcal{T}, t_i + p_i \leq C & (2) \\ \forall i \in \mathcal{T}, \sum_{j \in \Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1 & (3) \\ \forall i \in \mathcal{T}, \sum_{j \in \Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1 & (4) \\ \forall i \in \mathcal{T}, t_i \in \mathbb{N} & (5) \\ \forall e = (i,j) \in \mathcal{A}, x_{ij} \in \{0,1\} & (6) \end{cases}$$

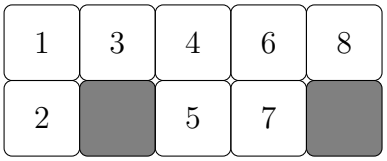
Variables are the starting times $t_i, \forall i \in \mathcal{T}$ of the tasks, the communication delays $x_{ij}, \forall (i,j) \in \mathcal{A}$ and the makespan C. Inequalities (1) express precedence relations and communication delays between tasks executions. Inequalities (2) define the makespan. Inequalities (3) express that any task has at most one successor performed at its completion time on the same processor. Similarly, inequalities (4) express that any task has at most one predecessor performed just before its starting time on the same processor. Any feasible schedule $\sigma(\mathcal{G})$ corresponds to a feasible solution of (P).

Let us consider for example the precedence graph of a UET-UCT model, presented by Figure 2.3a for $\mathcal{T} = \{1, 2, \ldots, 8\}$. Figure 2.3b presents an associated feasible schedule $\sigma(\mathcal{G})$ of makespan 5. Associated starting time is $t = (0, 0, 1, 2, 2, 3, 3, 4)$.

For the problem with limited number of processors $m < n$, $P_m|prec, p_i = 1, c_{ij} = 1|C_{max}$,



(a) A precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$.



(b) An optimum schedule for the precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ of Figure 2.3a.

Figure 2.3: A precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ and an associated optimum schedule.

one more condition is needed to be added in the program (P):

$$\forall \alpha \in \{0, 1, \ldots, n-1\}, |\{i, t_i = \alpha, i \in \mathcal{T}\}| \leq m$$

Therefore, the program (P) is no longer a linear program.

2.1.2 Motivations for scheduling optimization on UET-UCT model

Large parallel and distributed systems (cluster, grid and global computing) are the main stream in the computational world and the new challenges for researchers and developers on a large range of domains including data mining, multimedia, bio-computing and so on. More and more communication time is required in parallel and distributed systems, sometimes the communication time between processors is even longer than the real processing time. However, it is still a very large unexploited field to provide adequate and efficient algorithms and software tools for managing parallel resources with non negligible communication delays. This section will give existing algorithms and complexity results on scheduling problems with unit communication delays.

Given a precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ to be scheduled on a parallel system, a starting time t_i is to be allotted to each task $i \in \mathcal{T}$. In homogenerous scheduling model with communication delays, processors are identical and fully connected. Each arc $(i, j) \in \mathcal{A}$ represents a potential data transfer between task i and task j when i and j are processed on two different processors. So the difficulty, in this model, is to optimize the efficiency between sequential execution and parallel execution. Basic scheduling problems with communication delays were intensively studied since

the 1990s due to the importance of applications, *see.* the surveys [21, 23, 37, 44, 46, 87].

With the limitation on the number of processors, the problem $P|prec, p_i = 1, c_{ij} = 1|C_{max}$ is first introduced by Rayward-Smith [78], who has proved that the problem is NP-hard and showed that an active schedule is no longer than $3 - 2/m$ times the optimum. A schedule is active if no task can start earlier without increasing the start time of another task. Hoogeveen et al. [55] have studied the decision problem and shown that $P|prec, p_i = 1, c_{ij} = 1|C_{max} \leq 3$ can be solved in polynomial time and $P|prec, p_i = 1, c_{ij} = 1|C_{max} \leq 4$ is $NP-$complete. Lenstra et al. [63] showed that even if the precedence relation consists of a collection of trees, the problem $P|tree, p_i = 1, c_{ij} = 1|C_{max}$ is NP-hard. An exact dynamic programming algorithm of time complexity $\mathcal{O}(2^{w(G)}.n^{2w(G)})$ was developed by Veltman [88] for $P|prec, p_i = 1, c_{ij} = 1|C_{max}$ where $w(G)$ is the width of the precedence graph G. The width of a directed graph is the cardinality of the largest antichain of G. for the problem $P|prec, p_i = 1, c_{ij} = 1|C_{max}$ and $P|prec, p_i = 1, c_{ij} = 1|L_{max}$. Hanen and Munier [49] gave an approximation algorithm to the problem $P|prec, p_i = 1, c_{ij} = 1|C_{max}$ and proved that the worst relative performances of their algorithms are bounded by $7/3 - 4/m$.

Without the limitation on the number of processors, Picouleau [77] has shown that the decision problem $\overline{P}|prec, p_i = 1, c_{ij} = 1|C_{max} \leq 8$ is NP-complete. Hoogeveen et al. [55] have improved the result and showed that a polynomial-time algorithm without duplication exists for solving the problem $\overline{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$ when the makespan is bounded by 5, but it is NP-complete when the makespan is bounded by 6. So unless $P = NP$, there is no approximation algorithm with a worst case ratio smaller than $7/6$. Munier and König [71] gave a dynamic list scheduling heuristic with a relative performance equal to $4/3$ using integer linear programming.

2.2 Parameterized complexity for scheduling problems

Parameterized complexity classifies computational problems according to their difficulty with respect to multiple parameters of the input or output. The complexity of a problem is then measured as a function of not only the number of bits in the input but also their parameters. It classifies

$NP-$hard problems into finer sets than in the classical computational complexity theory. The first systematic work on parameterized complexity was done by Downey and Fellows [35].

2.2.1 Fixed-parameter tractability

Under the assumption that $P \neq NP$, many fundamental problems, such as $k-$vertex cover [20], require exponential running time, but are polynomial in the input size and only exponential or worse in a parameter k. Formally, a vertex cover V' of an undirected graph $G = (V, E)$ is a subset of V such that $uv \in E \Rightarrow u \in V' \vee v \in V'$. $k-$vertex cover problem is to decide if G has a vertex cover of at most k vertices, given a graph G and a parameter k. Even though it is considered unlikely to find an exact polynomial algorithm for $NP-$hard or $NP-$complete problems, some problems can be solved by algorithms that are only exponential to a fixed parameter but polynomial to the input size. Such an algorithm is named a fixed-parameter tractable (FPT-) algorithm. In this case, the problem can be solved efficiently when the fixed parameter is bounded by a small number. Therefore, FPT can be considered as a two-dimension complexity theory, which is defined by Downey and Fellows [35] as follows:

Definition 2.2.1. A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the parameter of the problem. A parameterized problem L is fixed-parameter tractable if the question "$(x, k) \in L$?" can be decided in running time $f(k) \cdot |x|^{O(1)}$, where f is an computable function depending only on k. The corresponding complexity class is called FPT.

This definition is to exclude functions of the form $f(n, k)$, such as n^k. Instead, we have XP as the class of parameterized problems that can be solved in time $n^{f(k)}$ for some computable function f. Thus, FPT is a strict subset of XP. Furthermore, we have the class FPL (fixed parameter linear) as the class of parameterized problems solvable in time $f(k) \cdot |x|$ for some computable function f. FPL is thus a strict subset of FPT.

For instance, for the $k-$vertex cover problem, the input size is the number of the vertex and arcs, the parameter can be k: the number of vertices in the cover. In the contrary, graph coloring

problem parameterized by the number of colors is known not to be in FPT. An FPT algorithm for $k-$coloring should have complexity in time $f(k)n^{O(1)}$, for $k = 3$, it would run in polynomial time in size of the input, but $3-$coloring problem is known as $NP-$hard. Thus, if graph coloring parameterized by the number of colors were in FPT, then $P = NP$.

The development of fixed-parameter algorithms for NP-complete problems is a way to get polynomial-time algorithms when some parameters are fixed [27, 36]. In many applications, the parameter is considered to be "small" compared to the total input size. Then it is challenging question to choose a proper parameter k and to find an algorithm which is exponential only in k, and not in the input size.

2.2.2 Fixed-parameter intractability

The way to show the fixed-parameter tractability of parameterized problems is to find fixed-parameter tractable algorithms and it is also very important to know how to prove that for some problems, fixed-parameter tractable algorithms do not exist. This can save us from wasting time on attacking the same problem over and over again with little hope of success. Besides, it is very helpful to look into a problem in both algorithm and complexity points of view at the same time. A failure on finding an algorithm can give a hint to the characteristics of the hard instances and the structure of hardness proofs. Conversely, if one can point out a flaw in a hardness proof, then it may suggest an algorithm idea of this problem. Hence a parameterized complexity theory helps not only in finding the lower bound of the complexity but also gives a direction to the algorithm of the questions.

2.2.2.1 Parameterized reductions

In $NP-$hardness proofs, polynomial-time reduction are used. A polynomial time reduction from decision problem A to decision problem B is a polynomial-time algorithm in which the input is any instance x of problem A and the output is a corresponding instance x' of problem B and the answer of problem x is "yes" if and only if the answer of problem x' is "yes". If there is a polynomial-time reduction from problem A to problem B and problem B is in P, then problem A

is also in P.

For parameterized problem, there is a parameterized reduction to transfer fixed-parameter tractability which is defined by Downey and Fellow [32, 33, 34].

Definition 2.2.2 (Parameterized reduction)**.** Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from A to B is an algorithm satisfies:

1. input is any instance (x, k) of A, output is an instance (x', k') of B,
2. the answer of problem (x, k) is "yes" if and only if the answer of problem (x', k') is "yes",
3. $k' \leq g(k)$ for some computable function g,
4. the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

Downey and Fellow [32, 33, 34] also proved that if there is a parameterized reduction from A to B and B is FPT, then A is also FPT.

2.2.2.2 The W-hierarchy

It is proven [35] that Independent set can be reduced to Dominating set, but it is unknown if there is a parameterized reduction in the other direction. Therefore, Downey and Fellow [32, 33, 34] have introduced the $W-$hierarchy to capture different complexity levels of hard parameterized problems.

In order to define $W-$hierarchy, we have to introduce the WEIGHTED CIRCUIT SATISFIABILITY (WCS) problem.

Definition 2.2.3 (Boolean circuit)**.** A Boolean circuit is a directed acyclic graph where the nodes are labeled in the following way:

1. every node of indegree 0 is an *input node*,
2. every node of indegree 1 is a *negation node*,
3. every node of indegree ≥ 2 is either an *and-node* or an *or-node*.

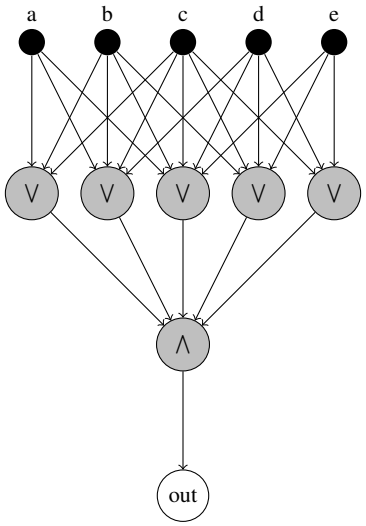


Figure 2.4: A depth-2, weft-2 circuit.

Additionally, exactly one of the nodes with outdegree 0 is labeled as the *output node*. The *depth* of a circuit is the maximum length of a path from an input node to the output node.

Assigning boolean values to the input nodes of a circuit will determine the value of the output node. If the output node has the value 1, then we say that the assignment of the input nodes *satisfies* the circuit. The *weight* of an assignment is the number of input nodes with value 1.

Definition 2.2.4 (WCS problem)**.** Given a boolean circuit C and an integer k, the WCS problem is to decide if C has a satisfying assignment of weight exactly k.

A *small node* in a circuit has indegree at most 2, a *large node* has indegree > 2. The *weft* of a circuit is the maximum number of large nodes on a path from an input node to the output node. Let $\mathcal{C}_{t,d}$ be the class of circuit with weft at most t and depth at most d. See example in Figure 2.4.

Definition 2.2.5 ($W-$hierarchy)**.** For $t \geq 1$, a parameterized problem L belongs to the class $W[t]$ if there is a parameterized reduction from L to WCS[$\mathcal{C}_{t,d}$] for some $d \geq 1$.

Mnich and van Bevern [70] surveyed main results on parameterized complexity for scheduling problems and identified 15 open problems. They identified width of precedence graph as a parameter for problems without communication delays, which is defined as the size of its largest

antichain. It is $W[2]$-hardness for problems like $P|prec, p_j = 1|C_{max}$, $P2|prec, p_j \in \{1, 2\}|C_{max}$, and $P3|prec, size_j \in \{1, 2\}|C_{max}$ considered by Bodlaender et al. [16] and van Bevern et al. [86]. Besides, it is proved by Bodlaender et al. [17] that even for chains of jobs with exact delays, on a single machine, parameterized by the number of chains, the problem is in XP by dynamic programming. For UET-UCT problem, an exact dynamic programming algorithm of time complexity $\mathcal{O}(2^{w(G)}.n^{2w(G)})$ was developed by Veltman [88] where $w(G)$ is the width of the precedence graph G. We can observe that it is also in XP.

2.3 Time windows and Pathwidth

A time window (r_i, d_i) associated to the task $i \in \mathcal{T}$ is given by a release date $r_i \in \mathbb{N}$ and a deadline $d_i \in \mathbb{N}$ such that the task i must be completed during the time interval (r_i, d_i), i.e. $t_i \geq r_i$ and $t_i + p_i \leq d_i$. We give the algorithms in Section 3.2.1, to any upper bound $\overline{C}$ of the minimum makespan, to calculate feasible release date r_i and deadline d_i for any task $i \in \mathcal{T}$ considering the precedence graph $\mathcal{G}$. For the problems to minimize the maximum lateness L_{max}, due date $\overline{d}_i$ for each task i is provided. With an upper bound of the maximum lateness $\overline{L}$, we can get the deadline $d_i = \overline{d}_i + \overline{L}$ for any task i. Then we can set a upper bound of minimum makespan $\overline{C} = \max_{i \in \{1,2,\ldots,n\}} \{d_i\} = \max_{i \in \{1,2,\ldots,n\}} \{\overline{d}_i + \overline{L}\}$.

With time windows (r_i, d_i) for all tasks, we can create an interval graph. An interval graph of a set of interval is an undirected graph with one vertex per interval and an edge between vertices whose intervals overlap. The formal definition is following.

Definition 2.3.1. An interval graph is an undirected graph I formed from a family of intervals

$$(r_i, d_i), \forall i \in \{1, 2, \ldots, n\}$$

by creating one vertex v_i for each interval (r_i, d_i), and connecting two vertices v_i and v_j by an edge whenever the corresponding two sets have a nonempty intersection. That is, the edge set of I is

$$E(I) = \{\{v_i, v_j\}, (r_i, d_i) \cap (r_j, d_j) \neq \emptyset\}$$

It is the intersection graph of the intervals.

For example, we can associate time windows to the precedence graph of Figure 2.3a as the follow Figure 2.5. The corresponding interval graph is as shown in Figure 2.6.

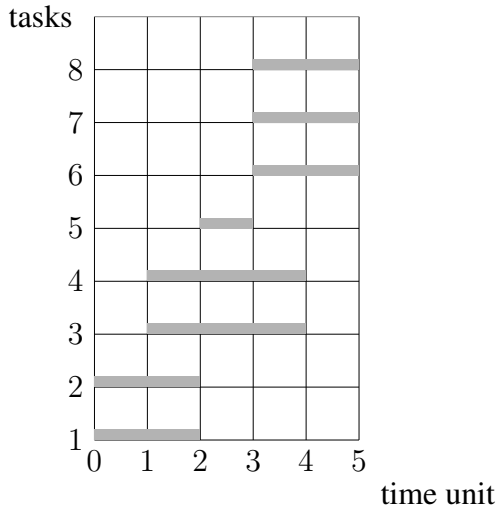


Figure 2.5: Time windows of precedence graph in Figure 2.3a.

In this thesis, we consider the pathwidth of interval graph of time windows as the parameter for fixed parameter algorithm. To understand what is pathwidth, we introduce path decomposition as follows. Path decomposition of a graph G could be interpreted as a method to represent the graph G as a path, and the pathwidth of graph G is a measurement of the thickness of the path formed

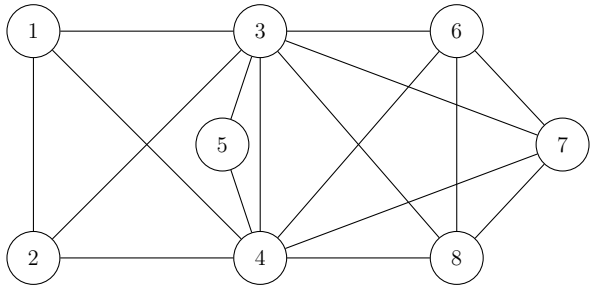


Figure 2.6: The interval graph of time windows in Figure 2.5.

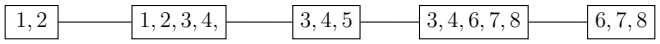


Figure 2.7: A path-decomposition of Figure 2.6.

from G. More formally, a path decomposition of a graph G is a path, each vertex of the path is a subset of vertices of G such that the endpoints of each edge are in the same subsets and each vertex appears in a contiguous part of the path. The definition of path decomposition was introduced by Robertson and Seymour [79] in their series of papers on graph minors.

Definition 2.3.2. A path decomposition of a graph $G = (V, E)$ to be an ordered sequence $P = (X_1, X_2, \ldots, X_n)$ where $X_i \subseteq V, \forall i \in \{1, 2, \ldots, n\}$, which satisfies the following two properties:

1. $\bigcup_{i=1}^{n} X_i = V$, meaning every vertex of G is in at least one subset X_i.
2. For each edge of G, there exists a subset X_i such that both endpoints of the edge are inside X_i.
3. For every three indices such that $i \leq j \leq k$, we have $X_i \cap X_k \subseteq X_j$.

The width of a path-decomposition is defined as $\max_{i \in \{1,2,\ldots,n\}} |X_i| - 1$, and the pathwidth of G is the minimum width of any path-decomposition of G.

Figure 2.7 is an example of a path-decomposition of the interval graph of Figure 2.6.

The reason of the subtraction of one from the maximum size of X_i is to have the pathwidth of a path graph equal to one. It is proven by Bodlaender [15] that pathwidth of graph G can be described in many equivalent ways such as interval thickness. Interval thickness is one less than the maximum clique size of an interval graph which contains graph G. In an interval graph, the interval thickness is equal to the maximum clique size minu.s one. For example, the interval graph in Figure 2.6 has a maximum clique with nodes $\{3, 4, 6, 7, 8\}$ with size of 5, so the interval thickness as well as the pathwidth of this graph is 4.

Pathwidth, and graphs of bounded pathwidth, also have applications in VLSI (Very Large Scale Integration) design [39, 68, 74], graph drawing [54, 83], and compiler design [14]. Many problems

in graph algorithms may be solved efficiently on graphs of bounded pathwidth, by using dynamic programming on a path-decomposition of the graph, see survey [6].

In this thesis, we consider the pathwidth of the interval graph of time windows of tasks. This parameter measures the maximum number of tasks that can possibly be executed at the same time instance. With release time r_i and deadline d_i for all task in $\mathcal{T}$, for all $\alpha \in \{0, 1, 2, \ldots, \overline{C} - 1\}$, we can define

$$X_\alpha = \{i \in \mathcal{T}, (\alpha, \alpha + 1) \cap (r_i, d_i) \neq \emptyset\}.$$

The pathwidth we consider is equal to the maximum size of X_α minus 1.

Since the deadlines of tasks are calculated with an upper bound of makespan $\overline{C}$ or an upper bound of maximum lateness $\overline{L}$, we note pathwidth as $pw(\overline{C})$ or $pw(\overline{L}) = \max_{\alpha \in \{0,\ldots,\overline{C}-1\}} (|X_\alpha| - 1)$.

2.4 Treewidth and Courcelle's theory

Similarly to pathwidth, treewidth measures how far the structure of a graph is from a tree-like structure. The smaller treewidth is, the better the graph can form a tree decomposition. Formally, a tree decomposition of a graph $G = (V(G), E(G))$ is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where $T = (V(T), E(T))$ is a tree whose every node $t \in V(T)$ is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following conditions are satisfied:

1. $\bigcup_{t \in V(T)} X_t = V(G)$. That is, each graph vertex is associated with at least one tree node.
2. For every edge $(u, v) \in E(G)$, there exist a node $t \in V(T)$ such that X_t contains both u and v.
3. For every node u in $V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subtree of T.

The width of a tree decomposition $\mathcal{T} = \{(T, \{X_t\}_{t \in V(T)})\}$ equals $\max_{t \in V(T)} |X_t| - 1$. The treewidth of a graph G denoted by $tw(G)$ is the minimum possible width of a tree decomposition of G. The treewidth of trees and forests is one. For interval graph, treewidth is equal to pathwidth, which is the maximum clique size minus one.

Many optimization problems can be solved by dynamic programming on a tree decomposition and are fixed-parameter tractable when parameterized by the treewidth such as Weighted independent set problem and dominating set problems. Courcelle's Theorem describes unified properties of problems which can be solved by dynamic programming over a tree decomposition and it is presented by a logical formalism called *Monadic Second-order logic on graphs (MSO_2)*. Courcelle's Theorem indicates that problems expressible in this formalism are always fixed-parameter tractable when parameterized by treewidth. To formally state Courcelle's Theorem, we will introduce Monadic second-order logic on graphs first.

2.4.1 Monadic second-order logic for graphs

A graph can be described in a logical structure. For every graph $G = (V_G, edge_G)$, let $\lfloor G \rfloor = \langle V_G, edge_G \rangle$ be a relational structure, where V_G is the domain, the set of vertex, and $edge_G \subseteq V_G \times V_G$ such that $(x, y) \in edge_G$ if and only if there exist an edge from x to y if G is directed graph and an edge between x and y if G is undirected.

An example of an undirected complete graph of n nodes K_n can be represented as follows:

$$\lfloor K_n \rfloor := \langle [n], edge_n \rangle,$$

$$edge_n(x, y) :\Leftrightarrow x, y \in [n] \text{ and } x \neq y.$$

Besides, properties of a graph G can be expressed by relational structures in $\lfloor G \rfloor$ too. For example, if G is directed, then every vertex of G has at least one predecessor and at least one successor if and only if

$$\lfloor G \rfloor \models \forall x \exists y, z(edge(y, x) \wedge edge(x, z)).$$

The two examples above are first-order logic formulas because the variables are vertices. Second-order logic allows quantification over arbitrary predicates, i.e. sets of logic sentences, as variables. Monadic second-order logic formulas only allows quantification over monadic pred-

icates, i.e. sets of vertices, as variables. Uppercase variables note sets of vertices and lowercase variables note individual vertices. An example of monadic second-order logic formula are following:

$$\lfloor G \rfloor \models \exists X(\exists x, x \in X \wedge \exists y, y \notin X \wedge \forall x, y(edge(x, y) \rightarrow (x \in X \Leftrightarrow y \in X)))$$

The formula above holds if and only if G is not connected.

Variables whose evaluations are given with graph G are called *free variables*. Such variables are also allowed in MSO_2 formulas and we assume that the evaluation of these free variables are provided with the graph for the evaluation of MSO_2 formulas. For example the formula follows:

$$\begin{aligned} partition(X, Y, Z) =& \forall x\{(x \in X \vee x \in Y \vee x \in Z) \wedge \\ & [\neg(x \in X \wedge x \in Y) \wedge \neg(x \in Y \wedge x \in Z) \wedge \neg(x \in X \wedge x \in Z)]\} \end{aligned}$$

The formula above has three free variables X, Y, Z and verifies that (X, Y, Z) is a partition of vertex set V_G.

Monadic second-order logic has many properties and applications, we have introduced the basic definitions above. Courcelle's Theorem identifies fixed parameter tractable problems when parameterized by treewidth and it is presented by MSO_2. For any MSO_2 formula ϕ, let $||\phi||$ be the length of encoding of ϕ as a string.

Theorem 2.4.1 (Courcelle's Theorem [26])**.** *Assume that ϕ is a formula of MSO_2 and G is an n-vertex graph equipped with evaluation of all the free variables of ϕ. Suppose, moreover, that a tree decomposition of G of width t is provided. Then there exists an algorithm that verifies whether ϕ is satisfied in G in time $f(||\phi||, t) \cdot n$, for some computable function f.*

Note that the theorem above states that a decision problem modeled with MSO_2 is a fixed parameter tractable when the parameter is treewidth. However, the scheduling problems discussed

in this thesis are optimization problems, we would have the following optimization variant of the theorem.

Theorem 2.4.2 ([7]). *let ϕ be an MSO_2 formula with p free monadic variables $X_1, \ldots, X_p$, and let $\alpha(x_1, \ldots, x_p)$ be an affine function. Assume that we are given an $n-$vertex graph G together with its tree decomposition of width t, and suppose G is equipped with evaluation of all the free variables of ϕ apart from $X_1, X_2, \ldots, X_p$. Then there exists an algorithm that in $f(||\phi||, t) \cdot n$ finds the minimum or the maximum value of $\alpha(|X_1|, \ldots, |X_p|)$ for sets $X_1, \ldots, X_p$ for which $\phi(X_1, \ldots, X_p)$ is true, where f is some computable function.*

In the next subsection, we will introduce how we can apply it to our scheduling problems.

2.4.2 Courcelle's Theorem's applications to UET-UCT problem

To model UET-UCT problem on unlimited number of machines in MSO_2 formula, we have a precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$. The free variables are the subsets of tasks, $X_1, \ldots, X_{\overline{C}}$, where $\overline{C}$ is an upper bound of makespan. The formula $\sigma(X_1, \ldots, X_{\overline{C}})$ is satisfied if and only if there exists a feasible schedule of $\mathcal{G}$ with tasks in X_α executed at time α.

$$
\begin{aligned}
\sigma(X_1, \ldots, X_{\overline{C}}) =& partition(X_1, \ldots, X_{\overline{C}}) \wedge \phi_1(X_1, \ldots, X_{\overline{C}}) \\
& \wedge \phi_2(X_1, \ldots, X_{\overline{C}}) \wedge \phi_3(X_1, \ldots, X_{\overline{C}}) \\
\phi_1(X_1, \ldots, X_{\overline{C}}) =& \forall u, v \in \mathcal{T}, (edge(u, v) \wedge u \in X_\alpha \wedge v \in X_\beta \rightarrow \beta - \alpha \leq 1) \\
\phi_2(X_1, \ldots, X_{\overline{C}}) =& \forall u \in \mathcal{T}, (\exists v, edge(v, u) \wedge u \in X_\alpha \wedge v \in X_\beta \wedge \alpha - \beta = 1 \\
& \rightarrow \forall w \in \mathcal{T}, w \neq v \wedge edge(w, u) \wedge w \in X_\theta \wedge \alpha - \theta > 1) \\
\phi_3(X_1, \ldots, X_{\overline{C}}) =& \forall u \in \mathcal{T}, (\exists v, edge(u, v) \wedge u \in X_\alpha \wedge v \in X_\beta \wedge \alpha - \beta = -1 \\
& \rightarrow \forall w \in \mathcal{T}, w \neq v \wedge edge(u, w) \wedge w \in X_\theta \wedge \alpha - \theta < -1)
\end{aligned}
$$

$partition(X_1, \ldots, X_{\overline{C}})$ makes sure that all the tasks in $\mathcal{G}$ are scheduled and only scheduled once. $\phi_1(X_1, \ldots, X_{\overline{C}}), \phi_2(X_1, \ldots, X_{\overline{C}}), \phi_3(X_1, \ldots, X_{\overline{C}})$ are corresponding to the constraints $(1), (3), (4)$

in the integer programming (P). According to the theorem 2.4.1, this formula shows that the decision problem if a schedule is a feasible UET-UCT problem is fixed-parameter tractable. According to theorem 2.4.2, if there exist an affine function $\alpha(|X_1|, \ldots, |X_{\overline{C}}|)$ that values the makespan of a schedule, then the optimization problem to minimize makespan is fixed-parameter tractable when parameterized with treewidth.

Let us define

$$\alpha(x_1, \ldots, x_p) = p_1 x_1 + p_2 x_2 + \ldots p_p x_p,$$

where

$$\begin{aligned} p_1 &= 1, \\ p_i &= n \cdot (p_1 + \cdots + p_{i-1}) + 1, \forall i \in \{2, \ldots, p\}, \end{aligned}$$

and n is the number of tasks.

Let's prove that for any two schedule $\sigma^1 = (X_1^1, \ldots, X_{\overline{C}}^1)$ and $\sigma^2 = (X_1^2, \ldots, X_{\overline{C}}^2)$, if the makespan of σ^1 is C_1, the makespan of σ^2 is C_2 and $C_1 < C_2$, then $\alpha(|X_1^1|, \ldots, |X_{\overline{C}}^1|) < \alpha(|X_1^2|, \ldots, |X_{\overline{C}}^2|)$. Indeed we have

$$\begin{aligned} &\alpha(|X_1^1|, \ldots, |X_{\overline{C}}^1|) - \alpha(|X_1^2|, \ldots, |X_{\overline{C}}^2|) \\ &= p_1(|X_1^1| - |X_1^2|) + p_2(|X_2^1| - |X_2^2|) \ldots p_{C_1}(|X_{C_1}^1| - |X_{C_1}^2|) - (p_{C_1+1} X_{C_1+1}^2 + \cdots + p_{C_2} X_{C_2}^2) \\ &\leq n(p_1 + \ldots p_{C_1}) - (p_{C_1+1} X_{C_1+1}^2 + \cdots + p_{C_2} X_{C_2}^2) \\ &\leq n(p_1 + \ldots p_{C_1}) - p_{C_1+1} \\ &\leq 0 \end{aligned}$$

Therefore, we have that the problem to optimize the makespan of UET-UCT schedule on unlimited number of machines is fixed-parameter tractable when parameterized by treewidth of precedence graph.

3. Kernelization of UET-UCT model

3.1 Introduction to kernelization

Kernelization is one of the basic techniques to form a fixed-parameter tractable algorithm. Kernelization is often a set of actions which cut away parts of the input that are easy to handle, and the left part is called the kernel. There are two main notations to express the kernelization process, listed as follows.

Definition 3.1.1 (Downey–Fellows notation [35])**.** A kernelization for a parameterized problem $L \subseteq \Sigma^* \times N$ is an algorithm that takes an instance $(x, k) \in L$ and maps it in time polynomial in $|x|$ and k to an instance (x', k') such that

- (x, k) is in L if and only if (x', k') is in L,
- the size of x' is bounded by a computable function f in k
- k' is bounded by a function in k.

The output (x', k') of kernelization is called a kernel.

Definition 3.1.2 (Flum–Grohe notation [40])**.** A kernelization for a parameterized problem L is an algorithm that takes an instance x with parameter k and maps it in polynomial time to an instance y such that

x is in L if and only if y is in L and the size of y is bounded by a computable function f in k. Note that in this notation, the bound on the size of y implies that the parameter of y is also bounded by a function in k.

The function f is often referred to as the size of the kernel. If $f = k^{O(1)}$, it is said that L admits a polynomial kernel. Similarly, for $f = O(k)$, the problem admits linear kernel.

It is proven that a problem is fixed-parameter tractable if and only if it is kernelizable and decidable [27]. An example of kernelization algorithm is the kernelization of $k-$vertex cover problem by Buss Jonathan F. and Goldsmith Judy [18]. Given a graph $G = (V, E)$, $k-$vertex cover problem is to decide if there is a vertex cover of size at most k. The rules for kernelization are as follows:

1. If there are more than k vertices of degree more than k, then reject.

2. Let U be the set of vertices of degree more than k and G' be a subgraph of G induced by $V - U$. If there are more than $k(k - |U|)$ edges in G', then reject.

In this chapter, we present the kernelizations for UET-UCT model. We have two main steps for kernelization. In Section 3.2, we will introduce algorithms to tighten time windows of tasks. In Section 3.3, we will introduce active schedules and its necessary conditions. We will use these two step of kernelization to form our main FPT-algorithm in the following chapters.

3.2 Modification on release date and deadline

We consider pathwidth of the interval graph of time windows of tasks as the parameter of our FPT algorithm, so to get short time windows for every tasks is essential. We introduce in this section a method to improve the release dates and deadlines of tasks. Garey and Johnson [42] gave a deadline modification algorithm (GJ algorithm in short) for problem $P2|prec, p_i = 1, r_i|*$ in time complexity $O(n^3)$. This algorithm has been extended by Hanen and Zinder [51] to arbitrary number of processors problem $P|prec, p_i = 1, r_i|L_{max}$ and they also analysed that the worst case ratio tends to 2 when the number of processors goes to infinity. Leung, Palem and Pnueli (in short LPP) algorithms [64] is also a deadline modification algorithm to give feasible schedules of problems in the presence of precedence constraints, unit execution tasks, release-times, deadlines, and fixed delays between tasks. Carlier, Hanen and Munier Kordon [19] have proved that GJ algorithm and LPP algorithm reach the same fixed point of deadlines. Hanen, Munier Kordon and Pedersen [50] extend both GJ algorithm and LPP algorithm to problems with arbitrary execution

duration tasks. Zinder et al. [92] have proposed a release date modification algorithm as the base of their branch and bound algorithm on UET-UCT problems.

In this section, we will give release date and deadline modification algorithms on limited and unlimited number of identical parallel processors for UET-UCT models. For unlimited number of processors, we give recursive algorithms; for limited number of processors, we give an extension of the algorithm given by Zinder et al. [92].

3.2.1 Modification algorithm on release dates

Let's recall the problem UET-UCT we considered. Let $\mathcal{T} = \{1, 2, \ldots, n\}$ be a set of tasks, graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ be the precedence graph of $\mathcal{T}$. There are m identical parallel machines to execute these tasks. If there is an arc (i, j) between task i and j and they are executed on executed on different machines, there will be a communication delay added between their executions. The execution time of each task is one unit, and the communication time between tasks is also one unit.

When there are unlimited number of processors, i.e. $m \geq n$, the release dates can be calculated as follows:

First, for tasks that have no predecessors, i.e. $\Gamma^-(i) = \emptyset$, we set $r_i = 0$.

For tasks that have predecessors, let the set of predecessors of task i be $\Gamma^-(i) = \{j_i, \ldots, j_p\}$. To calculate the release date of i, we number the predecessors $j_1, \ldots, j_p$ in decreasing order of the release dates, i.e. $r_{j_1} \geq r_{j_2} \geq \cdots \geq r_{j_p}$. Then, the release date r_i can be calculated recursively as follows:

$$r_i = \begin{cases} r_{j_1} + 1 & \text{if } |\Gamma^-(i)| = 1 \text{ or } (|\Gamma^-(i)| > 1 \text{ and } r_{j_1} > r_{j_2}) \\ r_{j_1} + 2 & \text{if } |\Gamma^-(i)| > 1 \text{ and } r_{j_1} = r_{j_2}. \end{cases} \tag{3.2.1}$$

If the number of machine $m < n$, the recursive algorithm for release date can be improved by a algorithm proposed by Yakov Zinder et al. [92].It is as follows:

For any task $i \in \mathcal{T}$, if $\forall j \in \Gamma^-(i)$, then set $\bar{r} = \max\limits_{j \in \Gamma^-(i)} r_j$,

$$r_i = \max_{0 \leq \tau \leq \bar{r}} \left\{ \tau + \left\lceil \frac{|\{j \in \Gamma^-(i) : r_j \geq \tau\}| - 1}{m} \right\rceil + 1 \right\} \tag{3.2.2}$$

The following lemma provides the feasibility of the equation 3.2.2.

Lemma 3.2.1. *Any schedule $\sigma(\mathcal{G})$ feasible for the precedence graph $\mathcal{G}$ is also feasible with the release dates r_i provided by equation 3.2.2.*

Proof. Let $\sigma(\mathcal{G})$ be a feasible schedule of precedence graph $\mathcal{G}$, the starting time of task $i \in \mathcal{T}$ is t_i^σ. We need to prove that $t_i^\sigma \geq r_i$ provided by equation 3.2.2.

For $i \in \mathcal{T}$ without predecessors, $r_i = 0$. Thus $t_i^\sigma \geq r_i$. For $i \in \mathcal{T}$ with $\Gamma^-(i) \neq \emptyset$, for all $j \in \Gamma^-(i)$, we assume that $t_j^\sigma \geq r_j$.

For any $\tau \in [0, \overline{r}]$, there exist one and only one $k \in \{-1, 0, 1, \dots\}$, such that $|\{j \in \Gamma^-(i) : r_j \geq \tau\}| - 1 \in (km, (k+1)m]$, then there is at least one predecessor of i who has to be scheduled after time $\tau + k + 1$, since there are m machines available. Thus we have $t_i^\sigma \geq \tau + k + 2$, which is equal to $t_i^\sigma \geq \max\limits_{0 \leq \tau \leq \overline{r}} \left\{\tau + \left\lceil \frac{|\{j \in \Gamma^-(i) : r_j \geq \tau\}| - 1}{m} \right\rceil + 1\right\}$. Thus we have $t_i^\sigma \geq r_i$. □

3.2.2 Modification algorithms on deadlines

For the problem to minimize the makespan C_{max}, we also need to calculate the deadlines. We assume that the release dates are given.

When we have unlimited number of machines, i.e. $m \leq n$, to calculate the deadline of task i, first, for tasks that have no predecessors, i.e. $\Gamma^-(i) = \emptyset$, we set $d_i = r_i + 1$.

Then, for the tasks that have predecessors, i.e. $\Gamma^-(i) \neq \emptyset$, we set $d_i = \max\{\max\limits_{j \in \Gamma^-(i)} d_j + 2, r_i + 1\}$.

When we have limited number of machines, i.e. $m < n$, the algorithm above will not work. An upper bound of makespan $\overline{C}$ is need to be able to calculate the deadlines. Thus we give the algorithm below:

First, for all tasks that have no successors, i.e. $\Gamma^+(i) = \emptyset$, we set $d_i = \overline{C}$.

Then the tasks with successors, i.e. $\Gamma^+(i) \neq \emptyset$, let $\overline{d} = \min\limits_{j \in \Gamma^+(i)} d_j$, we can calculate the deadlines recursively as follows:

$$d_i = \min_{\overline{d} \leq \tau \leq \overline{C}} \left\{ \tau - \left\lceil \frac{|\{j : j \in \Gamma^+(i), d_j \leq \tau\}| - 1}{m} \right\rceil - 1 \right\} \tag{3.2.3}$$

The following lemma provides the feasibility of the equation 3.2.3.

Lemma 3.2.2. *Any feasible schedule σ with makespan bounded by $\overline{C}$ for the precedence graph $\mathcal{G}$ is also feasible for the deadlines d_i provided by equation 3.2.3.*

Proof. Let σ be a feasible schedule of precedence graph $\mathcal{G}$ with makespan bounded by $\overline{C}$, the starting time of task $i \in \mathcal{T}$ is t_i^σ. We need to prove that $t_i^\sigma \leq d_i - 1$ provided by equation 3.2.3.

For $i \in \mathcal{T}$ without successors, $d_i = \overline{C}$. Thus $t_i^\sigma \leq d_i - 1$. For $i \in \mathcal{T}$ with $\Gamma^+(i) \neq \emptyset$, for all $j \in \Gamma^+(i)$, we assume that $t_j^\sigma \leq d_j$.

For any $\tau \in [\overline{d}, \overline{C}]$, there exist one and only one $k \in \{-1, 0, 1, \dots\}$, such that $|\{j \in \Gamma^+(i) : d_j \leq \tau\}| - 1 \in (km, (k+1)m]$, so there is at least one successor of i who has to be scheduled before time $\tau - k - 1$, when there are m machines available.

Thus we have $t_i^\sigma \leq \tau - k - 3$, which is equal to $t_i^\sigma \leq \min\limits_{\overline{d} \leq \tau \leq \overline{C}} \left\{\tau - \left\lceil \frac{|\{j \in \Gamma^+(i) : d_j \leq \tau\}| - 1}{m} \right\rceil - 1\right\} - 1$. Thus we have $t_i^\sigma \leq d_i - 1$. □

3.2.3 A necessary condition on feasible schedules

For any value $\alpha \in \{0, \dots, \overline{C} - 1\}$, we recall that X_α is the set of tasks that can be scheduled at time α following release times and deadlines,

$$X_\alpha = \{i \in \mathcal{T}, r_i \leq \alpha, \alpha + 1 \leq d_i\}.$$

We also denote by Z_α the set of tasks than must be completed at or before time $\alpha + 1$,

$$Z_\alpha = \{i \in \mathcal{T}, d_i \leq \alpha + 1\}.$$

Figure 3.1a shows the release time and deadline of tasks from the graph presented by Figure 2.3a with the upper bound of the makespan $\overline{C} = 6$. Figure 3.1b shows the associated sets X_α and Z_α for $\alpha \in \{0, 1, \dots, 5\}$. For this example, $pw(\overline{C}) = |X_3| - 1 = 5$.

Let us consider that $\sigma(\mathcal{G})$ is a feasible schedule of makespan $C \leq \overline{C}$. For every integer $\alpha \in \{0, \dots, C - 1\}$, we set $\mathcal{T}_\alpha^\sigma = \{i \in \mathcal{T}, t_i^\sigma = \alpha\}$.

tasks	1	2	3	4	5	6	7	8
r_i	0	0	1	1	2	3	3	3
d_i	3	3	5	5	4	6	6	6

(a) Release times r_i and deadlines d_i of all tasks from Figure 2.3a for $\overline{C} = 6$.

α	X_α	Z_α
0	$\{1, 2\}$	$\emptyset$
1	$\{1, 2, 3, 4\}$	$\emptyset$
2	$\{1, 2, 3, 4, 5\}$	$\{1, 2\}$
3	$\{3, 4, 5, 6, 7, 8\}$	$\{1, 2, 5\}$
4	$\{3, 4, 6, 7, 8\}$	$\{1, 2, 3, 4, 5\}$
5	$\{6, 7, 8\}$	$\{1, 2, 3, 4, 5, 6, 7, 8\}$

(b) Sets X_α and Z_α, $\alpha \in \{0, \ldots, 5\}$.

Figure 3.1: Release times, deadlines, and sets X_α and Z_α, $\alpha \in \{0, \ldots, 5\}$ for the instance presented by Figure 2.3a and $\overline{C} = 6$ on unlimited number of machines.

The following lemma will be considered further to reduce the size of the tasks sets built at each step of our algorithm.

Lemma 3.2.3. *Let $\sigma(\mathcal{G})$ be a feasible schedule of $\mathcal{G}$. For any $\alpha \in \{0, \ldots, C-1\}$, $\bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$.*

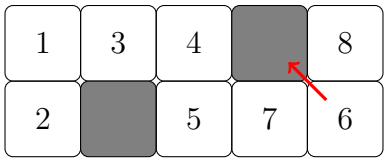
Proof. Since $\sigma(\mathcal{G})$ is feasible, for any $\alpha \in \{0, \ldots, C-1\}$, $\mathcal{T}_\alpha^\sigma \subseteq X_\alpha$ and thus, $\forall i \in \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma$, $r_i \leq \alpha$. Moreover, each task $i \notin Z_\alpha$ satisfies $d_i \geq \alpha + 2$.

Thus, for any task $i \in \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma - Z_\alpha$, $[\alpha, \alpha+2] \subseteq [r_i, d_i]$. Therefore $\bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$, and the lemma is proved. □

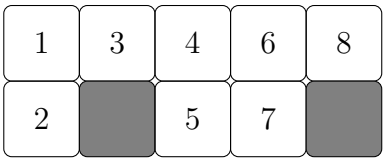
For our example presented by Figure 3.1 and $\alpha = 3$, we have $Z_3 = \{1, 2, 5\}$ and $X_3 \cap X_4 = \{3, 4, 6, 7, 8\}$. For the schedule showed in Figure 2.3, we have $\bigcup_{\beta=0}^{3} \mathcal{T}_\beta^\sigma = \{1, 2, 3, 4, 5, 6, 7\}$. We observe that $\bigcup_{\beta=0}^{3} \mathcal{T}_\beta^\sigma - Z_3 = \{3, 4, 6, 7\} \subseteq X_3 \cap X_4$.

3.3 Active schedules

The definition of active schedule is proposed by Giffler, B. and Thompson, G.L. [43]. A feasible schedule is active if no task can start earlier without increasing the start time of other tasks. Rayward-Smith[78] showed that an active schedule of UET-UCT problem is no longer than $3 - 2/m$ times the optimum. Active schedules are always considered to reduce the size of the



(a) An optimal non-active schedule of Figure 2.3a.



(b) An optimal active schedule of Figure 2.3a.

Figure 3.2: Example of active and non-active schedules.

solution space [88, 92]. In this section, we listed two necessary conditions for active schedules which can be applied to the algorithms.

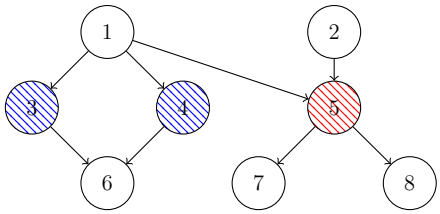
As shown in Figure 3.2a, task 6 can be scheduled in advance at time 3 instead of time 4 without increasing the starting time of other task, so it is not an active schedule. For schedule shown in Figure 3.2b, no task can start earlier without increasing the start time of other task and keep the schedule feasible, so it is an active schedule.

3.3.1 Preferred sons

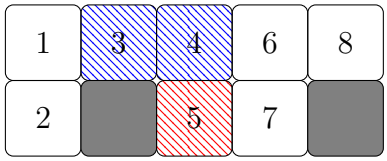
For a task $i \in \mathcal{T}$, there is at most one successor of i can be scheduled at time $t_i + 1$, right after i, and on the same machine of i. There can be many candidates for this position, we call these candidates the preferred sons of task i. The following is the formal definition and properties of preferred sons.

Let us consider that $\sigma(\mathcal{G})$ is a feasible schedule of makespan C. For every integer $\alpha \in \{0, \ldots, C-1\}$, we set $W_\alpha = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma$ and $B_\alpha = \mathcal{T}_\alpha^\sigma$. The set W_α contains all the tasks that are executed in time $[0, \alpha+1)$, and B_α contains all the tasks that are executed at time α. We show hereafter that, if a task $i \in \mathcal{T}$ has one or more successors schedulable at time $t_i^\sigma + 1$ (called the preferred sons of i), then we can impose that at most one of them is executed at time $t_i^\sigma + 1$ on the same processor as the task i if there are processors available.

Definition 3.3.1 (Preferred sons)**.** Let $\sigma(\mathcal{G})$ be a feasible schedule of makespan C. For every integer $\alpha \in \{0, \ldots, C-1\}$, a task $j \in \mathcal{T}$ is a **preferred son** of a task $i \in B_\alpha$ if j is a successor of i that is schedulable at time $\alpha + 1$. The set of the **preferred sons** of i with respect to W_α and B_α is defined as $PS_{W_\alpha, B_\alpha}(i) = \{j \in \Gamma^+(i), \Gamma^-(j) \subseteq W_\alpha \text{ and } \Gamma^-(j) \cap B_\alpha = \{i\}\}$.



(a) A precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$.



(b) An optimum schedule for the precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ of Figure2.3a.

Figure 3.3: Task 3 and task 4 are preferred sons of task 1, task 5 is not.

Definition 3.3.2 (Preferred sons property for unlimited resources)**.** A feasible schedule $\sigma(\mathcal{G})$ satisfies the **preferred sons property for unlimited resources** if, for every integer $\alpha \in \{0, \ldots, C-1\}$, each task $i \in B_\alpha$ such that $PS_{W_\alpha, B_\alpha}(i) \neq \emptyset$ has exactly one preferred son executed at time $\alpha + 1$.

Property 3.3.3 (Preferred sons property for m processors)**.** A feasible schedule $\sigma(\mathcal{G})$ satisfies the **preferred sons property for limited resources** if, for every integer $\alpha \in \{0, \ldots, C-1\}$, each task $i \in B_\alpha$ such that $PS_{W_\alpha, B_\alpha}(i) \neq \emptyset$ has exactly one preferred son executed at time $\alpha + 1$ if $|B_{\alpha+1}| < m$.

The condition $|B_{\alpha+1}| < m$ is necessary in Preferred sons property for m processors. When $|B_{\alpha+1}| = m$, it is possible that the schedule is active and no preferred sons of $i \in B_\alpha$ are executed at time $\alpha + 1$ because there are no processors available.

Let us consider the precedence graph and the feasible schedule presented by Figure 2.3. The preferred sons' relations are presented in Figure 3.3. For $\alpha = 0$, $W_0 = B_0 = \{1, 2\}$, $PS_{W_0, B_0}(1) = \{3, 4\}$ and $PS_{W_0, B_0}(2) = \emptyset$. Thus, we can enforce that exactly one task in $\{3, 4\}$ would be executed at time 1.

Lemma 3.3.4. *Let $\sigma(\mathcal{G})$ be a feasible schedule. There exists a corresponding feasible schedule $\sigma'(\mathcal{G})$ that satisfies the preferred sons property for m processors and such that for any task $i \in \mathcal{T}$, $t_i^{\sigma'} \leq t_i^{\sigma}$.*

Proof. Let us suppose that $\sigma(\mathcal{G})$ does not verify the preferred sons property. Let $(t_1^\sigma, t_2^\sigma, \ldots, t_n^\sigma)$ be the time vector and x^σ be the delay signal vector. Let $\alpha \in \{0, \ldots, C-1\}$ be the first time instant

at which the property is not fulfilled, and $i^\star \in B_\alpha$ is a corresponding task with $PS_{W_\alpha,B_\alpha}(i^\star) \neq \emptyset$ and $t^\sigma_{i^\star} = \alpha$. Then we have $|B_{\alpha+1}| < m$ and all the preferred sons of $i^\star$, i.e. $j \in PS_{W_\alpha,B_\alpha}(i^\star)$, are scheduled after time $\alpha + 1$, i.e. $t^\sigma_j \geq \alpha + 2$

For any task $j \in PS_{W_\alpha,B_\alpha}(i^\star)$, its predecessors are in W_α, thus $\forall k \in \Gamma^-(j), t^\sigma_k \leq \alpha$. According to the definition of communication delays, for every task $j \in \Gamma^+(i^\star)$, $x^\sigma_{i^\star j} = 1$.

We build another feasible schedule $\sigma'(\mathcal{G})$ as follows:

1. Choose a task $j^\star \in PS_{W_\alpha,B_\alpha}(i^\star)$. We then set $t^{\sigma'}_{j^\star} = \alpha + 1$
2. Keep other tasks' execution time, i.e. $t^{\sigma'}_i = t^\sigma_i, \forall i \in \mathcal{T} - \{j^\star\}$

For every task $i \in \mathcal{T}$, $t^{\sigma'}_i \leq t^\sigma_i$. Now, we get

$$\sum_{\ell \in \Gamma^+(i^\star)} x^{\sigma'}_{i^\star \ell} = \sum_{\ell \in \Gamma^+(i^\star) - \{j^\star\}} x^{\sigma'}_{i\ell} + x^{\sigma'}_{i^\star j^\star} = |\Gamma^+(i^\star)| - 1.$$

Similarly, we get

$$\sum_{\ell \in \Gamma^-(j^\star)} x^{\sigma'}_{\ell j^\star} = \sum_{\ell \in \Gamma^-(j^\star) - \{i^\star\}} x^{\sigma'}_{\ell j^\star} + x^{\sigma'}_{i^\star j^\star} = |\Gamma^-(j^\star)| - 1,$$

Besides, $|B_\alpha| \leq m, \forall \alpha \in \{0, 1, 2, \ldots, C-1\}$. Thus $\sigma'(\mathcal{G})$ is feasible.

This transformation can be continued on the new schedule σ' until we obtain a schedule that satisfies the preferred sons' property. Each time instance can be is considered only once and thus this transformation is done at most $\overline{C}$ times. Thus the lemma holds. □

Lemma 3.3.5. *Let $\sigma(\mathcal{G})$ be a feasible schedule. There exists a corresponding feasible schedule $\sigma'(\mathcal{G})$ that satisfies the preferred sons property and such that for any task $i \in \mathcal{T}$, $t^{\sigma'}_i \leq t^\sigma_i$.*

Proof. We can suppose without loss of generality that tasks are scheduled by $\sigma(\mathcal{G})$ as soon as possible following communication delay vector x^σ of $\sigma(\mathcal{G})$, i.e. $\forall i \in \mathcal{T}$ and $\Gamma^-(i) \neq \emptyset$, $t^\sigma_i = \max_{j \in \Gamma^-(i)} (t^\sigma_j + 1 + x^\sigma_{ji})$.

Let us suppose that $\sigma(\mathcal{G})$ does not verify the preferred sons property. Let then $\alpha \in \{0, \ldots, C-1\}$ be the first instant for which the property is not fulfilled, and $i^\star \in B_\alpha$ a corresponding task with $PS_{W_\alpha,B_\alpha}(i^\star) \neq \emptyset$. We show that, for every task $j \in \Gamma^+(i^\star)$, $x^\sigma_{i^\star j} = 1$.

- Since $i^\star$ is performed at time α, $i^\star$ cannot have two successors scheduled at time $\alpha+1$. So, every task $j \in PS_{W_\alpha,B_\alpha}(i^\star)$ satisfies $t^\sigma_j \geq t^\sigma_{i^\star} + 2$ and by coherence of $\sigma(\mathcal{G})$, $x^\sigma_{i^\star j} = 1$.
- Now, any task $j \in \Gamma^+(i^\star) - PS_{W_\alpha,B_\alpha}(i^\star)$ is not schedulable at time $\alpha+1$, thus $t^\sigma_j \geq t^\sigma_{i^\star} + 2$ and by coherence of $\sigma(\mathcal{G})$, $x^\sigma_{i^\star j} = 1$.

Now, any task $j \in PS_{W_\alpha,B_\alpha}(i^\star)$ has all its predecessors in W_α, thus $\forall k \in \Gamma^-(j)$, $t^\sigma_k + 2 \leq t^\sigma_{i^\star} + 2 \leq t^\sigma_j$, and by coherence of $\sigma(\mathcal{G})$, $x^\sigma_{kj} = 1$. We build another coherent schedule $\sigma'(\mathcal{G})$ as follows:

1. We first choose a task $j^\star \in PS_{W_\alpha,B_\alpha}(i^\star)$. We then set $x^{\sigma'}_{i^\star j^\star} = 0$ and for each arc $e = (k,\ell) \in \mathcal{A} - \{(i^\star, j^\star)\}$, $x^{\sigma'}_{k\ell} = x^\sigma_{k\ell}$.
2. We set $\forall i \in \mathcal{T}$, $t^{\sigma'}_i = \max(0, \max_{j \in \Gamma^-(i)}(t^{\sigma'}_j + 1 + x^{\sigma'}_{ji}))$.

For every task $i \in \mathcal{T}$, $t^{\sigma'}_i \leq t^\sigma_i$. Now, we get $\sum_{\ell \in \Gamma^+(i^\star)} x^{\sigma'}_{i^\star \ell} = \sum_{\ell \in \Gamma^+(i^\star) - \{j^\star\}} x^{\sigma'}_{i\ell} + x^{\sigma'}_{i^\star j^\star} = |\Gamma^+(i^\star)| - 1$. Similarly, we get $\sum_{\ell \in \Gamma^-(j^\star)} x^{\sigma'}_{\ell j^\star} = \sum_{\ell \in \Gamma^-(j^\star) - \{i^\star\}} x^{\sigma'}_{\ell j^\star} + x^{\sigma'}_{i^\star j^\star} = |\Gamma^-(j^\star)| - 1$, and thus $x^{\sigma'}$ is feasible.

Each task $i^\star$ is considered at most once and thus this transformation is done at most n times. So, it gives a feasible coherent schedule that satisfies the preferred sons property without increasing the makespan, thus the lemma holds. □

3.3.2 A general dominance property

In this section, we propose an as-soon-as-possible dominance property to reduce the number of idle processors.

Property 3.3.6. If a schedule $\sigma(\mathcal{G})$ satisfies **the general dominance property**, then we have for any task $i \in \mathcal{T}$, if the direct predecessors of task i are all scheduled before time $\alpha - 1$ and there

are unoccupied processors at time α, then we have task i is scheduled before time $\alpha + 1$. Thus,

$$\text{If } \Gamma^-(i) \subseteq \bigcup_{\beta=0}^{\alpha-2} \mathcal{T}_\beta^\sigma \text{ and } |\mathcal{T}_\alpha^\sigma| < m, \text{ then } t_i \leq \alpha.$$

The property above considers tasks $i \in B_\alpha$ without any preferred son. When a task i does not have preferred sons and there are available processors, its successors are to be sure to executed before or at $t_i + 2$ maintaining the feasibility of the schedule.

Lemma 3.3.7. *Let $\sigma(\mathcal{G})$ be a feasible schedule.Then there exists a feasible $\sigma'(\mathcal{G})$ satisfying the general dominance property such that, for each task $i \in \mathcal{T}$, $t_i^{\sigma'} \leq t_i^\sigma$.*

Proof. If $\sigma(\mathcal{G})$ satisfies the general dominance property, then $\sigma'(\mathcal{G})$ = $\sigma(\mathcal{G})$,

Let us suppose that $\sigma(\mathcal{G})$ does not verify the general dominance property. Let $(t_1^\sigma, t_2^\sigma, \ldots, t_n^\sigma)$ be the time vector and $(x_1^\sigma, x_2^\sigma, \ldots, x_n^\sigma)$ be the delay signal vector. Let $\alpha \in \{0, \ldots, C-1\}$ be the first time instant at which the property is not fulfilled, and task $i^\star$ is a corresponding task with $\Gamma^-(i^\star) \subseteq \bigcup_{\beta=0}^{\alpha-2} \mathcal{T}_\beta^\sigma$, $t_{i^\star}^\sigma = \alpha + 1$ and $|\mathcal{T}_\alpha^\sigma| < m$. Therefore, we have all $j \in \Gamma^-(i^\star)$ are scheduled before time $\alpha - 1$, i.e. $t_j^\sigma \leq \alpha - 2$

According to the definition of the delay signals vector, for every task $j \in \Gamma^-(i^\star)$, $x_{ji^\star}^\sigma = 1$.

We build another feasible schedule $\sigma'(\mathcal{G})$ as follows:

1. We set $t_{i^\star}^{\sigma'} = \alpha$
2. Keep other tasks' execution time, i.e. $t_i^{\sigma'} = t_i^\sigma, \forall i \in \mathcal{T} - \{i^\star\}$

So we have for every task $i \in \mathcal{T}$, $t_i^{\sigma'} \leq t_i^\sigma$.

Now, we get

$$\sum_{\ell \in \Gamma^+(i^\star)} x_{i^\star \ell}^{\sigma'} \geq \sum_{\ell \in \Gamma^+(i^\star)} x_{i^\star \ell}^{\sigma} \geq |\Gamma^+(i^\star)| - 1.$$

$$\sum_{\ell \in \Gamma^-(i^\star)} x_{\ell i^\star}^{\sigma'} = \sum_{\ell \in \Gamma^-(i^\star)} \min\{t_{i^\star} - t_\ell - 1, 1\} \geq |\Gamma^-(i^\star)| - 1,$$

Besides, $|B_\alpha| \leq m, \forall \alpha \in \{0, 1, 2, \ldots, C-1\}$. Thus $\sigma'(\mathcal{G})$ is feasible.

This transformation can be continued on the new schedule σ' until we obtain a schedule that satisfies the preferred sons' property. Each time instance can be is considered only once and thus this transformation is done at most $\overline{C}$ times, where $\overline{C}$ is a upper bound of C. Thus the lemma holds. □

4. Fixed-parameter complexity on optimization of makespan for UET-UCT model

In this chapter, we consider the optimization problem in the UET-UCT model on unlimited number of processors, i.e. $\overline{P}|prec, p_i = 1, c_i = 1, r_i, d_i|C_{max}$, which can be modelled using an integer linear program (P) defined in Section 2.1.1 with $p_i = 1$ and $c_{ij} = 1$.

4.1 Introduction

It is proven that problem $\overline{P}|prec, p_i = 1, c_i = 1, r_i, d_i|C_{max}$ is NP-hard [55], due to the importance of this problem, there are many researches on UET-UCT problems to provide exact and approximate algorithms on general and special precedence graphs.

Many authors considered scheduling problems with communication delays for a limited number of processors. An exact dynamic programming algorithm of time complexity $\mathcal{O}(2^{w(G)}.n^{2w(G)})$ was developed by Veltman [88] for $P|prec, p_i = 1, c_{ij} = 1|C_{max}$. The parameter $w(G)$ is the width of the precedence graph G defined as the size of its largest antichain. This algorithm can clearly be considered for solving the problem without limitation of the number of machines by setting the number of machines equal to the number of tasks. We can observe that it is a XP algorithm with parameter $w(G)$. Zinder et al. [92] have developed an exact branch-and-bound algorithm which converges to an optimal schedule for the problem $P|prec, p_i = 1, c_{ij} = 1|C_{max}$.

This problem was proved to be polynomial-time solvable for some special classes of graphs such as trees [22], series-parallel graphs [73] and generalized n-dimensional grid task graphs [5].

For the more general problem, $P|prec, c_{ij}|C_{max}$, Sinnen et al. in [81] have developed an enumerative $A^\star$ algorithm coupled with pruning methods. Orr and Sinnen [75] have developed an original techniques to reduce the space of exploration and to speed up branch-and-bound methods. Davies et. al. [29] gave a polynomial time $O(\log c \cdot \log m)-$approximation algorithm for this problem, where m is the number of machines and c is the communication delay, which based

on a Sherali-Adams lift of a linear programming relaxation and a randomized clustering of the semimetric space induced by this lift. Jansen et. al. [56] presented an efficient polynomial time approximation scheme (EPTAS) for scheduling fork-join task graphs with communication delay on homogeneous processors, denoted as $P|fork-graph, c_{ij}|C_{max}$, which is based on an integer program. Liu et al. [67] provided an $O(\frac{\ln c}{\ln \ln c})$-approximation algorithm with near-linear running time, i.e. in $\tilde{O}(|V|+|E|)$ time.

Several authors also considered integer linear programming formulations (ILP in short) to solve exactly scheduling problems with communications delays and a limited number of processors. Davidović et al. in [28] tackled the scheduling problems for a fixed network of processors; communications are proportional to both the amount of exchanged data between pairs of dependent tasks and the distance between processors in the multiprocessor architecture. They developed two formulations and they compared them experimentally. Later, Ait El Cadi et al. [3] improved this approach by reducing the size of the linear program (number of variables and constraints) and by adding cuts; they compared positively to the previous authors. Venugopalan and Sinnen in [89] provided a new ILP formulation for the usual problem $P|prec, c_{ij}|C_{max}$ and comparison with [28] for several classes of graphs and fixed number of processors.

Extensions of usual problems with communication delays were extensively studied. For example, the survey of Giroudeau and Koenig [45] considered a hierarchical communication model where processors are grouped into clusters. Shimada et al. [80] developed two heuristic based methods to consider both malleable tasks and communications delays for executing a program on an homogeneous multi-core computing system. Ait-Aba et al. [2] provided complexity results for an extension of the basic communication model for scheduling problems on an heterogeneous computing systems with two different resources.

Researchers are gaining interests on fixed-parameter tractability of scheduling problems. Mnich and van Bevern [70] surveyed main results on parameterized complexity for scheduling problems and identified 15 open problems. Bodlaender et. al. [17] considered scheduling problems with exact and minimum delay on chains of jobs. With exact delay and parameterized by the number

of chains, it is $W[1]$-complete on a single or a constant number of machines, and $W[2]$-complete when the number of machines is a variable.

Our algorithm is parameterized by pathwidth of the interval graph of tasks' time windows. This parameter is first used in the work of Munier [72] which developed a fixed-parameter algorithm for the problem $P|prec, p_i = 1|C_{max}$.

4.2 Dynamic programming approach and multistage graphs

This section presents our fixed-parameter algorithm. We start with the description of a multistage graph which presents the structure of the algorithm. After, we present the executable algorithm.

4.2.1 Description of the multistage graph

Let us consider a precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ and an upper bound $\overline{C}$ of the makespan. We build an associated multistage graph $S(\mathcal{G}) = (N, A)$ with $\overline{C}$ stages in which paths from the first stage to last stage represent all the feasible active schedules.

4.2.1.1 Nodes of $S(\mathcal{G})$

For any value $\alpha \in \{0, \ldots, \overline{C} - 1\}$, N_α is the set of nodes at stage α of graph $S(\mathcal{G})$. A node $p \in N$ is a couple $(W(p), B(p))$, where $W(p), B(p)$ are subsets of tasks and $B(p) \subseteq W(p) \subseteq \mathcal{T}$. If $p \in N_\alpha$, tasks from $W(p)$ have to be completed at time $\alpha + 1$, while those from $B(p)$ are scheduled exactly at time α. N_0 contains only one node p_0 with $B(p_0) = \{i \in \mathcal{T}, \Gamma^-(i) = \emptyset\}$ and $W(p_0) = B(p_0)$.

Observe that, for any value $\alpha \in \{0, \ldots, \overline{C} - 1\}$, all tasks from Z_α must be completed at time $\alpha + 1$, thus for any node $p \in N_\alpha$, $Z_\alpha \subseteq W(p)$. Moreover, by Lemma 3.2.3, $W(p) - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$.

4.2.1.2 Arcs of $S(\mathcal{G})$

For any $\alpha \in \{0, 1, \ldots, \overline{C} - 2\}$ and $(p, q) \in N_\alpha \times N_{\alpha+1}$, the arc $(p, q) \in A$ exists if there exists a feasible schedule such that tasks from $W(q)$ are all completed at time $\alpha + 2$ with tasks from $B(q)$

executed at time $\alpha+1$ and those from $B(p)$ at time α. The nodes p and q satisfy then the following conditions:

A.1 Since p is associated to a partial schedule of q, $W(p) \cup B(q) = W(q)$ and since tasks can only be executed once, $W(p) \cap B(q) = \emptyset$.

A.2 Any task $i \in B(q)$ must be schedulable at time $\alpha + 1$, thus all its predecessors must belong to $W(p)$. Then, $B(q) \subseteq \{i \in X_{\alpha+1}, \Gamma^-(i) \subseteq W(p)\}$.

A.3 Any task $i \in B(q)$ cannot have more than one predecessor scheduled at time α, thus $B(q) \subseteq \{i \in X_{\alpha+1}, |\Gamma^-(i) \cap B(p)| \leq 1\}$.

A.4 Any task $i \in X_{\alpha+1} - W(p)$ for which all its predecessors are completed at time α must be scheduled at time $\alpha + 1$. Thus, if $\Gamma^-(i) \subseteq W(p) - B(p)$, then $i \in B(q)$.

A.5 For any task $i \in B(p)$, if $PS_{W(p),B(p)}(i) \cap X_{\alpha+1} \neq \emptyset$, then by Definition 3.3.1, these successors of i are schedulable at time $\alpha + 1$. Following Lemma 3.3.5, we impose that exactly one among them is executed at time α on the same processor as i and thus $|PS_{W(p),B(p)}(i) \cap B(q)| = |\Gamma^+(i) \cap B(q)| = 1$. Otherwise, if $PS_{W(p),B(p)}(i) \cap X_{\alpha+1} = \emptyset$, no successor of i can be scheduled at time $\alpha + 1$ which corresponds to $|\Gamma^+(i) \cap B(q)| = 0$.

Remark 4.2.1. The preferred sons of a task $i \in \mathcal{T}$ were initially defined with respect to two sets of tasks W_α and B_α built from a feasible schedule. Here, for any node $p \in N_\alpha$, the definition of PS is extended to consider the sets $W(p)$ and $B(p)$ simply by assuming that tasks from $W(p)$ (resp. $B(p)$) are those which are completed at time $\alpha + 1$ (resp. performed at time α).

Figure 4.1 is the multistage graph associated with the precedence graph of Figure 2.3a and $\overline{C} = 6$. We observe that the path $(p_0, p_1^1, p_2^1, p_3^1, p_4^1)$ corresponds to the schedule shown in Figure 2.3b. On the same way, the path $(p_0, p_1^0, p_2^0, p_3^0, p_4^0)$ corresponds to the schedule shown by Figure 4.2.

4.2.2 Description of the algorithm

Algorithm 1 builds iteratively the multistage graph $S(\mathcal{G}) = (N, A)$. This algorithm returns false if there is no feasible schedule of makespan bounded by $\overline{C}$, otherwise it returns the optimum

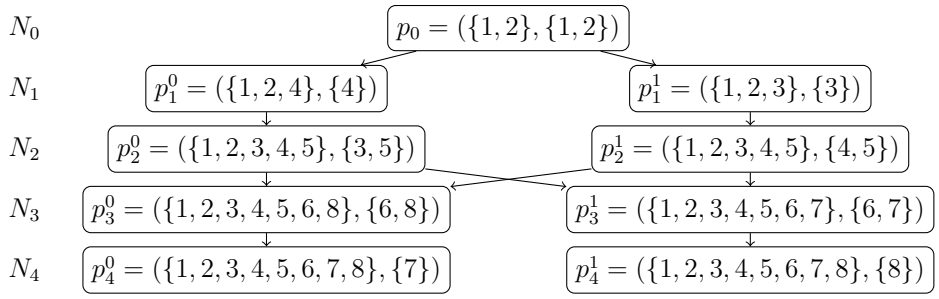


Figure 4.1: The multistage graph associated with the precedence graph of Figure 2.3a and $\overline{C} = 6$.

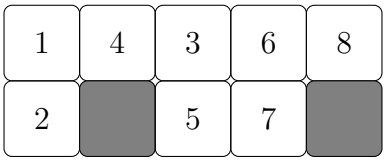


Figure 4.2: An optimum schedule corresponding to the path $(p_0, p_1^0, p_2^0, p_3^0, p_4^0)$ of Figure 4.1.

makespan. For any set of tasks $X \subseteq \mathcal{T}$, let $\mathcal{P}(X)$ be the power set of X, i.e. the set of all subsets of X including the empty ones. This algorithm is composed by three main sections. Lines $1-6$ correspond to the initialization step. Lines $7-9$ build all the possible nodes. Lines $10-17$ build the arcs and delete all the non connected nodes.

4.3 Correctness of the algorithms

In this section, we prove that all feasible active schedules are presented by a path in graph $S(\mathcal{G})$, and all paths from the first stage to a node p with $W(p) = \mathcal{T}$ are feasible schedules.

Lemma 4.3.1. *Any feasible schedule $\sigma(\mathcal{G})$ of makespan $C \leq \overline{C}$ corresponds to a path of $S(\mathcal{G})$ ending with a node p with $W(p) = \mathcal{T}$.*

Proof. Let suppose that $\sigma(\mathcal{G})$ is a feasible active schedule of makespan $C \leq \overline{C}$. By Lemma 3.3.5, we have that $\sigma(\mathcal{G})$ satisfies the preferred sons property and $\sigma(\mathcal{G})$ is an as-soon-as-possible schedule, i.e. for any task $i \in \mathcal{T}$, $t_i^\sigma = \max(0, \max_{i \in \Gamma^-(j)}(t_j^\sigma + 1 + x_{ji}))$.

For every integer $\alpha \in \{0, \ldots, C-1\}$, we set $\mathcal{T}_\alpha^\sigma = \{i \in \mathcal{T}, t_i^\sigma = \alpha\}$. Let us consider the sequence $q_\alpha = (W(q_\alpha), B(q_\alpha))$ defined as $W(q_\alpha) = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma$ and $B(q_\alpha) = \mathcal{T}_\alpha^\sigma$ for $\alpha \in$

Algorithm 1: Optimum makespan $C \leq \overline{C}$ if it exists, false otherwise.

Input: A precedence graph $\mathcal{G} = (\mathcal{T}, \mathcal{A})$, an upper bound of the makespan $\overline{C}$
Output: Optimum makespan $C \leq \overline{C}$ if it exists, false otherwise

for $i \in \mathcal{T}$ **do**
  Calculate r_i and d_i
for $\alpha \in \{0, 1, \ldots, \overline{C} - 1\}$ **do**
  Calculate X_α and Z_α
Initialize a set of arcs $A = \emptyset$
Let $N_0 = \{p_0\}$ with $B(p_0) = \{i \in \mathcal{T}, \Gamma^-(i) = \emptyset\}$ and $W(p_0) = B(p_0)$
for $\alpha \in \{1, 2, \ldots, \overline{C} - 1\}$ **do**
  Build the sets $\mathcal{P}(X_\alpha \cap X_{\alpha+1})$ and $\mathcal{P}(X_\alpha)$
  $N_\alpha = \{p = (W, B), W = Y \cup Z_\alpha, Y \in \mathcal{P}(X_\alpha \cap X_{\alpha+1}), B \in \mathcal{P}(X_\alpha), B \subseteq W\}$
for $\alpha \in \{0, 2, \ldots, \overline{C} - 2\}$ **do**
  for $(p, q) \in N_\alpha \times N_{\alpha+1}$ **do**
    if *conditions A.1, A.2, A.3, A.4, A.5 are met for* (p, q) **then**
      $A = A \cup \{(p, q)\}$
      if $W(q) = \mathcal{T}$ **then**
        return $\alpha + 2$
  Delete all the vertices $p \in N_{\alpha+1}$ without predecessor
return False

$\{0, \ldots, C - 1\}$.

For $\alpha = 0$, $W(q_0) = \mathcal{T}_0^\sigma = \{i \in \mathcal{T}, t_i^\sigma = 0\} = \{i \in \mathcal{T}, \Gamma^-(i) = \emptyset\} = W(p_0)$, where p_0 is the only node in the first stage of graph $S(\mathcal{G})$. Thus q_0 has the same structure as node p_0.

Since $\sigma(\mathcal{G})$ is feasible, for every value $\alpha \in \{0, \ldots, C - 1\}$, $\mathcal{T}_\alpha^\sigma \subseteq X_\alpha$. According to Lemma 3.2.3, $\bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$. So the node $q_\alpha = (W(q_\alpha), B(q_\alpha))$ has been built at stage α.

We prove then that, for every value $\alpha \in \{0, \ldots, C - 2\}$, $(q_\alpha, q_{\alpha+1}) \in A$.

- $W(q_{\alpha+1}) = \bigcup_{\beta=0}^{\alpha+1} \mathcal{T}_\beta^\sigma = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma \cup \mathcal{T}_{\alpha+1}^\sigma = W(q_\alpha) \cup B(q_{\alpha+1})$. Moreover, $W(q_\alpha) \cap B(q_{\alpha+1}) = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma \cap \mathcal{T}_{\alpha+1}^\sigma = \emptyset$. Thus, *A.1* is verified.

- Since $\sigma(\mathcal{G})$ is feasible, tasks from $B(q_{\alpha+1})$ are schedulable at time $\alpha + 1$ and thus, properties *A.2* and *A.3* are verified.

- Since $\sigma(\mathcal{G})$ is an as-soon-as-possible schedule, property *A.4* is fulfilled.

- Lastly, since $\sigma(\mathcal{G})$ satisfies the preferred sons property, *A.5* is fulfilled.

We conclude that $(q_0, q_1, \ldots, q_{C-1})$ is a path of $S(\mathcal{G})$. Moreover, since $\sigma(\mathcal{G})$ is of makespan C, $W(q_{C-1}) = \mathcal{T}$, and thus the lemma is verified. □

Lemma 4.3.2. *Let $C \leq \overline{C}$ and $(p_0, p_1, \ldots, p_{C-1})$ be a path of $S(\mathcal{G})$ with $W(p_{C-1}) = \mathcal{T}$. Then, for each task $i \in \mathcal{T}$, there exists a unique value $\alpha \in \{0, \ldots, C-1\}$ such that $i \in B(p_\alpha)$.*

Proof. According to the definition of $S(\mathcal{G})$, $W(p_0) \subset W(p_1) \subset \cdots \subset W(p_{C-1})$. Moreover, by assumption, $W(p_{C-1}) = \mathcal{T}$. Thus, for each task $i \in \mathcal{T}$, there is a unique $\alpha \in \{0, \ldots, C-1\}$ with $i \in W(p_\alpha)$ and $i \notin W(p_{\alpha-1})$. Since $W(p_{\alpha-1}) \cup B(p_\alpha) = W(p_\alpha)$, we get $i \in B(p_\alpha)$. □

Lemma 4.3.3. *Every path $(p_0, p_1, \ldots, p_{C-1})$ of $S(\mathcal{G})$ with $C \leq \overline{C}$ and $W(p_{C-1}) = \mathcal{T}$ is associated to a feasible schedule whose makespan is equal to C.*

Proof. Let $(p_0, p_1, \ldots, p_{C-1})$ be a path of $S(\mathcal{G})$ with $C \leq \overline{C}$ and $W(p_{C-1}) = \mathcal{T}$. A schedule $\sigma(\mathcal{G})$ of makespan C is defined as follows:

- By Lemma 4.3.2, for any task $i \in \mathcal{T}$, there exists a unique value $\alpha \in \{0, \ldots, C-1\}$ with $i \in B(p_\alpha)$. Thus, we set $t_i^\sigma = \alpha$.

- For any arc $(i, j) \in \mathcal{A}$, we set $x_{ij}^\sigma = 1$ if $t_j^\sigma > t_i^\sigma + 1$, otherwise $x_{ij}^\sigma = 0$.

We prove that the schedule $\sigma(\mathcal{G})$ satisfies the integer linear program (P) defined in Section 2.1.1 with $p_i = 1$ and $c_{ij} = 1$.

According to the condition *A.2*, we get $\Gamma^-(i) \subseteq W(p_{\alpha-1})$, so $t_j^\sigma + 1 \leq t_i^\sigma, \forall (j, i) \in \mathcal{A}$. Following the definition of x^σ, we observe that equations (1) are true.

Now, by definition of $\sigma(\mathcal{G})$, $t_i^\sigma \leq C - 1, \forall i \in \mathcal{T}$ and thus equations (2) are validated.

According to the condition *A.5*, for any task $i \in B(p_\alpha)$, $\alpha \in \{0, \ldots, C-2\}$

C.1 If $PS_{W(p_\alpha), B(p_\alpha)}(i) \cap X_{\alpha+1} \neq \emptyset$, then there is exactly one task $j^\star \in \Gamma^+(i) \cap B(p_{\alpha+1})$, i.e. such that $t_{j^\star}^\sigma = \alpha + 1 = t_i^\sigma + 1$. The task $j^\star$ is thus the unique successor of i for which $x_{ij^\star}^\sigma = 0$ and $\forall j \in \Gamma^+(i) - \{j^\star\}$, $x_{ij}^\sigma = 1$. Thus, $\forall i \in \mathcal{T}, \sum_{j \in \Gamma^+(i)} x_{ij}^\sigma = |\Gamma^+(i)| - 1$.

C.2 If $PS_{W(p_\alpha),B(p_\alpha)}(i) \cap X_{\alpha+1} = \emptyset$, then no successor of i is scheduled at time $\alpha + 1$, thus $\forall j \in \Gamma^+(i)$, $x^\sigma_{ij} = 1$ and $\sum_{j\in\Gamma^+(i)} x^\sigma_{ij} = |\Gamma^+(i)|$.

Therefore, equations (3) are checked. Lastly, according to the condition *A.3*, any task $i \in B(p_{\alpha+1})$ cannot have more than one predecessor in $B(p_\alpha)$, thus i has at least one predecessor $j^\star$ such that $x^\sigma_{j^\star i} = 0$. Therefore, $\forall i \in \mathcal{T}, \sum_{j\in\Gamma^-(i)} x^\sigma_{ji} \geq |\Gamma^-(i)| - 1$ and equations (4) are validated. We conclude that $\sigma(\mathcal{G})$ is a feasible schedule, and the lemma is proved. □

Theorem 4.3.4 (Validity of Algorithm 1)**.** *Algorithm 1 returns the minimum makespan C of a feasible schedule if $C \leq \overline{C}$, false otherwise.*

Proof. Let us suppose first that our algorithm returns $C \leq \overline{C}$, then the minimum path from p_0 to a node p with $W(p) = \mathcal{T}$ is of length C. By Lemma 4.3.3, this path is associated to a feasible schedule of makespan C. Thus this schedule is optimal.

Now, let us suppose that such a path does not exist; in this case, Algorithm 1 returns false. By Lemma 4.3.1, there is no feasible schedule of makespan $C \leq \overline{C}$ and the theorem is proved. □

4.4 Complexity results

In this section, we prove that Algorithm 1 is a fixed-parameter tractable algorithm.

Lemma 4.4.1. *Let us denote by n the number of tasks and $pw(\overline{C})$ the pathwidth of the interval graph built with the upper bound $\overline{C}$ of the minimum makespan. The number of nodes $|N|$ of the multistage graph $S(\mathcal{G}) = (N, A)$ is $\mathcal{O}(n \cdot 2^{2pw(\overline{C})})$ and the number of arcs $|A|$ is $\mathcal{O}(n \cdot 2^{4pw(\overline{C})})$.*

Proof. According to Algorithm 1, each node $p \in N_\alpha$ is such that $p = (W(p), B(p))$ with $W(p) = Y(p) \cup Z_\alpha$ and $Y(p) \subseteq X_\alpha \cap X_{\alpha+1}$. The number of possibilities for $Y(p)$ is thus bounded by $2^{|X_\alpha \cap X_{\alpha+1}|} \leq 2^{|X_\alpha|}$. Now, since $B(p) \subseteq X_\alpha$, the number of possibilities for $B(p)$ is bounded by $2^{|X_\alpha|}$. Then, the number of nodes in N_α for $\alpha \in \{0, \ldots, \overline{C} - 1\}$ is bounded by $2^{2|X_\alpha|}$.

By definition of the pathwidth, the value $|X_\alpha|$ is bounded by $pw(\overline{C}) + 1$, thus the number of nodes $|N_\alpha|$ is $\mathcal{O}(2^{2pw(\overline{C})})$. Now, since $\overline{C} \leq n$, the number of nodes $|N|$ is $\mathcal{O}(n \cdot 2^{2pw(\overline{C})})$. Moreover,

the size of $N_\alpha \times N_{\alpha+1}$ for $\alpha \in \{0, \ldots, \overline{C}-1\}$ is $\mathcal{O}(2^{4pw(\overline{C})})$, thus the whole number of arcs $|A|$ is $\mathcal{O}(n \cdot 2^{4pw(\overline{C})})$, and we get the lemma. □

Lemma 4.4.2. *For any $\alpha \in \{0, \ldots, \overline{C}-2\}$, the time complexity of checking the conditions A.1 to A.5 for a couple of nodes $(p, q) \in N_\alpha \times N_{\alpha+1}$ is $O(n^2 \cdot pw(\overline{C}))$.*

Proof. The time complexity for checking the condition *A.1* is $O(n)$. For the condition *A.2*, we need to build the set $\{i \in X_{\alpha+1}, \Gamma^-(i) \subseteq W(p)\}$. If we denote by m the number of arcs of $\mathcal{G}$, building this set requires to enumerate all the successors of tasks in $X_{\alpha+1}$, which is in time complexity equal to $\mathcal{O}(m)$. Since $m \leq n^2$, the time complexity for checking the condition *A.2* is thus $O(n^2 \cdot pw(\overline{C}))$. For the same reasons, time complexity for checking the conditions *A.3* and *A.4* is also $O(n^2 \cdot pw(\overline{C}))$. For condition the *A.5*, the time complexity of the computation of the preferred sons of a task $i \in B(p)$ is also $O(n^2)$, and thus checking this condition also takes $\mathcal{O}(n^2 \cdot pw(\overline{C}))$, and the lemma holds. □

Theorem 4.4.3 (Complexity of Algorithm 1)**.** *The time complexity of Algorithm 1 is $\mathcal{O}(n^3 \cdot pw(\overline{C}) \cdot 2^{2pw(\overline{C})})$, where $pw(\overline{C})$ is the pathwidth of the interval graph associated to the time windows $[r_i, d_i]$, $i \in \mathcal{T}$.*

Proof. The time complexity of the computation of the release dates and deadlines (lines $1-2$) and the sets X_α and Z_α for $\alpha \in \{0, \ldots \overline{C}\}$ (lines $3-4$) is $\mathcal{O}(n^2)$ since $\overline{C}$ is bounded by n. The time complexity for building N at lines $7-9$ is $\mathcal{O}(n \cdot 2^{2pw(\overline{C})})$ by Lemma 4.4.1. Following Lemma 4.4.1 and 4.4.2, the complexity of building arcs of $S(\mathcal{G})$ in lines $10-17$ is $\mathcal{O}(n^3 \cdot pw(\overline{C}) \cdot 2^{4pw(\overline{C})})$, thus the theorem holds. □

4.5 Conclusion

We have shown in this paper that the problem $\overline{P}|prec, p_i = 1, c_{ij} = 1|C_{max}$ is fixed-parameter tractable. The parameter considered is the pathwidth associated with an upper bound $\overline{C}$ of the makespan. For this purpose, we have developed a dynamic programming algorithm of complexity $\mathcal{O}(n^3 \cdot pw(\overline{C}) \cdot 2^{4pw(\overline{C})})$. This is, as far as we know, the first fixed-parameter algorithm for a scheduling problem with communication delays.

This work opens up several perspectives. The first one is to test experimentally the efficiency of this algorithm, and to compare it to other exact methods such as integer linear programming or dedicated exact methods [75, 81]. A second perspective is to study the extension of this algorithm to more general problems in order to get closer to applications and to evaluate if these approaches can be considered to solve real-life problems.

5. Extensions to maximum lateness

In the previous chapter, we have given an algorithm to minimizing the makespan on UET-UCT model. In this chapter, we will do some modification to the algorithm to minimizing the maximum lateness L_{max} of a schedule. It considers the minimization of the maximum lateness for a set of dependent tasks with unit duration, unit communication delays release times and due dates. The number of processors is limited, and each task requires one processor for its execution. A fixed-parameter algorithm based on a dynamic programming approach is developed to solve this optimization problem. This is, as far as we know, the first fixed-parameter algorithm for a scheduling problem with communication delays and limited number of processors.

This chapter is submitted to journal RAIRO and is co-authored by my supervisor Alix Munier-Kordon.

5.1 Problem definition

The scheduling problem considered is described in Section 5.1.1, while a small example is presented in Section 5.1.2. In Section 5.1.3, a dominance property of active schedules is described.

5.1.1 Problem definition

Let $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ be a precedence graph of unit execution time tasks. For each task $i \in \mathcal{T}$, let $\Gamma^+(i)$ (resp. $\Gamma^-(i)$) be the set of direct successors (resp. predecessors) of i, i.e. $\Gamma^+(i) = \{j \in \mathcal{T}, (i,j) \in \mathcal{A}\}$ and $\Gamma^-(i) = \{j \in \mathcal{T}, (j,i) \in \mathcal{A}\}$.

We observe that a feasible schedule σ is completely defined by the starting times vector $t^\sigma \in \mathbb{N}^n$. Indeed, for any arc $e = (i,j) \in \mathcal{A}$, we note x_{ij}^σ the communication delay between the tasks i and j; we set $x_{ij}^\sigma = 0$ if the execution of the task j starts right after the task i. These two tasks are necessarily executed by a same processor and the communication delay is removed. Otherwise, a communication delay is required between the completion time of the task i and the starting time of the task j and thus $x_{ij}^\sigma = 1$. We then set $x_{ij}^\sigma = \min\{t_j^\sigma - t_i^\sigma - 1, 1\}$ for each arc $e = (i,j) \in \mathcal{A}$.

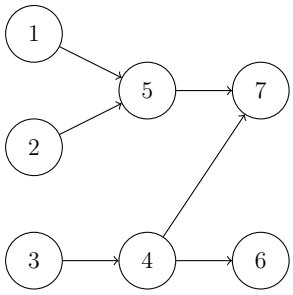
The problem considered is expressed below. A time-indexed formulation should be considered to transform it into an integer linear program [82] for modelling the resource constraints. We set $d_{max} = \max_{i \in \mathcal{T}} d_i$ (resp. $r_{max} = \max_{i \in \mathcal{T}} r_i$) the maximum due date (resp. release time); we also suppose that an upper bound of the maximum lateness $\overline{L}$ is fixed. We then observe that $\overline{C} = \min(r_{max} + 2n, d_{max} + \overline{L})$ is an upper bound of the makespan of any active feasible schedule which maximum lateness is bounded by $\overline{L}$.

$$
\begin{cases}
\text{minimize } L & \\
t \in \mathbb{N}^n; L \in \mathbb{Z}; \forall e = (i,j) \in \mathcal{A}, x_{ij} \in \{0,1\} & (1) \\
\forall i \in \mathcal{T}, L \geq t_i + 1 - d_i \text{ and } t_i \geq r_i & (2) \\
\forall e = (i,j) \in \mathcal{A}, x_{ij} = \min\{t_j - t_i - 1, 1\} & (3) \\
\forall e = (i,j) \in \mathcal{A}, t_i < t_j & (4) \\
\forall i \in \mathcal{T}, \sum\limits_{j \in \Gamma^+(i)} x_{ij} \geq |\Gamma^+(i)| - 1 & (5) \\
\forall i \in \mathcal{T}, \sum\limits_{j \in \Gamma^-(i)} x_{ji} \geq |\Gamma^-(i)| - 1 & (6) \\
\forall \alpha \in \{0, \ldots, \overline{C}\}, |\{i \in \mathcal{T}, t_i = \alpha\}| \leq m & (7)
\end{cases}
$$

Since communications delays and length of the tasks are unitary, starting times can be reduced to integer values; Inequalities (2) come from the definition of the maximum lateness and the release dates. Communication delays are defined from the starting time of the tasks (3). Inequalities (4), (5) and (6) express the communication delay constraints: any task i has at most one successor (resp. predecessor) performed at its completion time (resp. just before its starting time) on the same processor. Inequalities (7) express the limitation on the number of processors.

5.1.2 Example

Let us consider an instance of our scheduling problem defined by 7 tasks of unit length. The precedence graph, release dates and due dates are reported by Figure 5.1. The number of machines is fixed to 2. A feasible schedule σ of maximum lateness $L_{\max}(\sigma) = 2$ is given by Figure 5.2.



$i \in \mathcal{T}$	1	2	3	4	5	6	7
r_i	0	0	0	1	1	2	2
d_i	1	1	1	2	2	3	3

Figure 5.1: An instance of $P|r_i, prec, p_i = 1, c_{ij} = 1|L_{\max}$ with $m = 2$ machines.

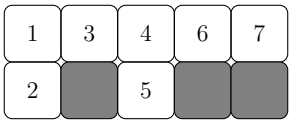


Figure 5.2: A feasible schedule σ of maximum lateness $L_{\max}(\sigma) = 2$ associated to the example given in Figure 5.1.

5.1.3 A general dominance property of active schedules

Let us consider that σ is a feasible schedule of maximum lateness bounded by $\bar{L}$. For every integer $\alpha \in \{-1, \ldots, \overline{C} - 1\}$, we set $W_\alpha = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma$ and $B_\alpha = \mathcal{T}_\alpha^\sigma$. The set W_α contains all the tasks that are executed in time $[0, \alpha + 1)$, and B_α contains all the tasks that are executed at time α. Notice that $W_{-1} = B_{-1} = \emptyset$.

For a fixed value $\alpha \in \{-1, \ldots, \overline{C} - 2\}$, we note $\mathcal{S}(W_\alpha, B_\alpha)$ to be the set of tasks from $X_{\alpha+1} - W_\alpha$ that are schedulable at time $\alpha + 1$ following W_α and B_α. Formally, $\mathcal{S}(W_\alpha, B_\alpha) = \{i \in X_{\alpha+1} - W_\alpha, \Gamma^-(i) \subseteq W_\alpha \text{ and } |\Gamma^-(i) \cap B_\alpha| \leq 1\}$.

Now, we observe that a set of tasks B can be scheduled at time $\alpha + 1$ following W_α and B_α if $B \subseteq \mathcal{S}(W_\alpha, B_\alpha)$ with $|B| \leq m$ and there is no couple of tasks $(i, j) \in B^2$ with a same predecessor in B_α (i.e. for each couple of tasks $(i, j) \in B^2$, $\Gamma^-(i) \cap \Gamma^-(j) \cap B_\alpha = \emptyset$). We set then $C(W_\alpha, B_\alpha)$ to be the set of all the subsets of $\mathcal{S}(W_\alpha, B_\alpha)$ that fulfills all these conditions.

Lastly, we may reduce our study to active schedules without loss of generality; then we set $A(W_\alpha, B_\alpha)$ to be the set of the elements from $C(W_\alpha, B_\alpha)$ that are maximum for the inclusion.

Lemma 5.1.1. *Let us consider that σ is an active feasible schedule of maximum lateness bounded by $\bar{L}$. For any value $\alpha \in \{-1, \ldots, \overline{C} - 2\}$, $\mathcal{T}^{\sigma}_{\alpha+1} \in A(W_\alpha, B_\alpha)$.*

Proof. Let us suppose by contradiction that, for a fixed value $\alpha \in \{-1, \ldots, \overline{C} - 2\}$, $\mathcal{T}^{\sigma}_{\alpha+1} \notin A(W_\alpha, B_\alpha)$. Since tasks from $\mathcal{T}^{\sigma}_{\alpha+1}$ are all schedulable together at time $\alpha + 1$, $\mathcal{T}^{\sigma}_{\alpha+1} \in C(W_\alpha, B_\alpha)$, and thus $\mathcal{T}^{\sigma}_{\alpha+1} \in C(W_\alpha, B_\alpha) - A(W_\alpha, B_\alpha)$. The consequence is that $\mathcal{T}^{\sigma}_{\alpha+1}$ is not maximum for the inclusion in $C(W_\alpha, B_\alpha)$, and thus σ is not active, a contradiction. □

5.2 Description of the algorithm

This section is dedicated to the description of our algorithm. Subsection 5.2.1 describes the multistage graph $S(\mathcal{G})$ built, while Subsection 5.2.2 is devoted to the algorithm.

5.2.1 Description of the multistage graph

Our algorithm builds an associated multistage graph $S(\mathcal{G}) = (N, A)$ described as follows:

5.2.1.1 Nodes of $S(\mathcal{G})$

The set of nodes N is partitioned into $\overline{C} + 1$ stages. For any value $\alpha \in \{-1, \ldots, \overline{C} - 1\}$, N_α is the set of nodes at stage α. A node $p \in N_\alpha$ is a triple $(W(p), B(p), L(p))$, where $B(p) \subseteq W(p) \subseteq \mathcal{T}$ and $L(p) \in \mathbb{Z} \cup \{+\infty\}$. The node p is associated to the feasible schedules $\sigma(p)$ of tasks from $W(p)$ ending at time $\alpha + 1$ with tasks from $B(p)$ scheduled at time α. $L(p)$ is the minimum maximum lateness among all the feasible schedules associated to p. Moreover, $N_{-1} = \{s\}$ with $W(s) = B(s) = \emptyset$ and $L(s) = -\infty$. For each value $\alpha \in \{0, \ldots, \overline{C} - 1\}$, each node $p \in N_\alpha$ fulfils next conditions:

N.1 Tasks from Z_α must be completed before time $\alpha + 1$, thus for each $p \in N_\alpha$, $Z_\alpha \subseteq W(p)$;

N.2 For each node $p \in N_\alpha$, the tasks from $B(p)$ are all executed simultaneously at time α, thus $B(p) \subseteq X_\alpha$ and $|B(p)| \leq m$;

N.3 By Lemma 3.2.3, for each node $p \in N_\alpha$, $W(p) - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$.

5.2.1.2 Arcs of $S(\mathcal{G})$

For each value $\alpha \in \{0, \ldots, \overline{C} - 2\}$ and $(p, q) \in N_\alpha \times N_{\alpha+1}$, there is an arc $(p, q) \in A$ if the following conditions are fulfilled:

A.1 Tasks from $W(q)$ are completed at time $\alpha+2$ with tasks from $B(q)$ executed at time $\alpha+1$ and those from $B(p)$ at time α. Thus, $W(p) \cup B(q) = W(q)$ and since tasks are only executed once, $W(p) \cap B(q) = \emptyset$;

A.2 Any task $i \in B(q)$ must be schedulable at time $\alpha + 1$ and all the schedule considered are active, thus $B(q) \in A(W(p), B(p))$;

A.3 The node s is a source of $S(\mathcal{G})$, thus for any node $p \in N_0$, $(s, p) \in A$.

5.2.1.3 Maximum Lateness of a node of $S(\mathcal{G})$

For any node $q \in N_\alpha$ with $\alpha \in \{0, \ldots, \overline{C} - 1\}$, let $\ell(q) = \alpha + 1 - \min_{i \in B(q)} d_i$ be the maximum lateness of the tasks from $B(q)$. Recall that these tasks are executed at time α.

For any value $\alpha \in \{-1, \ldots, \overline{C} - 1\}$ and $q \in N_\alpha$, $L(q)$ is the minimum maximum lateness of a schedule of tasks from $W(q)$ with $B(q) \subseteq W(q)$ scheduled at time α. L is defined as follows:

1. by convention, $L(s) = -\infty$;
2. for any value $\alpha \in \{0, \ldots, \overline{C} - 1\}$ and $q \in N_\alpha$,

$$L(q) = \max(\ell(q), \min_{p \in \Gamma^-(q)} L(p)).$$

Here, $\Gamma^-(q)$ is the set of the immediate predecessors of q in $S(\mathcal{G})$.

5.2.2 Description of the algorithm

Algorithm 2 builds iteratively the multistage graph $S(\mathcal{G}) = (N, A)$. For any set of tasks $X \subseteq \mathcal{T}$, let $\mathcal{P}(X)$ be the set of all subsets of X including the empty set. This algorithm returns the minimum value of the maximum lateness if it is upper bounded by $\overline{L}$, false otherwise.

The algorithm is composed by three main sections. Lines 1-4 correspond to the initialization step. Lines 5-6 build all the possible nodes following conditions *N.1*, *N.2* and *N.3*. Lines 7-18 build the arcs and delete all the non connected nodes.

Algorithm 2: Minimum maximum lateness L_{opt} if $L_{opt} \leq \overline{L}$, false otherwise.

1 **for** $\alpha \in \{0, 1, \ldots, \overline{C}\}$ **do**
2 Calculate X_α and Z_α
3 $N_{-1} = \{s = (W, B, L), W = B = \emptyset$ and $L = -\infty\}$
4 $N = N_{-1}$, $A = \emptyset$, $L_{opt} = +\infty$
5 **for** $\alpha \in \{0, \ldots, \overline{C} - 1\}$ **do**
6 $N_\alpha = \{p = (W, B, L), W = Y \cup Z_\alpha, Y \in \mathcal{P}(X_\alpha \cap X_{\alpha+1}), B \in \mathcal{P}(X_\alpha), B \subseteq W, |B| \leq m$ and $L = +\infty\}$
7 **for** $\alpha \in \{-1, \ldots, \overline{C} - 2\}$ **do**
8 **for** $p \in N_\alpha$ **do**
9 **if** $W(p) \neq \mathcal{T}$ **then**
10 **for** $B \in A(W(p), B(p))$ **do**
11 Find $q \in N_{\alpha+1}$ such that $(W(p) \cup B, B) = (W(q), B(q))$
12 $\ell(q) = \alpha + 2 - \min_{i \in B(q)} d_i$
13 **if** $L(q) > \ell(q)$ **then**
14 $L(q) = \max(\ell(q), \min(L(p), L(q)))$
15 $A = A \cup \{(p, q)\}$
16 **else**
17 $L_{opt} = \min(L_{opt}, L(p))$
18 $N_{\alpha+1} = \{q \in N_{\alpha+1}, \Gamma^-(q) \neq \emptyset\}$, $N = N \cup N_{\alpha+1}$
19 **if** $L_{opt} = +\infty$ **then**
20 **return** false
21 **return** L_{opt}

Figure 5.3 presents the graph $S(\mathcal{G})$ built by Algorithm 2 associated to the example shown by Figure 5.1 and the upper bound $\overline{L} = 2$. Algorithm 2 returns the optimal value $L_{opt} = 1$.

We observe that the schedule presented by Figure 5.2 is associated to the path $s \to p_0 \to p_1 \to p_2 \to p_3 \to p_4$ with $p_0 = (\{1, 2\}, \{1, 2\}, 0)$, $p_1 = (\{1, 2, 3\}, \{3\}, 1)$, $p_2 = (\{4, 5\}, \{4, 5\}, 1)$, $p_3 = (\{6\}, \{6\}, 1)$ and $p_4 = (\emptyset, \{7\}, 2)$. The maximum lateness of the path is $L(p_4) = 2$.

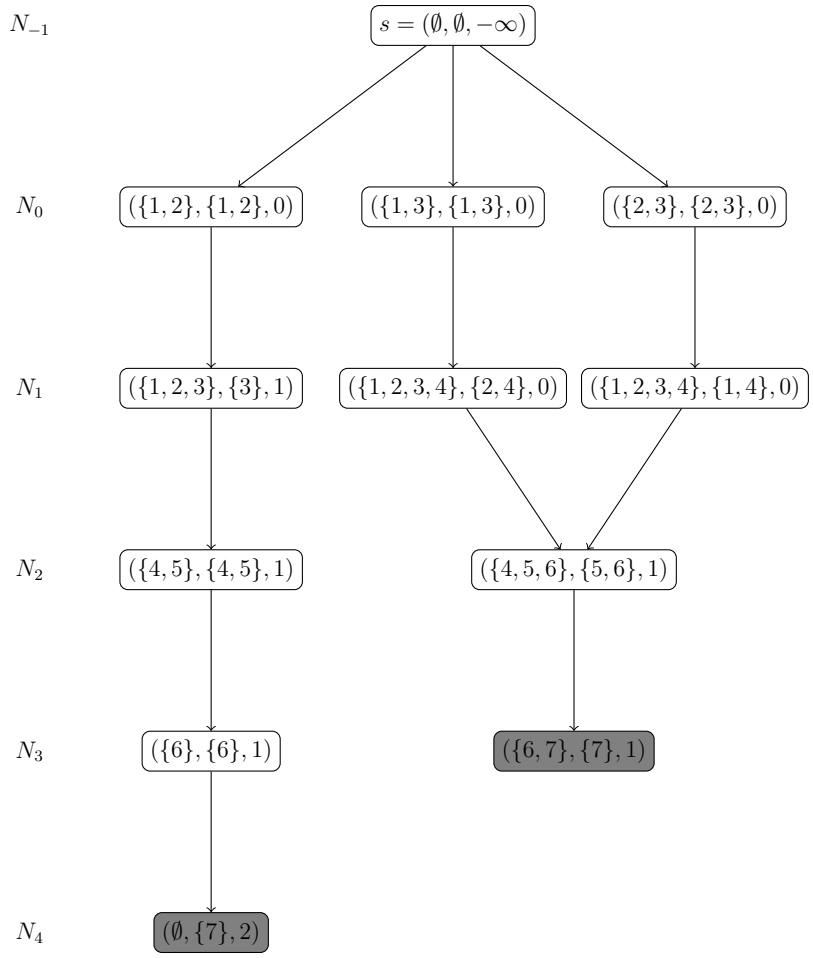


Figure 5.3: The multistage auxiliary graph $S(\mathcal{G}) = (N, A)$ associated with the example given in Figure 5.1. Each node $p \in N_\alpha$ is designated by the triple $(W(p) - Z_\alpha, B(p), L(p))$.The nodes p filled in gray are associated to a set of feasible schedules where minimum maximum lateness is $L(p)$.

Conversely, the path $s \to q_0 \to q_1 \to q_2 \to q_3$ with $q_0 = (\{1,3\},\{1,3\},0)$, $q_1 = (\{1,2,3,4\},\{2,4\},0)$, $q_2 = (\{4,5,6\},\{5,6\},1)$ and $q_3 = (\{6,7\},\{7\},1)$ corresponds to the active feasible schedule σ of maximum lateness $L_{\max}(\sigma) = 1$ presented by Figure 5.4.

5.3 Correctness of the algorithm

This section is devoted to the proof of the correctness of Algorithm 2. Lemma 5.3.1 shows that the evaluation of the maximum lateness $L(q)$ for each node $q \in N$ is correct with respect to the definition of Subsection 5.2.1. Lemma 5.3.2 shows that any active feasible schedule is associated

Figure 5.4: The active feasible schedule σ of maximum lateness $L_{\max}(\sigma) = 1$ associated to the path $s \to q_0 \to q_1 \to q_2 \to q_3$ with $q_0 = (\{1,3\},\{1,3\},0)$, $q_1 = (\{1,2,3,4\},\{2,4\},0)$, $q_2 = (\{4,5,6\},\{5,6\},1)$ and $q_3 = (\{6,7\},\{7\},1)$.

to a path of $S(\mathcal{G})$ from s to a node without successor, while Lemma 5.3.4 proves that extremum paths of $S(\mathcal{G})$ are associated to feasible schedules. Our main theorem follows.

Lemma 5.3.1. *For any node $q \in N_\alpha$ with $\alpha \in \{-1,\dots,\overline{C}-1\}$, the value $L(q)$ computed by Algorithm 2 follows the definition of the minimum maximum lateness of a node (see. Subsection 5.2.1).*

Proof. $L(s)$ is set to $-\infty$ and is not modified. Let us consider a node $q \in N_\alpha$ of $S(\mathcal{G})$ with $\alpha \in \{0,\dots,\overline{C}-1\}$ and the value $\ell(q) = \alpha + 1 - \min_{i\in B(q)} d_i$. If $\alpha = 0$, $L(q) = \ell(q)$ is correctly set at line 5 of Algorithm 2.

Now, if $\alpha > 0$, $L(q) = +\infty$ at the initialization of the node q. Since q belongs to $S(\mathcal{G})$, then $\Gamma^-(q) \neq \emptyset$, thus the value $L(q)$ is adjusted once for each predecessor of q. At the first adjustment, corresponding to its predecessor p_1, we set $L(q) = \max(\ell(q), L(p_1))$ since $L(q)$ was initialized to $+\infty$.

Now, if there exists a predecessor p of q such that $L(p) \leq \ell(q)$, then, let p_1 be the first predecessor of q considered by the Algorithm 2 such that $L(p_1) \leq \ell(q)$. At its corresponding loop, $L(q)$ is set to $\ell(q)$ and is not modified further; the maximum lateness of q is then correct.

Otherwise, all the predecessors p of q verify $L(p) > \ell(q)$, thus $L(q) > \ell(q)$ and $L(q)$ is then equal to $\min_{p\in\Gamma^-(q)} L(p)$, which is also the right definition of the maximum lateness of q. □

Lemma 5.3.2. *Any active feasible schedule σ of maximum lateness bounded by $\overline{L}$ is associated to a path $\nu(\sigma)$ of $S(\mathcal{G})$ ending at a node p with $W(p) = \mathcal{T}$. Moreover, the maximum lateness $L_{\max}(\sigma) = \max_{q\in\nu(\sigma)} L(q)$.*

Proof. Let us consider an active feasible schedule of maximum lateness bounded by $\overline{L}$. Let us denote by $C(\sigma)$ the length of the schedule σ, i.e. $C(\sigma) = \max_{i \in \mathcal{T}}(t_i^\sigma + 1)$. Clearly, since σ is active, $C(\sigma) \leq \overline{C}$. For $\alpha \in \{0, \ldots, C(\sigma) - 1\}$, we set $W_\alpha = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma$ and $B_\alpha = \mathcal{T}_\alpha^\sigma$.

The set $A(\emptyset, \emptyset)$ contains the maximum sets of tasks schedulable at time 0. Since σ is a feasible active schedule, there exists $q_0 \in N_0$ such that $W(q_0) = B(q_0) = W_0 = B_0$. Moreover, $(s, q_0) \in A$ and thus *A.3* is verified.

Moreover, since σ is feasible, for every value $\alpha \in \{0, \ldots, C(\sigma) - 1\}$, $\mathcal{T}_\alpha^\sigma \subseteq X_\alpha$. According to Lemma 3.2.3, $\bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma - Z_\alpha \subseteq X_\alpha \cap X_{\alpha+1}$. So the node $q_\alpha = (W(q_\alpha), B(q_\alpha), +\infty)$ has been built at stage α in the loop of lines 5-6 of Algorithm 2.

We prove that for every value $\alpha \in \{0, \ldots, C(\sigma) - 2\}$, $(q_\alpha, q_{\alpha+1}) \in A$.

- $W(q_{\alpha+1}) = \bigcup_{\beta=0}^{\alpha+1} \mathcal{T}_\beta^\sigma = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma \cup \mathcal{T}_{\alpha+1}^\sigma = W(q_\alpha) \cup B(q_{\alpha+1})$. Moreover, $W(q_\alpha) \cap B(q_{\alpha+1}) = \bigcup_{\beta=0}^{\alpha} \mathcal{T}_\beta^\sigma \cap \mathcal{T}_{\alpha+1}^\sigma = \emptyset$. Thus, *A.1* is verified.

- Since σ is a feasible active schedule, *A.2* is verified.

We conclude that $(s, q_0, q_1, \ldots, q_{C(\sigma)-1})$ is a path of $S(\mathcal{G})$. Moreover, $W(q_{C(\sigma)-1}) = \mathcal{T}$ since the schedule σ ends at time $C(\sigma)$, thus $p = q_{C(\sigma)-1}$ is an ending node.

Lastly, by Lemma 5.3.1, each value $L(q_\alpha)$ computed by Algorithm 2 for $\alpha \in \{0, \ldots, C(\sigma)-1\}$ is the minimum maximum lateness of the sub-schedule associated to q_α; the maximum lateness of the schedule σ is thus $L_{\max}(\sigma) = \max_{q \in \nu(\sigma)} L(q)$, which concludes the proof. □

Lemma 5.3.3. *Let $(s, p_0, p_1, \ldots, p_{C-1})$ be a path of $S(\mathcal{G})$ with $W(p_{C-1}) = \mathcal{T}$. For each task $i \in \mathcal{T}$, there exists a unique value $\alpha \in \{0, \ldots, C-1\}$ such that $i \in B(p_\alpha)$.*

Proof. According to the definition of $S(\mathcal{G})$, $W(p_0) \subseteq W(p_1) \subseteq \cdots \subseteq W(p_{C-1})$. Moreover, by assumption, $W(p_{C-1}) = \mathcal{T}$. Thus, for each task $i \in \mathcal{T}$, there is a unique $\alpha \in \{0, \ldots, C-1\}$ with $i \in W(p_\alpha)$ and $i \notin W(p_{\alpha-1})$. Since $W(p_{\alpha-1}) \cup B(p_\alpha) = W(p_\alpha)$, we get $i \in B(p_\alpha)$. □

Lemma 5.3.4. *Each node $p \in N$ such that $W(p) = \mathcal{T}$ is associated to an active feasible schedule σ of maximum lateness $L_{\max}(\sigma) = L(p)$.*

Proof. Let us consider a node $p \in N_{C-1}$ with $W(p) = \mathcal{T}$ and $C \in \{0, \ldots, \overline{C} - 1\}$. We build iteratively a sequence of nodes $p_{-1}, p_0, p_1, \ldots, p_{C-1}$ of $S(\mathcal{G})$ as follows:

1. $p_{C-1} = p$;
2. for each $k \in \{1, \ldots C-1\}$, p_{k-1} is a predecessor of p_k in $S(\mathcal{G})$ such that $L(p_{k-1})$ is minimum;
3. $p_{-1} = s$.

This sequence is defined since each node of N_α with $\alpha \in \{0, \ldots, \overline{C} - 1\}$ has at least one predecessor (or it will be deleted at line 18). Moreover, $p_k \in N_k$ for $k \in \{-1, \ldots C-1\}$.

Now, by Lemma 5.3.3, for each task $i \in \mathcal{T}$, there exists a unique value $\alpha \in \{0, \ldots, C-1\}$ with $i \in B(p_\alpha)$. Thus, a starting time can be defined for i by setting $t_i^\sigma = \alpha$. We prove in the following that these starting times define an active feasible schedule σ.

We first observe that for each value $\alpha \in \{-1, \ldots, C-1\}$, $B(p_\alpha) \subseteq X_\alpha$ and $|B(p_\alpha)| \leq m$ following the condition *N.2*. Thus, the maximum lateness of each task is bounded by $\bar{L}$; the constraints (2) and (7) of the problem definition are fulfilled.

Now, let consider a task $i \in B(p_\alpha)$ with $\alpha \in \{0, \ldots, C-1\}$. By condition *A.2*, i is schedulable at time α. Thus, all its predecessors are belonging to $W(p_{\alpha-1})$ and the condition (4) of the problem definition is verified.

Moreover, $B(p_\alpha) \in A(W(p_{\alpha-1}), B(p_{\alpha-1}))$. Thus, there is at least one predecessor j of i scheduled at time $\alpha - 1$ and j has no other successor scheduled at time α. Thus, conditions (5) and (6) of the problem definition are validated.

Lastly, elements from $A(W(p_\alpha), B(p_\alpha))$ for $\alpha \in \{0, \ldots, C-1\}$ are maximum for the inclusion; this condition guarantees that σ is an active schedule.

Now, by definition of the sequence p_k, $L(p_k) = \max(\ell(p_k), L(p_{k-1}))$ with $\ell(p_k) = k + 1 - \min_{i \in B(p_k)} d_i$. Thus, $L(p_0) \leq L(p_1) \ldots \leq L(p_{C-1})$. By Lemma 5.3.1, the maximum lateness of the schedule σ is $L_{\max}(\sigma) = L(p_{C-1}) = L(p)$ and the lemma is proved. □

Theorem 5.3.5 (Correctness of Algorithm 2)**.** *Algorithm 2 returns the minimum maximum lateness $L_{\max}(\sigma) \leq \bar{L}$ of a feasible schedule σ if it exists, false if there is no feasible schedule of maximum lateness bounded by $\overline{L}$.*

Proof. Let us suppose first that Algorithm 2 returns a value $L^\star$. Then, let a node $p \in N$ such that $L^\star = L(p) = \min\{L(p), p \in N, W(p) = \mathcal{T}\}$. By Lemma 5.3.4, p is associated to an active feasible schedule of maximum lateness $L(p)$, thus the minimum maximum lateness of our instance $L_{opt} \leq L^\star$. Now, let us suppose by contradiction that $L^\star > L_{opt}$. Thus, there exists an active feasible schedule σ such that $L_{\max}(\sigma) = L_{opt}$ and σ is not associated to a path of $S(\mathcal{G})$, which contradicts Lemma 5.3.2, and thus $L^\star = L_{opt}$.

Now, let us suppose that there is no node $p \in N$ such that $W(p) = \mathcal{T}$; in this case, Algorithm 2 returns false. By Lemma 5.3.2, there is no active feasible schedule and the theorem is proved. □

5.4 Complexity analysis

We study in this section the complexity of Algorithm 2 to conclude that our scheduling problem is fixed-parameter tractable in the pathwidth.

Lemma 5.4.1. *Let us denote by n the number of tasks and $pw(\overline{L})$ the pathwidth associated to the upper bound $\overline{L}$ of the maximum lateness. For any value $\alpha \in \{0, \ldots, \overline{C} - 1\}$, the number of elements of N_α belongs to $\mathcal{O}(2^{2pw(\overline{L})})$.*

Proof. By Algorithm 2, the number of nodes in N_α for $\alpha \in \{0, \ldots, \overline{C} - 1\}$ is bounded by $2^{|X_\alpha|} \times 2^{|X_{\alpha+1}|}$. The values $|X_\alpha|$ and $|X_{\alpha+1}|$ are both bounded by $pw(\overline{L}) + 1$, thus the lemma holds. □

Lemma 5.4.2. *The time complexity of the inner loop of Algorithm 2 (Lines 8-17) for a fixed node $p \in N_\alpha$ and $\alpha \in \{-1, \ldots, \overline{C} - 2\}$ is $\mathcal{O}(pw(\overline{L}) \times 2^{pw(\overline{L})})$.*

Proof. Let us suppose that $W(p) \neq \mathcal{T}$. By definition, $A(W(p), B(p)) \subseteq \mathcal{P}(X_{\alpha+1})$ and thus $|A(W(p), B(p))| \leq 2^{|X_{\alpha+1}|}$.

Searching for a node q in $N_{\alpha+1}$ can be done in time $\mathcal{O}(\log |N_{\alpha+1}|)$; By Lemma 5.4.1, $\mathcal{O}(\log |N_{\alpha+1}|) \subseteq \mathcal{O}(pw(\overline{L}))$, thus the overall time of the inner loop is $\mathcal{O}(pw(\overline{L}) \times 2^{pw(\overline{L})})$, and the lemma is proved. □

Next lemma bounds the maximum value of the release dates and $\overline{C}$:

Lemma 5.4.3. *We can suppose that* $r_{max} = \max_{i \in \mathcal{T}} r_i \leq 2(n-1)$ *and* $\overline{C} \leq 4n - 2$ *without loss of generality.*

Proof. Let suppose by contradiction that $r_n > 2(n-1)$, and let $i^\star$ be the smallest value $i \in \{1, \ldots, n\}$ with $r_i > 2(i-1)$; since $r_1 = 0$, we get that $i^\star > 1$. For every task $j \in \{1, \ldots, i^\star - 1\}$, $r_j \leq 2(j-1)$.

The biggest values for the release date of tasks j in $\{1, \ldots, i^\star - 1\}$ is $r'_j = 2(j-1)$ and the most constrained instance is a path $1 \rightarrow 2 \ldots \rightarrow i^\star - 1$. In this case, the only active feasible schedule σ is $t^\sigma_j = 2(j-1)$. Thus, any active feasible schedule of tasks $\{1, \ldots, i^\star - 1\}$ would end at time $2(i^\star - 2) + 1$ or before. Since $r_{i^\star} \geq 2(i^\star - 1) + 1$, there will be at least two idle time slots at time $2(i^\star - 2) + 1$ and $2(i^\star - 1)$ before the beginning of tasks in $\{i^\star, i^\star + 1, \ldots n\}$. Since release times are compatible with respect to precedence, we can treat separately the two sets of tasks $\{1, \ldots, i^\star - 1\}$ and $\{i^\star, \ldots, n\}$. The second scheduling problem considers tasks $\{i^\star, \ldots, n\}$ with release times $\tilde{r}_j = r_j - r_{i^\star}$ and due dates $\tilde{d}_j = d_j - r_{i^\star}$. Thus, the first part of the lemma is proved.

Now, since $\overline{C} = \min(r_{max} + 2n, d_{max} + \overline{L})$, $\overline{C} \leq 4n - 2$ and the lemma is proved. □

Theorem 5.4.4 (Complexity of Algorithm 2)**.** *The time complexity of Algorithm 2 is* $\mathcal{O}(n^2 + n \times pw(\overline{L}) \times 2^{3pw(\overline{L})})$*, where* $pw(\overline{L})$ *is the pathwidth of the interval graph associated to the time windows* $(r_i, d_i + \overline{L})$*,* $i \in \mathcal{T}$*.*

Proof. The time complexity of the computation of the sets X_α and Z_α for $\alpha \in \{0, \ldots, \overline{C}\}$ (lines 3-4) is $\mathcal{O}(n^2)$ since $\overline{C}$ is bounded by $4n - 2$ following Lemma 5.4.3.

The time complexity for building N at lines 7-8 is $\mathcal{O}(n \times 2^{2pw(\overline{L})})$ by Lemmas 5.4.1 and 5.4.3. Following Lemma 5.4.2, the whole complexity of building arcs of $S(\mathcal{G})$ in lines 9-20 is $\mathcal{O}(n \times 2^{2pw(\overline{L})} \times pw(\overline{L}) \times 2^{pw(\overline{L})})$.

The overall complexity of the algorithm is thus $\mathcal{O}(n^2 + n \times pw(\overline{L}) \times 2^{3pw(\overline{L})})$, and the theorem holds. □

5.5 Conclusion and perspectives

We proved that the scheduling problem $P|r_i, prec, p_i = 1, c_{ij} = 1|L_{\max}$ is fixed-parameter tractable in the pathwidth $pw(\bar{L})$ associated to an upper bound $\bar{L}$ of the maximum lateness. We extended previous approaches [72, 84] to tackle both communications delay, a limited number of machine, and to optimize the maximum lateness. We also limit our enumeration to active schedules, which allows to decrease the worst-case complexity of the method.

We believe that this work opens up many perspectives. From a theoretical point of view, many fundamental questions remain open as the existence of a fixed-parameter algorithm in the width, or the possible extension of this work to scheduling problems with large communication delays. From a practical point of view, our algorithm defines an original exploration scheme probably well suited to general scheduling problems. Similarly to Branch-And-Bound methods, dominance properties allow to reduce the size of the generated multistage graph. It would then be interesting to test this new class of algorithms to compare their performance with those from the literature.

6. Evaluation of the Age Latency of a Real-Time Communicating System using the LET paradigm

A real-time system (RTS) is a system that responds to external events created by its environment in a timely fashion [65]. It has been developed and has been researched in demand in the market especially in industrial environments[4, 30]. Typical examples include Air Traffic Control Systems, Networked Multimedia Systems, Command Control Systems etc. In a Real-Time System the quality of the system services depend not only on the correctness of the computations, but also on the instant at which the computation results are obtained. It is expected to have the correct computation at the correct time. The responses have specified constraints or deadlines, there are two types of real-time systems according to the timing constraints:

1. Hard real-time system: This type of system does not allowed missed deadlines such as avionics or automotive control systems. These systems must verify hard timing constraints, a missed deadline can cause system failure or disastrous consequences. Their design and analysis are usually a complex processes that require efficient methods, for example, automated piloting systems.

2. Soft real-time system: In this type of system, missed deadline is allowed with acceptable low probability or tardiness, for example the telephone switches systems, audio and video steaming software for entertainment (lag will cost the quality of service but not be a disaster)

A real-time system usually communicates with its environment through sensors that detect events and actuators that traduce its reaction. Paths from a sensor to an actuator are usually referred to event chains (see as example [47]). The time needed to propagate a data from a sensor to an actuator is closely related to the reaction delay of the system. One of the most important features is that the response time of a real-time system must be predictable and limited. Several measures can be defined to capture these delays, as presented by Feiertag et al.[38]. We limit out study to

the age latency, also called the end-to-end latency, which is the maximum time interval from a specific input value on a sensor and the last corresponding output value. It can be interpreted as the maximum delay that a specific data spends in the system. This value measures the freshness of a data producing a response of the system, and insures that the action of the actuator is not too old.

Real-time computer systems are often associated with low-latency systems. Many applications of real-time computing are also low-latency applications. However, a hard real-time system must be guaranteed that the system finishes a certain task by a certain time. Therefore, it is important that the latency in the system be measurable and a maximum allowable latency for tasks be set.

A real-time system is a system that responds to external events created by its environment in a timely fashion [65]. In various contexts such as avionics or automotive, these systems must verify hard timing constraints. Their design and analysis are usually a complex processes that require efficient methods.

We consider a set $\mathcal{T}$ of periodic tasks with different periods that are executed following the model of Liu and Layland [66]. A directed acyclic graph $\mathcal{G} = (\mathcal{T}, E)$ defines communication links between tasks executions. Each arc $(t_i, t_j) \in E$ between the two tasks t_i and t_j is associated to a shared memory variable that is modified by t_i and read by t_j. We assume that each execution of t_i updates the variable at its completion time, while each execution of t_j read it at its starting time. This communication scheme, usually known as "implicit communication" follows the AUTOSAR requirement [1] and is commonly used for the design of automotive real-time systems.

However, the instants of the exchanges between tasks depend on the successive starting times and completion times of the tasks, and are thus not predictable. Logical Execution Time (LET in short) paradigm [59] delays writings to the periodic deadlines of the tasks and advances the reading to their periodic release dates. The communication instants are then fixed before the execution of the tasks and the system is deterministic. This communication scheme was implemented by the time-triggered language Giotto [52]. This timing predictability makes it particularly suitable for safety-critical applications. This model was thus considered in industrial domains as automotive [11, 47] or avionics [53, 90]. We suppose in this chapter that tasks are periodic with different

periods and that all communications follow LET paradigm.

The major contribution of the chapter is to develop and to prove a general framework to model the communications on the successive tasks executions using LET communications for a general graph. The computation of the age latency of the application can then be seen as an example of concrete application. Observe that most of the authors limited their methods to chains [9, 38, 69] and that our methodology can handle easily a general graph.

Indeed, we first prove that dependencies induces by a LET communication $e = (t_i, t_j) \in E$ between the successive executions of t_i and t_j can be modelled by an original simple inequality involving parameters of the tasks t_i and t_j and the execution numbers considered.

Then, it can easily be observed that, if T_i denotes the period of task t_i, these dependence relations between tasks executions are repeated within the hyper-period $T = lcm_{t_i \in \mathcal{T}}(T_i)$. An expanded valued graph $P_N(\mathcal{G})$ can then be easily built by duplicating each task $N_i = \frac{T}{T_i}$ times. We prove in this chapter that setting any vector K with $K_i \in \{1, \ldots, N_i\}$ for any $t_i \in \mathcal{T}$, a partial expanded graph $P_K(\mathcal{G})$ can be built by duplicating each task K_i times. Each arc of this graph includes the modelling of the dependence relation between the corresponding executions of its adjacent tasks duplicates. This partial expanded graph is inspired from Bodin et al. [12] and de Groote [31] for Synchronous DataFlow Graph [62], for which the initial inequality modelling dependence is slightly different.

Subsequently, we show that upper bounds of the latency between adjacent duplicates of $P_K(\mathcal{G})$ can be derived and considered as a valuation of the arcs. The longest paths of $P_K(\mathcal{G})$ provide then an upper bound of the latency. However, the computation of these paths has a time complexity proportional to $\sum_{e=(t_i,t_j)\in E} K_i \times K_j$. The main problem is then to find the values of K that minimises this function with an exact evaluation of the age latency.

We first prove that our study can be limited to vectors K such that, for any task t_i, K_i divides N_i. We then develop a greedy algorithm that converges to a vector $K^\star$ that provides the exact value of the age latency. This algorithm can be seen as an adaptation of the K-iter algorithm [13] for the determination of the maximum throughput of a Synchonous DataFlow Graph, which is up to now

one of the best algorithm to solve this latter problem. Our algorithm was experimentally tested on random generated graphs with periods inspired from automotive real-life benchmarks [48, 61].

This chapter is organised as follows. Section 6.1 presents related works. The problem and our characterisation of the dependence between tasks executions are presented in Section 6.2. Section 6.3 is devoted to the construction of the partial expanded graph $P_K(\mathcal{G})$ for any fixed vector K. It is shown in Section 6.4 that our exploration can be limited to K vectors such that, for any task $t_i \in \mathcal{T}$, K_i is a divisor of N_i. Section 6.5 presents our greedy algorithm for the computation of a vector $K^\star$ leading to an optimum value of the age latency. This algorithm is experimented in Section 6.6 on a the Case study ROSACE [76] and in Section 6.7 on randomly generated graphs. Section 6.8 is our conclusion.

6.1 Related works

The evaluation of the age latency of an event chain is a challenging question tackled by several authors. Feirtag et al. [38] first introduced the definition of dependence between tasks of an event chain and four metrics to evaluate the delay between a sensor and an actuator. Becker et al. [9] developed a general framework to evaluate the age latency of an event chain using feasible intervals. They built an expanded graph by evaluating the possible propagation of an input data by the successive execution of tasks. They tested in [10] their approach against the evaluation of the latency of a fixed schedule or under the LET hypothesis. They concluded that if there is no information on the communications nor on the schedule, a pessimistic value of the age latency will be obtained, which is very similar to the value obtained using LET paradigm. However, the computation time grows exponentially with the number of tasks if an enumeration is needed, while it remains constant for LET paradigm.

Under the LET assumption, times of the communications between tasks are known before the executions of the tasks. This strong assumption allows to characterise the dependencies between tasks if their parameters are fixed. Martinez et al. [69] gave a formal characterisation of the dependencies between tasks in an event chain using time instants. They then derived the age

latency by enumerating all the possible paths of the corresponding expanded graph. They also proved that the release times influence the age latency and they developed a heuristic to fix them in order to minimise it.

All these approaches cannot be extended easily to a general graphs. Indeed, the number of paths between two vertices is exponential. The complexity of a method that enumerates all the paths for evaluating their age latency will thus grow exponentially following the parameters of the graph.

Forget et al.[41] has developed a language to express the constraints and a multi-periodic synchronous model to represent the whole system for a general graph. This approach supports several metrics. However, the complexity of the method to evaluate the different latencies is equivalent to building the expanded graph.

Khatib et al. [58] proved that constraints between the successive executions of two adjacent tasks can be modelled using a Synchronous DataFlow Graph [62]. Our equation is slightly different since for any arc $e = (t_i, t_j)$, they do not considered the successive constraints between two adjacent tasks if $T_i > T_j$, dealing only with precedence constraints. They then compute the age latency using the expansion of the Synchronous DataFlow graph which is equivalent to $P_N(\mathcal{G})$. They also proposed the computation of a polynomial upper bound of the age latency equivalent to the determination of the longest paths of $P_{\mathbb{1}^n}\mathcal{G}$ with $n = |\mathcal{T}|$. Lastly, they showed that the difference between this bound and the age latency is around 30 percent. This result motivates the development of efficient methods to evaluate more precisely the age latency of a graph $\mathcal{G}$.

6.2 Modelling of the system

This section presents formally the problem tackled in this chapter. Subsection 6.2.1 defines the periodic tasks model considered according to LET restrictions. Subsection 6.2.2 is dedicated to the definition of the dependence relation between the successive executions of two adjacent tasks. Subsection 6.2.3 defines formally the age latency of a graph. Subsection 6.2.4 is devoted to the definition of the problem and the presentation of a small pedagogic example.

6.2.1 Periodic tasks model considering LET communications

Let us consider a set $\mathcal{T} = \{t_1, \ldots, t_n\}$ of real-time periodic tasks following the model of Liu and Layland [66]. Each task $t_i \in \mathcal{T}$ is characterised by a quadruplet (r_i, C_i, D_i, T_i) such that:

- r_i is the release date (the offset) of the first execution of t_i;
- C_i is the worst-case execution time of t_i;
- D_i is the relative deadline of t_i;
- T_i is the period of t_i.

For any value $n \in \mathbb{N}^\star$, we denote by $\langle t_i, n\rangle$ the nth execution of t_i and by $s(t_i, n)$ its starting time. For any value $n \in \mathbb{N}^\star$, the execution of $\langle t_i, n\rangle$ must be scheduled in its time window, that is $r_i + (n-1) \times T_i \leq s(t_i, n)$ and $s(t_i, n) + C_i \leq D_i + (n-1) \times T_i$.

Logical execution time communication model separates the tasks executions to the communications. In this model, data are read at the release dates of the reading tasks, while they are sent at the deadlines of its sending task. Moreover, reading tasks always get the last emitted data. The main advantage of this model is to define a deterministic communications system even if tasks are delayed inside their time window.

In this chapter, we only consider LET communications and we limit the characterization of the tasks to their successive time windows. The execution times associated to the nth execution of t_i is then set to its release date, that is $\mathcal{S}(t_i, n) = r_i + (n-1) \times T_i$. Similarly, the completion time is fixed to $\mathcal{S}(t_i, n) + D_i$. Each task t_i is then given by the triplet (r_i, D_i, T_i).

6.2.2 LET dependencies

Communications are expressed by a directed graph $\mathcal{G} = (\mathcal{T}, E)$. Every arc $e = (t_i, t_j) \in E$ is associated to a LET communication from t_i to t_j. A dependence between two executions of adjacent tasks is defined as follows:

Definition 6.2.1. Let us suppose that $e = (t_i, t_j) \in E$ and that ν_i and ν_j are two positive integers. There exists a dependence relation from $\langle t_i, \nu_i\rangle$ to $\langle t_j, \nu_j\rangle$ if $\langle t_j, \nu_j\rangle$ gets data from $\langle t_i, \nu_i\rangle$.

Figure 6.1 presents successive time windows of the first executions of two periodic tasks t_1 and t_2 with a LET communication $e = (t_1, t_2) \in E$. Since $T_1 > T_2$ a same data from t_1 can be read by several executions of t_2.

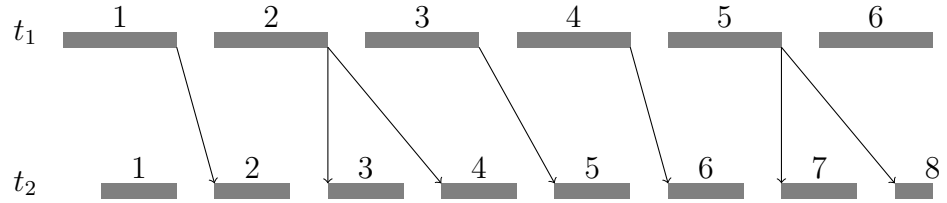


Figure 6.1: Time windows associated to two periodic tasks t_1 and t_2 with a LET dependence $e = (t_1, t_2)$. Parameters of tasks are respectively $(r_1, D_1, T_1) = (0, 3, 4)$ and $(r_2, D_2, T_2) = (1, 2, 3)$.

Next theorem characterises formally the dependence relation between the executions of two communicating tasks:

Theorem 6.2.2. *Let suppose that* $e = (t_i, t_j) \in E$. *Let* $gcd_T^e = \gcd(T_i, T_j)$ *and* $M^e = T_j + \lceil \frac{r_i - r_j + D_i}{gcd_T^e} \rceil \times gcd_T^e$. *For any couple* $(\nu_i, \nu_j) \in \mathbb{N}^\star \times \mathbb{N}^\star$, *there exists a dependence from* $\langle t_i, \nu_i \rangle$ *to* $\langle t_j, \nu_j \rangle$ *iff* $T_i \geq M^e + T_i\nu_i - T_j\nu_j > 0$.

Proof. There exists a dependence from $\langle t_i, \nu_i \rangle$ to $\langle t_j, \nu_j \rangle$ iff the two following conditions hold:

1. $\langle t_j, \nu_j \rangle$ begins after the end of $\langle t_i, \nu_i \rangle$, thus $\mathcal{S}(t_i, \nu_i) + D_i \leq \mathcal{S}(t_j, \nu_j)$. Since $\mathcal{S}(t_i, \nu_i) = r_i + (\nu_i - 1) \times T_i$ and $\mathcal{S}(t_j, \nu_j) = r_j + (\nu_j - 1) \times T_j$, we get

$$r_i + (\nu_i - 1) \times T_i + D_i \leq r_j + (\nu_j - 1) \times T_j,$$

thus,

$$T_i \geq T_j + (r_i - r_j + D_i) + T_i\nu_i - T_j\nu_j,$$

and since in the inequality above only $r_i - r_j + D_i$ can't be divided by gcd_T^e, we obtain that

$$T_i \geq M^e + T_i\nu_i - T_j\nu_j.$$

2. At the beginning of $\langle t_j, \nu_j \rangle$, data from $\langle t_i, \nu_i + 1 \rangle$ are not available, thus $\mathcal{S}(t_i, \nu_i + 1) + D_i > \mathcal{S}(t_j, \nu_j)$ and then

$$r_i + \nu_i T_i + D_i > r_j + (\nu_j - 1) \times T_j,$$

thus,

$$T_j + (r_i - r_j + D_i) + T_i \nu_i - T_j \nu_j > 0.$$

Since $M^e \geq T_j + (r_i - r_j + D_i)$

$$M^e + T_i \nu_i - T_j \nu_j > 0.$$

Merging the two equations, we get the theorem. □

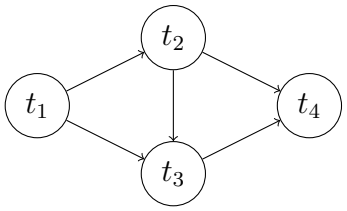
Let us consider as example the two tasks t_1 and t_2 with the LET communication $e = (t_1, t_2)$ presented by Figure 6.1. We get $\gcd_T^e = \gcd(3, 4) = 1$ and $M^e = 3 + (0 - 1 + 3) = 5$. The inequality of Theorem 6.2.2 is $4 \geq 5 + 4\nu_1 - 3\nu_2 \geq 0$. One can observe that the first executions of t_1 and t_2 with a dependence relation correspond to the couples that verify this inequality. For $(\nu_1, \nu_2) = (1, 2)$, we get $5+4\nu_1-3\nu_2 = 5+4-6 = 3 \in \{1, \ldots, 4\}$. Similarly, for $(\nu_1, \nu_2) = (2, 3)$, we get $5 + 4\nu_1 - 3\nu_2 = 5 + 8 - 9 = 4 \in \{1, \ldots, 4\}$. Now, if we consider $(\nu_1, \nu_2) = (2, 5)$, $5 + 4\nu_1 - 3\nu_2 = 5 + 8 - 15 = -2 \notin \{1, \ldots, 4\}$ and there is no dependence from $\langle t_1, 2 \rangle$ to $\langle t_2, 5 \rangle$.

6.2.3 Age latency

Let us suppose that $e = (t_i, t_j) \in E$ and let the set $\mathcal{R}(e)$ be the couples $(\nu_i, \nu_j) \in \mathbb{N}^\star \times \mathbb{N}^\star$ such that e induces a dependence from $\langle t_i, \nu_i \rangle$ to $\langle t_j, \nu_j \rangle$. Then, for any couple $(\nu_i, \nu_j) \in \mathcal{R}(e)$, the latency of the executions $\langle t_i, \nu_i \rangle$ and $\langle t_j, \nu_j \rangle$ associated to e is

$$\mathcal{L}_{\nu_i, \nu_j}(e) = \mathcal{S}(t_j, \nu_j) - \mathcal{S}(t_i, \nu_i) = r_j - r_i + T_i - T_j - (T_i \nu_i - T_j \nu_j). \tag{6.2.1}$$

Now, for any path $p = t_1 e_1 t_2 e_2 \ldots e_{k-1} t_k$ of $\mathcal{G}$, let us define the set $\mathcal{R}(p)$ as the k-uplets $(\nu_1, \ldots, \nu_k) \in \mathbb{N}^{\star k}$ such that $\forall \ell \in \{1, \ldots, k-1\}$, $(\nu_\ell, \nu_{\ell+1}) \in \mathcal{R}(e_\ell)$. Then, for any k-uplet



t_i	t_1	t_2	t_3	t_4
r_i	0	1	2	3
D_i	1	0.5	4	3
T_i	2	1	6	3

Figure 6.2: An instance of 4 periodic tasks and the associated DAG $\mathcal{G}$.

$(\nu_1, \ldots, \nu_k) \in \mathcal{R}(p)$, we get

$$\mathcal{L}_{\nu_1,\ldots,\nu_k}(p) = \sum_{\ell=1}^{k-1} \mathcal{L}_{\nu_\ell,\nu_{\ell+1}}(e_\ell) + D_k.$$

The maximum latency of a path p of $\mathcal{G}$ is then defined as

$$\mathcal{L}^\star(p) = \max\{\mathcal{L}_{\nu_1,\ldots,\nu_k}(p), (\nu_1, \ldots, \nu_k) \in \mathcal{R}(p)\}$$

and the maximum latency of a directed graph $\mathcal{G}$ corresponds to

$$\mathcal{L}^\star(\mathcal{G}) = \max\{\mathcal{L}^\star(p), p \text{ path of } \mathcal{G}\}.$$

Let us observe that, if the initial graph $\mathcal{G}$ contains circuits, its latency may be not bounded. So, we suppose in the following that $\mathcal{G}$ is acyclic. Moreover, since the latency between two executions is positive, $\mathcal{L}^\star(\mathcal{G})$ is reached for a path p such that t_1 has no predecessor and t_k no successor.

6.2.4 Problem definition and example

The problem tackled in this chapter can be formalised as follows: let us consider a directed acyclic graph $\mathcal{G} = (\mathcal{T}, E)$, each arc modelling a LET communication. Each periodic task $t_i \in \mathcal{T}$ is associated to a triplet (r_i, D_i, T_i). The problem is to compute the maximal age latency $\mathcal{L}^\star(\mathcal{G})$.

Figure 6.2 presents an instance of our problem composed by 4 periodic tasks and the associated directed acyclic graph $\mathcal{G}$. Dependence relations between the first executions of tasks t_1, t_2 and t_4

are pictured by Figure 6.3.following the path $p = t_1 t_2 t_4$ of $\mathcal{G}$. The latency of the path from $\langle t_1, 1\rangle$ to $\langle t_4, 1\rangle$ is $\mathcal{L}_{1,2,1}(p) = \mathcal{S}(t_4, 1) - \mathcal{S}(t_1, 1) + 2 = 3 - 0 + 2 = 5$. On the same way, the latency of the path p from $\langle t_1, 3\rangle$ to $\langle t_4, 2\rangle$ is $\mathcal{L}_{3,5,2}(p) = \mathcal{S}(t_4, 2) - \mathcal{S}(t_1, 3) + 2 = 6 - 4 + 2 = 4$. We deduce that $\mathcal{L}^\star(p) = 5$.

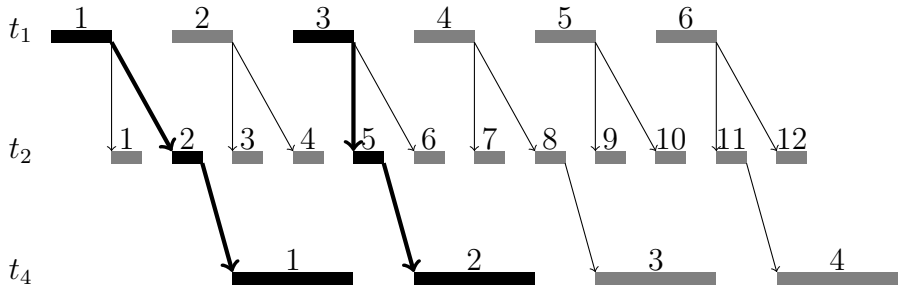


Figure 6.3: A path $p = t_1 e_1 t_2 e_2 t_4$ from the graph $\mathcal{G}$ pictured by Figure 6.2.

6.3 Construction of a partial expanded graph

The aim of this section is to detail and prove the construction of a partial expanded graph $P_K\mathcal{G}$ associated to a fixed vector K. The main idea is to duplicate any task t_i K_i times and to express the dependence directly on duplicates.

Subsection 6.3.1 is devoted to the proof of Theorem 6.3.3 that characterises the dependence relations between the duplicates of two adjacent tasks. An upper bound of the latency between two duplicates corresponding to dependant executions is then evaluated in Subsection 6.3.2. Subsection 6.3.3 defines formally the partial expanded graph $P_K\mathcal{G}$ associated with a vector K, while subsection 6.3.4 evaluates the complexity its computation.

6.3.1 Dependence between duplicates of the partial expanded graph

Let us suppose that for any task t_i, a number of duplicates $K_i \in \mathbb{N}^\star$ is fixed. Then, for any $a_i \in \{1, \ldots, K_i\}$, the a_ith duplicate of t_i is simply associated to the executions $a_i + pK_i$ for $p \in \mathbb{N}$. Next technical lemma characterises the dependence relation between two executions of adjacent

tasks taking into account the number of duplicates of the tasks:

Lemma 6.3.1. *Let $e = (t_i, t_j) \in E$ and gcd_T^e (resp. gcd_K^e) the greatest common divisor between T_i and T_j (resp. K_iT_i and K_jT_j). Let $\nu_i = a_i + p_iK_i$ and $\nu_j = a_j + p_jK_j$ with $(a_i, a_j) \in \{1, \ldots, K_i\} \times \{1, \ldots, K_j\}$ and $(p_i, p_j) \in \mathbb{N} \times \mathbb{N}$. Let us define the four values*

- $\alpha_e(a_i, a_j) = \dfrac{T_i a_i - T_j a_j}{gcd_T^e}$,
- $\pi_e(p_i, p_j) = \dfrac{T_i p_i K_i - T_j p_j K_j}{gcd_K^e}$,
- $\pi_e^{max}(a_i, a_j) = \left\lfloor \dfrac{-M^e + T_i - \alpha_e(a_i, a_j) \cdot gcd_T^e}{gcd_K^e} \right\rfloor$,
- *and* $\pi_e^{min}(a_i, a_j) = \left\lceil \dfrac{-M^e + gcd_T^e - \alpha_e(a_i, a_j) gcd_T^e}{gcd_K^e} \right\rceil$.

If e induces a dependence from $\langle t_i, \nu_i \rangle$ to $\langle t_j, \nu_j \rangle$, then

$$T_i\nu_i - T_j\nu_j = \pi_e(p_i, p_j) \cdot gcd_K^e + \alpha(a_i, a_j) \cdot gcd_T^e$$

with $\pi_e(p_i, p_j) \in \{\pi_e^{min}(a_i, a_j), \ldots, \pi_e^{max}(a_i, a_j)\}$.

Proof. By definition of ν_i and ν_j,

$$\begin{aligned} T_i\nu_i - T_j\nu_j &= T_i \times (a_i + K_i \cdot p_i) - T_j \times (a_j + K_j \cdot p_j) \\ &= (T_iK_ip_i - T_jK_jp_j) + (T_ia_i - T_ja_j) \\ &= \pi_e(p_i, p_j) \cdot gcd_K^e + \alpha_e(a_i, a_j) \cdot gcd_T^e. \end{aligned}$$

By Theorem 6.2.2, $T_i - M^e \geq T_i\,\nu_i - T_j\nu_j > -M^e$. Thus, since all the terms of this inequality are divisible by gcd_T^e, its right part is equivalent to $T_i - M^e \geq T_i\,\nu_i - T_j\nu_j \geq -M^e - gcd_T^e$ and we get

$$T_i - M^e \geq \pi_e(p_i, p_j) \cdot gcd_K^e + \alpha_e(a_i, a_j) \cdot gcd_T^e \geq -M^e + gcd_T^e.$$

From the right part of the inequality,

$$\pi_e(p_i, p_j) \geq \frac{-M^e + gcd_T^e - \alpha_e(a_i, a_j) \cdot gcd_T^e}{gcd_K^e}.$$

Since $\pi_e(p_i, p_j)$ is an integer, we can tighter the lower bound of $\pi_e(p_i, p_j)$ by

$$\pi_e(p_i, p_j) \geq \lceil \frac{-M^e + gcd_T^e - \alpha_e(a_i, a_j) \cdot gcd_T^e}{gcd_K^e} \rceil = \pi_e^{min}(a_i, a_j).$$

On the same way, the left part of the previous inequality is

$$\frac{T_i - M^e - \alpha_e(a_i, a_j) \cdot gcd_T^e}{gcd_K^e} \geq \pi_e(p_i, p_j)$$

Since $\pi_e(p_i, p_j)$ is an integer, we can tighter the upper bound of $\pi_e(p_i, p_j)$ by:

$$\lfloor \frac{T_i - M^e - \alpha_e(a_i, a_j) \cdot gcd_T^e}{gcd_K^e} \rfloor \geq \pi_e(p_i, p_j)$$

So we get $\pi_e^{max}(a_i, a_j) \geq \pi_e(p_i, p_j)$ and the lemma is proved. □

Let consider as example the arc $e = (t_2, t_4)$ of the example pictured by Figure 6.2 with fixed values $K_2 = 4$ and $K_4 = 2$. We get $gcd_T^e = \gcd(1, 3) = 1$, $gcd_K^e = \gcd(4, 6) = 2$ and $M^e = 3 + \lceil \frac{1-3+0.5}{1} \rceil = 2$. The corresponding values of $\alpha_e(a_i, a_j)$, $\pi_e^{max}(a_i, a_j)$ and $\pi_e^{min}(a_i, a_j)$ are reported by Table 6.1.

For the couple $(a_2, a_4) = (3, 2)$, let suppose that there exists a dependence from $\langle t_2, \nu_2 \rangle$ to $\langle t_4, \nu_4 \rangle$ with $\nu_2 = a_2 + p_2 K_2 = 3 + 4p_2$ and $\nu_4 = a_4 + p_4 K_4 = 2 + 2p_4$.

$$\begin{aligned} T_2\nu_2 - T_4\nu_4 &= \nu_2 - 3\nu_4 \\ &= (3 + 4p_2) - 3(2 + 2p_4) \\ &= 2(2p_2 - 3p_4) - 3 = gcd_K^e \pi_e(p_2, p_4) - \alpha_e(3, 2). \end{aligned}$$

a_2 \ a_4	1	2
1	-2	-5
2	-1	-4
3	0	-3
4	1	-2

$\alpha_e(a_2, a_4)$

a_2 \ a_4	1	2
1	0	2
2	0	1
3	-1	1
4	-1	0

$\pi_e^{max}(a_2, a_4)$

a_2 \ a_4	1	2
1	1	2
2	0	2
3	0	1
4	-1	1

$\pi_e^{min}(a_2, a_4)$

Table 6.1: Values $\alpha_e(a_2, a_4)$, $\pi_e^{max}(a_2, a_4)$ and $\pi_e^{min}(a_2, a_4)$ for $a_2 \in \{1, 2, 3, 4\}$ and $a_4 \in \{1, 2\}$.

By Lemma 6.3.1, we get $\pi_e(p_2, p_4) = 2p_2 - 3p_4 = 1$.

Let consider now the couple $(a_2, a_4) = (1, 1)$. Then, since $\pi_e^{max}(1, 1) < \pi_e^{min}(1, 1)$, such a decomposition of the difference $T_2\nu_2 - T_4\nu_4$ with $\nu_2 = 1 + p_2K_2$ and $\nu_4 = 1 + p_4K_4$ is not possible; a simple consequence of Lemma 6.3.1 is that there is no dependence relation between executions $\langle t_2, 1 + p_2K_2\rangle$ and $\langle t_4, 1 + p_4K_4\rangle$.

For the general case, let us define

$$\mathbb{A}(e) = \{(a_i, a_j) \in \{1, \ldots, K_i\} \times \{1, \ldots, K_j\}, \pi_e^{max}(a_i, a_j) \geq \pi_e^{min}(a_i, a_j)\}.$$

Next Lemma is the reverse of Lemma 6.3.1.

Lemma 6.3.2. *Let $e = (t_i, t_j) \in E$ and a couple $(a_i, a_j) \in \mathbb{A}(e)$.*

For any value $\pi \in \{\pi_e^{min}(a_i, a_j), \ldots, \pi_e^{max}(a_i, a_j)\}$, there exist an infinite number of couples $(p_i, p_j) \in \mathbb{N}^2$ such that $\pi = \pi_e(p_i, p_j)$. Moreover, setting $\nu_i = a_i + p_iK_i$ and $\nu_j = a_j + p_jK_j$, e induces a relation from $\langle t_i, \nu_i\rangle$ to $\langle t_j, \nu_j\rangle$.

Proof. By Bezout, there exists $(x, y) \in \mathbb{Z}^2$ such that $xK_iT_i + yK_jT_j = gcd_K^e$ and thus $\pi xK_iT_i + \pi yK_jT_j = \pi gcd_K^e$.

For $z \in \mathbb{N}$, let us define $p_i = \pi x + zK_jT_j$ and $p_j = -\pi y + zK_iT_i$. Let us also consider values ν_i and ν_j such that $\nu_i = a_i + K_ip_i$ and $\nu_j = a_j + K_jp_j$. For z sufficiently large ($z \geq z_0$), $p_i \geq 1$

and $p_j \geq 1$ and thus ν_i and ν_j are both greater than 1. Then,

$$\begin{aligned} T_i p_i K_i - T_j p_j K_j &= K_i T_i(\pi x + z K_j T_j) - K_j T_j(-\pi y + z K_i T_i) \\ &= x\pi K_i T_i + y\pi K_j T_j = \pi gcd_K^e, \end{aligned}$$

thus $\pi = \pi_e(p_i, p_j)$. Now,

$$T_i\nu_i - T_j\nu_j = a_i T_i - a_j T_j + K_i T_i p_i - K_j Z_j p_j = a_i T_i - a_j T_j + \pi gcd_K^e$$

and thus, by definition of α_e, $T_i\nu_i - T_j\nu_j = \alpha_e(a_i, a_j)gcd_T^e + \pi gcd_K^e$. Recall now that $\pi \in \{\pi_e^{min}(a_i, a_j), \ldots, \pi_e^{max}(a_i, a_j)\}$, thus

$$T_i\nu_i - T_j\nu_j \leq \alpha_e(a_i, a_j)gcd_T^e + \pi_e^{max}(a_i, a_j)gcd_K^e,$$

and, since $\pi_e^{max}(a_i, a_j)gcd_K^e \leq -M^e + T_i - \alpha_e(a_i, a_j)gcd_T^e$,

$$T_i\nu_i - T_j\nu_j \leq -M^e + T_i. \tag{6.3.1}$$

Similarly, since $\pi_e^{min}(a_i, a_j)gcd_K^e \geq -M^e + gcd_T^e - \alpha_e(a_i, a_j)gcd_T^e$,

$$\begin{aligned} T_i\nu_i - T_j\nu_j &\geq \pi_e^{min}(a_i, a_j)gcd_K^e + \alpha_e(a_i, a_j)gcd_T^e \\ &\geq -M^e + gcd_T^e > -M^e. \end{aligned} \tag{6.3.2}$$

From equations 6.3.1 and 6.3.2, we get $T_i \geq M^e + T_i\nu_i - T_j\nu_j > 0$ and by Theorem 6.2.2 there is a dependence relation from $\langle t_i, \nu_i\rangle$ to $\langle t_j, \nu_j\rangle$. The lemma is proved. □

From Lemmas 6.3.1 and 6.3.2, we deduce our following main theorem:

Theorem 6.3.3. *Let t_i and t_j be two tasks such that t_i (resp. t_j) is duplicated K_i (resp. K_j) times. Let also $e = (t_i, t_j) \in E$ and $(a_i, a_j) \in \{1, \ldots, K_i\} \times \{1, \ldots, K_j\}$. There exists*

a dependence relation from $\langle t_i, a_i + p_i K_i \rangle$ *to* $\langle t_j, a_j + p_j K_j \rangle$ *for* $(p_i, p_j) \in \mathbb{N}^2$ *iff* $\pi_e(p_i, p_j) \in \{\pi_e^{min}(a_i, a_j), \ldots, \pi_e^{max}(a_i, a_j)\}$.

For the arc $e = (t_2, t_4)$ pictured by Figure 6.2 with values $K_2 = 4$ and $K_4 = 2$, we get from Table 6.1 that $\mathbb{A}(e) = \{(1,2), (2,1), (3,2), (4,1)\}$. By Theorem 6.3.3, there is no dependence due to e between $\langle t_2, a_2 + 4p_2 \rangle$ to $\langle t_4, a_4 + 2p_4 \rangle$ for any value p_2 and p_4 if $(a_2, a_4) \notin \mathbb{A}(e)$. Now, if we consider for example $(a_2, a_4) = (3,2) \in \mathbb{A}(e)$, the set of dependence due to e between $\langle t_2, 3 + 4p_2 \rangle$ to $\langle t_4, 2 + 2p_4 \rangle$ is exactly couples (p_2, p_4) that verifies $2p_2 - 3p_4 = 1$.

6.3.2 Upper bound on the latency

For any arc $e = (t_i, t_j) \in E$ and any couple $(a_i, a_j) \in \mathbb{A}(e)$, Theorem 6.3.3 proved the existence of a dependence from some executions $\langle t_i, \nu_i \rangle$ to $\langle t_j, \nu_j \rangle$ with $\nu_i = a_i + p_i K_i$ and $\nu_j = a_j + p_j K_i$. In order to evaluate the age latency of the whole graph $\mathcal{G}$, next theorem evaluates the maximum latency associated to these executions of t_i and t_j.

Theorem 6.3.4 (Upper bound of the latency between two tasks)**.** *Let* t_i *and* t_j *be two tasks such that* t_i *(resp.* t_j*) is duplicated* K_i *(resp.* K_j*) times. Let also* $e = (t_i, t_j) \in E$ *and* $(a_i, a_j) \in \mathbb{A}(e)$*. Then*

$$\mathcal{L}^{max}_{(a_i,a_j)}(e) = r_j - r_i + T_i - T_j - (\pi_e^{min}(a_i, a_j) \cdot gcd_K^e + \alpha_e(a_i, a_j) \cdot gcd_T^e)$$

is the maximal value of the latency $\mathcal{L}_{\nu_i,\nu_j}(e)$ *for* $(\nu_i, \nu_j) \in \mathcal{R}(e)$ *with* $\nu_i = a_i \mod K_i$ *and* $\nu_j = a_j \mod K_j$.

Proof. By Equation 6.2.1, the latency between executions $\langle t_i, \nu_i \rangle$ and $\langle t_j, \nu_j \rangle$ for $(\nu_i, \nu_j) \in \mathcal{R}(e)$ is $\mathcal{L}_{\nu_i,\nu_j}(e) = r_j - r_i + T_i - T_j - (T_i\nu_i - T_j\nu_j)$. Assuming that $\nu_i = a_i + p_i K_i$ and $\nu_j = a_j + p_j K_j$ with $(p_i, p_j) \in \mathbb{N}^2$ we get by Lemma 6.3.1 that

$$\mathcal{L}_{\nu_i,\nu_j}(e) = r_j - r_i + T_i - T_j - (\pi_e(p_i, p_j) \cdot gcd_k^b + \alpha_b(a_i, a_j) \cdot gcd_T^b) \tag{6.3.3}$$

By Theorem 6.3.3, $\pi_e(p_i, p_j) \in \{\pi_e^{min}(a_i, a_j), \ldots, \pi_e^{max}(a_i, a_j)\}$. We conclude that $\mathcal{L}_{\nu_i,\nu_j}(e)$ is maximum for $\pi_e(p_i, p_j) = \pi_e^{min}(a_i, a_j)$ and the theorem is proved. □

6.3.3 Definition of the partial expanded graph

We suppose that the vector $K \in \mathbb{N}^{\star n}$ is fixed. The associated expanded graph $P_K(\mathcal{G}) = (V, B, \mathcal{L}^{max})$ is a valued directed acyclic graph defined as follows:

1. Each task t_i is duplicated K_i times. For any value $a \in \{1, \ldots, K_i\}$, the ath duplicate of t_i is denoted by t_i^a and is associated to the executions $\langle t_i, a + pK_i \rangle$ for $p \in \mathbb{N}$.

2. For any arc $e = (t_i, t_j) \in E$, we build an arc (t_i^a, t_j^b) for every couple $(a, b) \in \{1, \ldots, K_i\} \times \{1, \ldots, K_j\}$ if $\pi_e^{max}(a, b) \geq \pi_e^{min}(a, b)$.

3. For every arc $\beta = (t_i^a, t_j^b) \in B$, $\mathcal{L}^{max}(\beta) = \mathcal{L}_{(a,b)}^{max}(e)$ following Theorem 6.3.4.

4. Lastly, two additional fictitious tasks s and f are considered with the arcs defined as:

   - For any duplicate t_i^a without predecessor, add the arcs $\beta = (s, t_i^a)$ with $\mathcal{L}^{max}(\beta) = 0$;
   - For any duplicate t_i^a without successor, add the arcs $\beta = (t_i^a, f)$ with $\mathcal{L}^{max}(\beta) = D_i$.

Let K be a fixed positive integer vector. Let us denote by $LP^{max}(P_K(\mathcal{G}))$ the length of the longest path of the associated partial expanded graph $P_K(\mathcal{G})$. By Theorem 6.3.4, $LP^{max}(P_K(\mathcal{G}))$ is an upper bound of the maximum latency of $\mathcal{G}$.

Figure 6.4 presents the expanded graph $P_K(\mathcal{G})$ associated with the vector $K = (2, 4, 1, 2)$ for the instance pictured by Figure 6.2. The longest path is given by $p = s, t_1^2, t_2^3, t_3^1, t_4^2, f$ with a corresponding length equal to 12, i.e. $LP^{max}(P_K(\mathcal{G})) = 12$. We conclude that $\mathcal{L}^{\star}(\mathcal{G}) \leq LP^{max}(P_K(\mathcal{G})) = 12$.

6.3.4 Complexity of the computation of $P_K(\mathcal{G})$nd its longest paths

$P_K(\mathcal{G})$ is a graph without circuit. Thus, the computation of the longest paths can be done in time complexity $\Theta(|V| + |B|)$ by simply sorting the vertices following a topological order used in the next step to explore the vertices.

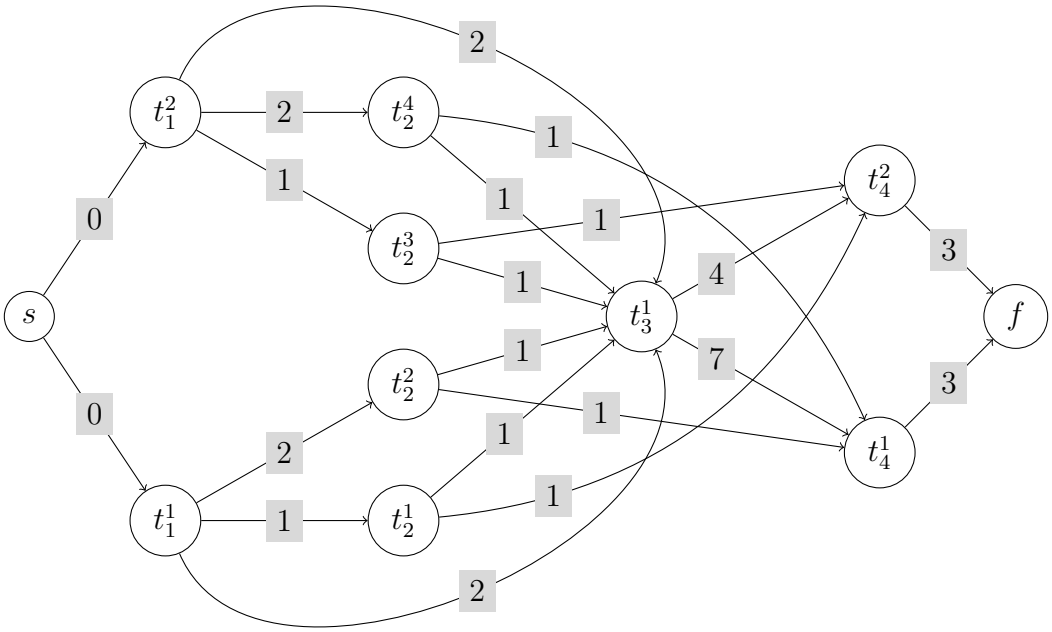


Figure 6.4: Expanded graph $P_K(\mathcal{G}) = (V, B, \mathcal{L}^{max})$ for the instance pictured by Figure 6.2 associated with the vector $K = (2, 4, 1, 2)$. Arcs $\beta \in B$ are valued by $\mathcal{L}^{max}(\beta)$ in gray.

Note that the total number of vertices of $P_K(\mathcal{G})$ is $|V| = \sum_{i=1}^{n} K_i + 2$, while the number of arcs $|B|$ is bounded by $\mathcal{O}(\sum_{e=(t_i,t_j)\in B} K_i \times K_j)$. These two values may be huge for important values of K. The main problem consists then in the determination of the vector K of small values such that the bound $LP^{max}(P_K(\mathcal{G}))$ is as close as possible from the optimum value of the latency.

6.4 Dominant set for the expansion vector K

This section is devoted to the study of dominance properties on K w.r.t the age latency to reduce the set of vectors K. Subsection 6.4.1 formally proves that the value of the longest paths of expanded graphs $P_N(\mathcal{G})$ associated with the hyper-period of $\mathcal{G}$ is the age latency $\mathcal{L}^{\star}(\mathcal{G})$. We prove in Subsection 6.4.2 that we can reduce our study to the set of the partial expansions $P_K(\mathcal{G})$ such that each component K_i divises N_i and we provide a partial order relation between these vectors

that will be exploited in next section for the computation of the latency.

6.4.1 Optimal value of the age latency for $K = N$

Let the least common multiplier define as $T = lcm_{t_i \in \mathcal{T}}(T_i)$ and the vector $N \in \mathbb{N}^{*n}$ define as $N_i = \dfrac{T}{T_i}$ for any task $t_i \in \mathcal{T}$. N is usually called the repetition vector, since if we consider N_i successive executions of each task t_i, the set of constraints between tasks are repeated. For our example pictured by Figure 6.2, we get $T = lcm(2, 1, 6, 3) = 6$ and thus $N = (3, 6, 1, 2)$.

Lemma 6.4.1 is a simple technical lemma.

Lemma 6.4.1. *Let $P_N(\mathcal{G}) = (V, B, \mathcal{L}^{max})$ be the expanded graph with $K = N$ and an arc $e = (T_i, t_j)$ of $\mathcal{G}$. For any arc $\beta = (t_i^{a_i}, t_j^{a_j}) \in B$ associated with e and any couple $(q_i, q_j) \in \mathbb{N}^2$, $\pi_e(q_i, q_j) = q_i - q_j$.*

Proof. By definition of π_e, $\pi_e(q_i, q_j) = \frac{T_i q_i K_i - T_j q_j K_j}{gcd_K^e}$. As $T_i K_i = T_j K_j = T = gcd_K^e$, we get $\pi_e(q_i, q_j) = q_i - q_j$ and the lemma is proved. □

We prove formally in the following that the value of the longest paths of the expanded graph $P_N(\mathcal{G})$ is the age latency of $\mathcal{G}$, i.e. $\mathcal{L}^\star(\mathcal{G})$:

Theorem 6.4.2. *For any acyclic directed graph $\mathcal{G}$, $LP^{max}(P_N(\mathcal{G})) = \mathcal{L}^\star(\mathcal{G})$.*

Proof. By Theorem 6.3.4, $LP^{max}(P_N(\mathcal{G})) \geq \mathcal{L}^\star(\mathcal{G})$. We prove that $LP^{max}(P_N(\mathcal{G})) \leq \mathcal{L}^\star(\mathcal{G})$.

Let consider a path $p_N = t_1^{a_1} \beta_1 t_2^{a_2} \dots \beta_{k-1} t_k^{a_k}$ of $P_N(\mathcal{G})$ and the corresponding path $p = t_1 e_1, \dots, e_{k-1} t_k$ of $\mathcal{G}$. By Lemma 6.4.1, we get for any vector $(q_1, \dots, q_k) \in \mathbb{N}^k$ and $\ell \in \{1, \dots, k-1\}$, $\pi_{e_\ell}(q_\ell, q_{\ell+1}) = q_\ell - q_{\ell+1}$.

Let us consider the sequence of integers $\tilde{q}_1, \dots, \tilde{q}_k$ defined as follows:

- $\tilde{q}_{\ell+1} = \tilde{q}_\ell + \pi_{e_\ell}^{max}(a_\ell, a_{\ell+1})$
- $\tilde{q}_1$ is fixed arbitrarily sufficiently large such that, $\forall \ell \in \{1, \dots, k\}$, $\tilde{q}_\ell \geq 0$.

This sequence verifies that, $\forall \ell \in \{1, \dots, k-1\}$, $\pi_{e_\ell}(\tilde{q}_\ell, \tilde{q}_{\ell+1}) = \pi_{e_\ell}^{max}(a_\ell, a_{\ell+1})$, thus by Theorem 6.3.3, there is a dependence relation from $\langle t_\ell, a_\ell + \tilde{q}_\ell K_\ell \rangle$ to $\langle t_{\ell+1}, a_{\ell+1} + \tilde{q}_{\ell+1} K_{\ell+1} \rangle$. Moreover, by definition of the sequence $\mathcal{L}^{max}$, $\mathcal{L}^{max}(\beta_\ell) = \mathcal{L}_{\tilde{q}_\ell, \tilde{q}_{\ell+1}}(e_\ell)$ and then $\mathcal{L}_{\tilde{q}_1, \dots, \tilde{q}_k}(p) = LP^{max}(p_N)$.

If p_N is the longest path $P_N(\mathcal{G})$, $LP^{max}(P_N(\mathcal{G})) = LP^{max}(p_N) = \mathcal{L}_{\tilde{q}_1,\ldots,\tilde{q}_k}(p) \leq \mathcal{L}^{\star}(\mathcal{G})$, which proves the theorem. □

6.4.2 Order relation between the divisors of the repetition vector N

Next theorem introduces an order relation between vectors $K \in \mathbb{N}^{\star n}$:

Theorem 6.4.3. *For any acyclic directed graph $\mathcal{G}$, let us suppose that K and K' are two different vectors such that $\forall t_i \in \mathcal{T}$, K'_i is a divisor of K_i, then $LP^{max}(P_{K'}(\mathcal{G})) \geq LP^{max}(P_K(\mathcal{G}))$.*

Proof. Let us consider the arc $e = (t_i, t_j)$ of $\mathcal{G}$. By hypothesis, there exists $(x_i, x_j) \in \mathbb{N}^{\star 2}$, such that $K_i = x_i K'_i$ and $K_j = x_j K'_j$. Let $\beta = (t_i^{a_i}, t_j^{a_j})$ be an arc of $P_K(\mathcal{G})$ with $(a_i, a_j) \in \{1, \ldots, K_i\} \times \{1, \ldots, K_j\}$. Then, there exists $(\nu_i, \nu_j) \in \mathbb{N}^{*2}$ such that $\nu_i = a_i + p_i K_i$, $\nu_j = a_j + p_j K_j$ and the latency between execution $\langle t_i, \nu_i \rangle$ and $\langle t_j, \nu_j \rangle$, $\mathcal{L}_{\nu_i,\nu_j}(t_i, t_j)$, reaches the maximum latency of arc β, i.e. $\mathcal{L}_{\nu_i,\nu_j}(t_i, t_j) = \mathcal{L}^{max}(\beta)$.

Let us consider now integer values $a'_i \in \{1, 2, \ldots, K'_i\}$, $a'_j \in \{1, 2, \ldots, K'_j\}$, y_i and y_j such that $a_i = a'_i + y_i K'_i$ and $a_j = a'_j + y_j K'_j$. Thus, $\nu_i = a'_i + (y_i + x_i p_i) K'_i$ and $\nu_j = a'_j + (y_j + x_j p_j) K'_j$. Since there is a dependence relation between $\langle t_i, \nu_i \rangle$ and $\langle t_j, \nu_j \rangle$, $\beta' = (t_i^{a'_i}, t_j^{a'_j})$ belongs to $P_{K'}(G)$ and $\mathcal{L}_{\nu_i,\nu_j}(t_i, t_j) \leq \mathcal{L}^{max}(\beta')$, thus we get $\mathcal{L}^{max}(\beta) \leq \mathcal{L}^{max}(\beta')$.

For any path $p = t_1^{a_1} \beta_1 t_2^{a_2} \beta_2 \ldots \beta_{n-1} t_q^{a_q}$ in $P_K(\mathcal{G})$, there is a corresponding path

$$p' = t_1^{a'_1} \beta'_1 t_2^{a'_2} \beta'_2 \ldots \beta'_{n-1} t_q^{a'_q}$$

in $P_{K'}(G)$ that includes all executions represented by path p. Therefore, $LP^{max}(P_{K'}(\mathcal{G})) \geq LP^{max}(P_K(\mathcal{G}))$. □

For any couple of vectors $(K, K') \in \mathbb{N}^{\star 2}$, we set $K' \preceq K$ if, for any $t_i \in \mathcal{T}$, K'_i divides K_i. By Theorem 6.4.2, the optimum value of the latency is reached for $K = N$. The consequence of this last theorem is that we can limit our study to the set $\mathcal{K}$ of vectors $K \preceq N$. Let us consider the graph $H = (\mathcal{K}, \preceq)$. The evaluation of the age latency is improved following paths from $K = \mathbb{1}^n$

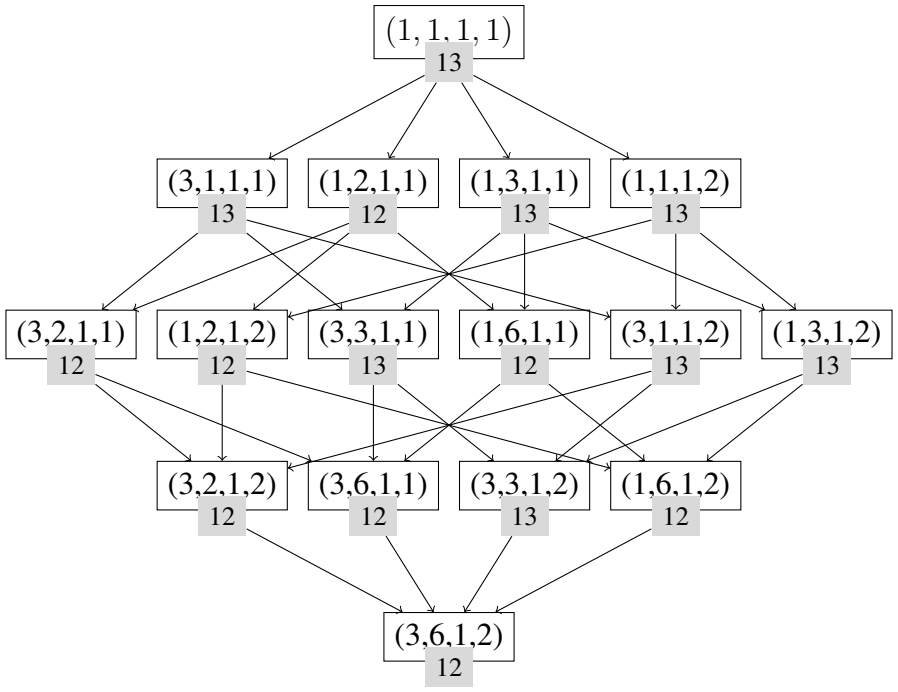


Figure 6.5: Graph $H = (\mathcal{K}, \preceq)$ associated with the example pictured by Figure 6.2. Values $LP^{max}(P_K(\mathcal{G}))$ is given in gray for each vertex $K \in \mathcal{K}$.

to $K = N$. Figure 6.5 shows the graph H associated with the example pictured by Figure 6.2. We observe that the optimum value of the age latency can be reached for vectors K smaller than N.

6.5 Determination of an optimum vector K

The problem considered in this section is to compute iteratively a vector $K \in \mathcal{K}$ to reach a vector $K^\star$ such that $LP^{max}(P_{K^\star}(\mathcal{G})) = \mathcal{L}^\star(\mathcal{G})$. Our algorithm is based on the optimality test expressed by next lemma:

Lemma 6.5.1 (Optimality test)**.** *Let us consider a vector $K \in \mathcal{K}$, a longest path p_K of $P_K(\mathcal{G})$ and its corresponding path p of $\mathcal{G}$. If, for every task $t_i \in p$, K_i is a multiple of $N_i(p) = \dfrac{lcm_{t_j \in p}\{T_j\}}{T_i}$, then $LP^{max}(p_K) = \mathcal{L}^\star(\mathcal{G})$.*

Proof. Let a vector K and the path p of $\mathcal{G}$ following the assumptions of the theorem. Then, by definition of p, $LP^{max}(P_K(\mathcal{G})) = LP^{max}(p_K)$.

By optimality of $\mathcal{L}^\star$, $\mathcal{L}^\star(p) \leq \mathcal{L}^\star(\mathcal{G}) \leq LP^{max}(P_K(\mathcal{G}))$. Now, since for any task t_i of p, $N_i(p)$

is a divisor of K_i, we get by Theorem 6.4.3 that $LP^{max}(P_{N(p)}(p)) \geq LP^{max}(p_K)$. Moreover, by Theorem 6.4.2, $LP^{max}(P_{N(p)}(p)) = \mathcal{L}^{\star}(p)$, thus $\mathcal{L}^{\star}(p) \geq LP^{max}(p_K)$. We conclude that $\mathcal{L}^{\star}(p) = LP^{max}(p_K)$.

Recall that p_K is of maximum age latency in $P_K(\mathcal{G})$, thus $LP^{max}(P_K(\mathcal{G})) = LP^{max}(p_K)$. Now, $\mathcal{L}^{\star}(\mathcal{G}) \geq \mathcal{L}^{\star}(p) = LP^{max}(P_K(\mathcal{G}))$. Since $K \preceq N$, $\mathcal{L}^{\star}(\mathcal{G}) \leq LP^{max}(P_K(\mathcal{G}))$ by Theorem 6.4.3, and thus $LP^{max}(P_K(\mathcal{G})) = \mathcal{L}^{\star}(\mathcal{G}) = LP^{max}(p_K)$, which achieves the proof. □

Algoritm 3 is inspired from K-iter algorithm [13] which computes an expansion vector K for the determination of the optimum throughput of a Synchronous DataFlow Graph. For the initialisation phase, $K = \mathbb{1}^n$. K is simply increased at each step for tasks from the longest path of $P_K(\mathcal{G})$ until the optimality test is met.

Algorithm 3: Compute the maximum latency of $\mathcal{G}$.

Require: A DAG $\mathcal{G} = (\mathcal{T}, E)$, (r_i, D_i, T_i) for every $t_i \in \mathcal{T}$
Ensure: A vector K such that $LP^{max}(P_K(\mathcal{G})) = \mathcal{L}^{\star}(\mathcal{G})$
  Set $K = \mathbb{1}^n$
  repeat
    Compute $P_K(\mathcal{G})$ and a longest path p_K of $P_K(\mathcal{G})$
    Set $p = s, t_1, e_1, \ldots e_{k-1}, t_k, f$ to the corresponding path of $\mathcal{G}$
    Set $T(p) \leftarrow lcm(T_1, \ldots, T_k)$ and $\forall i \in \{1, \ldots, k\}$, $N_i(p) \leftarrow \frac{T(p)}{T_i}$
    OptPathFound$\leftarrow \forall t_i \in p, N_i(p) | K_i$
    if not OptPathFound **then**
      $\forall i \in \{1, \ldots, k\}$, $K_i \leftarrow lcm(K_i, N_i(p))$
    end if
  until OptPathFound

Theorem 6.5.2 shows the convergence of the algorithm.

Theorem 6.5.2. *For any directed acyclic graph $\mathcal{G}$, Algoritm 3 converges to a vector $K^{\star} \in \mathcal{K}$ such that $LP^{max}(P_{K^{\star}}(\mathcal{G})) = \mathcal{L}^{\star}(\mathcal{G})$.*

Proof. For any $q > 0$, we denote by $K(q)$ the vector K at the end of the qth iteration. $q = 0$ corresponds to the initialisation phase. We show that for any integer $q \geq 0$, $K(q) \in \mathcal{K}$ and

$K(q) \preceq K(q+1)$ with $K(q) \neq K(q+1)$.

- At the initialisation step, $K(0) = \mathbb{1}^n \in \mathcal{K}$.
- Now, let suppose that at step q, the optimality test is not true and that $K(q) \in \mathcal{K}$.

  Let consider a task $t_i \in \mathcal{T}$. If t_i does not belong to p, $K_i(q+1) = K_i(q)$. Otherwise, $K_i(q+1) = lcm(K_i(q), N_i(p))$ where $K_i(q)$ and $N_i(p)$ are both divisors of N_i. Thus, $K_i(q+1)$ is also a divisor of N_i, and we get that $K(q+1) \in \mathcal{K}$ with $K(q) \preceq K(q+1)$.
- Lastly, we prove by contradiction that $K(q) \neq K(q+1)$. Indeed, let suppose that $K_i(q) = K_i(q+1)$ for any task $t_i \in \mathcal{T}$, then since $K_i(q+1) = lcm(K_i(q), N_i(p))$, we deduce that $N_i(p)$ is a divisor of $K_i(q)$. We get that the optimality test is true, which is a contradiction.

We conclude that vectors $K(q)$ are strictly increasing while the optimality test is false. By Lemma 6.5.1, the vector $K(q)$ obtained leads to an optimal value of the age latency when the optimality test is true. The optimality test is be true for the repetition vector N, this insures the convergence of the algorithm. □

The number of iterations of Algoritm 3 is not bounded and can be theoretically proportional to the maximum length of a path of the graph $H = (\mathcal{K}, E_{\preceq})$. We will show in Section 6.7 that it can be bounded experimentally for our benchmark to a logarithmic function of $n = |\mathcal{T}|$.

For example, let us consider the first step of Algorithm 3 for the example pictured by Figure 6.2. At the initialisation, $K = \mathbb{1}^4$. The corresponding partial expanded graph $P_K(\mathcal{G})$ is shown by Figure 6.6. Its longest path of $P_K(\mathcal{G})$ is $p_K = s, t_1^1, t_2^1, t_3^1, t_4^1, f$ valued by $LP^{max}(p_K) = 13$. The optimality test fails, and we get $N(p) = (3, 6, 1, 2)$ which is here the repetition vector and thus $K^\star = K(1) = N$. Our algorithm simply obtained the repetition vector. However, our experimentations show that in the general case, the vectors obtained are slightly inferior than the repetition vector.

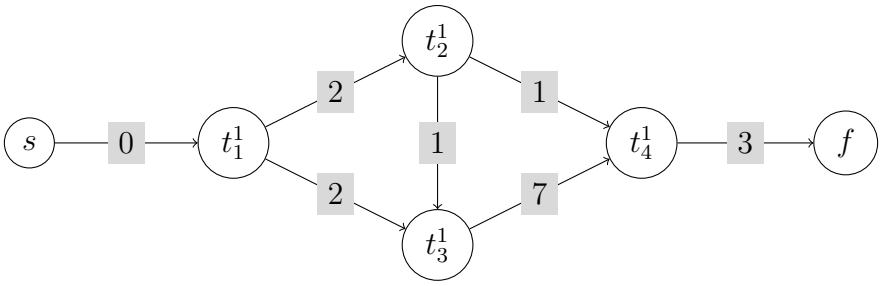
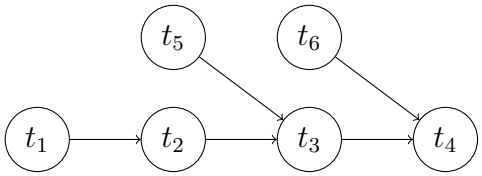


Figure 6.6: The partial expanded graph for the instance pictured by Figure 6.2 and a unit vector $K = (1, 1, 1, 1)$. Arcs are valued by $\mathcal{L}^{max}$ in gray.

6.6 ROSACE Case Study

ROSACE is the acronym for Research Open-Source Avionics and Control Engineering. This case study was developed by Pagetti et al. [76] to illustrate the implementation of a real-time system on a many-core architecture.

Figure 6.7 presents an instance of the problem extracted from [41]. We arbitrarily set $r_i = 0$ and $D_i = T_i$ for any task $t_i \in \mathcal{T}$.



t_i	t_1	t_2	t_3	t_4	t_5	t_6
r_i	0	0	0	0	0	0
D_i	60	60	40	30	30	30
T_i	60	60	40	30	30	30

Figure 6.7: An instance of 6 periodic tasks and the associated DAG $\mathcal{G}$ extracted from ROSACE case study [41].

Figure 6.8 presents the partial expansion of the instance of Figure 6.7 for the unit expansion vector $K = \mathbb{1}^6$. The path of maximum length is $p_K = s, t_1^1, t_2^1, t_3^1, t_4^1, f$ with $LP^{max}(P_K(\mathcal{G})) = LP^{max}(P_K(p)) = 260$ms.

At the first iteration of Algorithm 3, $p = s, t_1, t_2, t_3, t_4, f$ is expanded. We get $T(p) = \text{lcm}(60, 40, 30) = 120$, $N_1(p) = N_2(p) = 2$, $N_3(p) = 3$ and $N_4(p) = 4$. For next iteration,

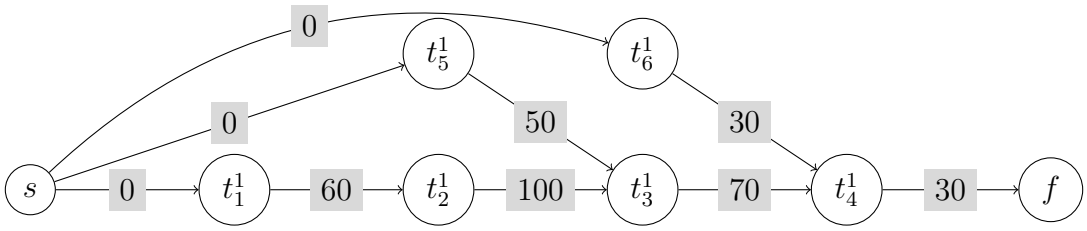


Figure 6.8: The partial expanded graph $P_K(\mathcal{G})$ for the instance pictured by Figure 6.7 and a unit vector $K = \mathbb{1}^6$. Each arc β is valued by $\mathcal{L}^{max}(\beta)$ in gray.

we get $K = (2, 2, 3, 4, 1, 1)$.

The partial expanded graph $P_K(\mathcal{G})$ built at the second iteration is pictured by Figure 6.9. $p_K = s, t_1^1, t_2^2, t_3^2, t_4^4, f$ is a longest path of $P_K(\mathcal{G})$ with $LP^{max}(P_K(\mathcal{G})) = LP^{max}(p) = 240$ Moreover, its associated path $p = s, t_1, t_2, t_3, t_4, f$ verifies $T(p) = lcm(30, 40, 60)$, $N_1(p) = N_2(p) = 2$, $N_3(p) = 3$ and $N_4(p) = 4$. The optimality test is true and we get $K^\star = (2, 2, 3, 4, 1, 1)$. The maximum age latency of $\mathcal{G}$ is thus $\mathcal{L}^\star(\mathcal{G}) = LP^{max}(p_{K^\star}) = 240$ms.

We observe in this example that all the tasks of the critical path (i.e. the paths p of $\mathcal{G}$ such that $\mathcal{L}^\star(p) = \mathcal{L}^\star(\mathcal{G})$) were expanded at least following $N(p)$. Moreover, tasks from other paths are not necessarily duplicated: as example, $K_5^\star = K_6^\star = 1$ with repetition vectors $N_5 = N_6 = 4$. Thus, we can identify that paths s, t_5, t_3, t_4, f and s, t_6, t_4, f are not critical, thus tasks can be delayed without influence on the age latency.

6.7 Experimental results

Our experimentations aim at testing the performance of the Algorithm 3. Following the experimentations of Khatib et al. [58], the bound obtained from the longest paths of $P_{\mathbb{1}^n}(\mathcal{G})$ can be computed quickly, but its performance is in average around 30 percent from the optimal value $\mathcal{L}^\star(\mathcal{G})$. Moreover, their method does not identify precisely the real critical paths w.r.t the age latency of the initial graph.

Our Benchmarks were randomly generated: they are detailed is Subsection 6.7.1. The analysis of the computation time of our algorithm is presented in Subsection 6.7.2. Subsection 6.7.3 deals

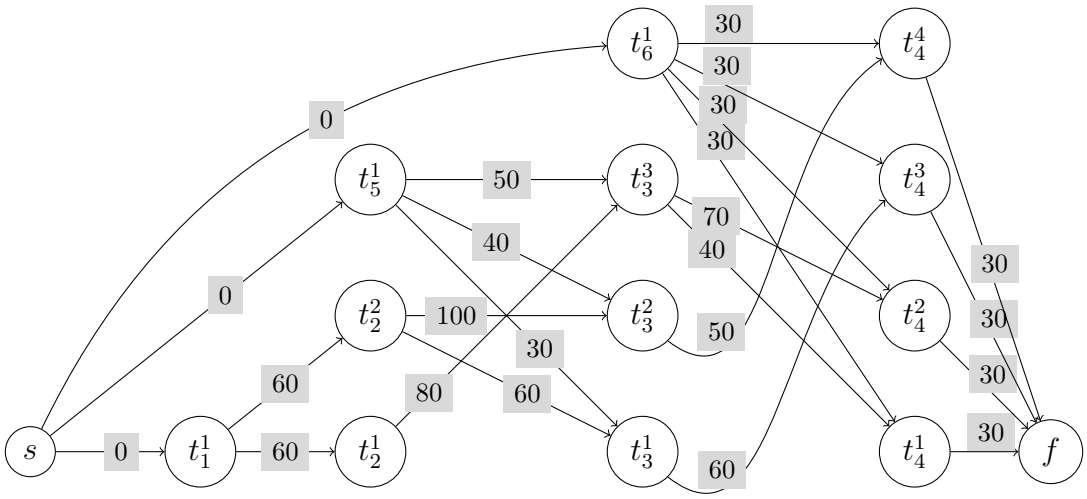


Figure 6.9: The partial expanded graph $P_K(\mathcal{G})$ for the instance pictured by Figure 6.7 and the vector $K = (2, 2, 3, 4, 1, 1)$. Each β arc is valued by $\mathcal{L}^{max}(\beta)$.

with the analysis of the critical vectors $K^\star$ obtainde by our algorithm. All our experiments were performed on an Intel(R) Core(TM) i5-8400 CPU (6 cores at 2.80GHz) and 15 GB of RAM.

6.7.1 Benchmarks

Random instances of n tasks were generated as follows. Periods of tasks are selected uniformly in $\mathcal{H} = \{1, 2, 5, 10, 20, 50, 100\}$. $\mathcal{H}$ is a subset of the values presented by Kramer et al. [61] for the 2015 WATERS challenge and several authors dealing with the age latency for automotive applications [10, 47].

Release times r_i are uniformly selected in $\{0, 1, 2, 3, 4, 5\}$, while we fix the relative deadline D_i equal to the period of the task i.e. $D_i = T_i$ for any task $t_i \in \mathcal{T}$. All functions dealing with graphs were implemented using the Python package NetworkX. Graphs are randomly generated using the function `dense_gnm_random_graph`. Nodes are arbitrary numbered from 1 to n. A directed acyclic graph is then built by replacing each edge $e = \{i, j\}$ with $i < j$ by an arc $e = (i, j)$.

For any number n of tasks, we set the number of arcs to $m_\ell = \lfloor \frac{(n(n-1)}{4} \rfloor$ arcs for *low density* graphs and $m_h = \lceil \frac{(n(n-1)}{3} \rceil$ for *high density*. We start with $n = 5$ tasks with a step of 5. For each data point, 150 random instances were generated and an average value of the functions considered

are shown.

6.7.2 Analysis of the computation time of the Algorithm 3

For n sufficiently large, the hyper-period of an instance is exactly $T = lcm\{\alpha \in \mathcal{H}\} = 100$. The consequence is that the number of duplicates (resp. the number of arcs) of the expanded graph $P_N(\mathcal{G})$ is bounded by $T \times n$ (resp. $T^2 \times n^2$).

We measured the running time and the number of iterations of Algorithm 3. We stopped at $n = 90$ tasks, since the running time exceeded 15 minutes in average for instances with higher values of n. Figure 6.10 reports the average running times (on the top) and the average number of iterations (on the bottom) following the number of tasks.

We observed that the running time of Algorithm 3 is a quadratic function of the number of tasks, and thus is linear following the number of arcs of the graph $\mathcal{G}$. With no surprise, these running times are more important for high density graphs. This observation seems to be in opposition with the experimental results of Becker et al.[10]: indeed, they remarked that the average running time for the computation of the age latency of a chain is linear w.r.t the number of tasks. However the number of arcs here equals $n - 1$, the running time is then also linear w.r.t the number of arcs, which is coherent with our result.

We also noticed that the whole number of iterations of the Algorithm 3 grows in average following a logarithmic way. Our first experimental conclusion is thus that the convergence of the algorithm to the exact value seems to be a logarithmic function of the number of tasks. The important running time is thus due to the time needed to build the successive partial expansion and not to the increase of the number of iterations of the algorithm.

6.7.3 Analysis of the partial expanded graph obtained

Figure 6.11 presents the evolution of the ratio $r(n) = \frac{\sum_{i=1}^{n} K_i^\star}{\sum_{i=1}^{n} N_i}$ following the number of tasks and the density of the graph. We observed that is a roughly linear function that remains bounded by 0.8 for high density graphs and 0.65 for low density. The consequence is that in many case we clearly do not need to expand completely the graph to get the exact value of the age latency and

that good algorithms should be sought to identify critical paths of a graph.

6.8 Conclusion

We exhibited in this chapter a new definition of the dependence between the successive executions of two tasks that communicate following a LET paradigm. This definition was exploited to build a partial expanded graph $P_K(\mathcal{G})$ associated to any vector $K \in \mathbb{N}^{\star n}$ for the computation of a lower bound of the age latency. A greedy algorithm to compute an accurate value $K^\star$ leading to the optimal value of the age latency was developed and tested on random realistic instances. This optimal partial expansion allows to identify the critical paths of the graph $\mathcal{G}$.

Many extensions of our study may be considered. This methodology can surely be applied to evaluate accurate lower bounds of the age latency. Coupling the upper and the lower bounds will allow then to measure precisely the error between the longest paths of $P_K(\mathcal{G})$ and $\mathcal{L}^\star(\mathcal{G})$. Our general framework should also be extended to tackle other possible latencies [38]. Lastly, an implicit communication between two tasks of same period (which corresponds to two tasks in the same runnable for an AUTOSAR compatible system) can also easily be considered in our model.

This is the fitting function and the standard errors of the fitting below:

measurements	density	Fitting function	standard deviation errors
$iter$	high	$1.34log(0.62(n+5.89)) - 0.64$	$[2.90e-01, 1.08e+06, 5.66e+00, 2.32e+06]$
$iter$	low	$1.96log(1.59(n+13.42)) - 4.81$	$[3.01e-01, 1.04e+06, 6.36e+00, 1.29e+06]$
$\sum_i N_i$	high	$27.04n - 11.59$	$[0.33, 17.57]$
$\sum_i N_i$	low	$27.75n - 37.99$	$[0.37, 19.83]$
rt	high	$2.02e-03n^2 - 0.03n + 0.29$	$[1.08e-04, 1.05e-02, 2.17e-01]$
rt	low	$1.53e-03n^2 - 0.05n + 0.51$	$[1.29e-04, 1.26e-02, 2.60e-01]$
$\frac{\sum_i K_i}{\sum_i N_i}$	high	$8.67e-04n + 0.69$	$[1.71e-04, 9.26e-03]$
$\frac{\sum_i K_i}{\sum_i N_i}$	low	$9.10e-04n + 5.27e-01$	$[2.37e-4, 1.28e-2]$

Table 6.2: Fitting function of each measurements.

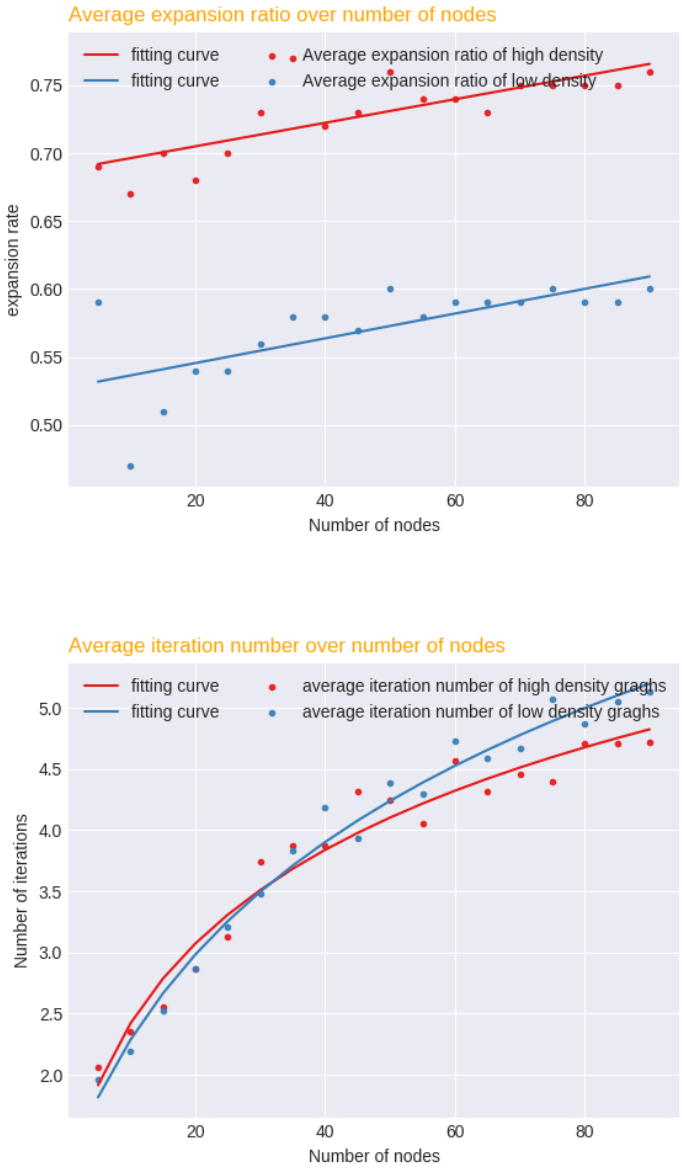


Figure 6.10: Average running times (on the top) and average number of iterations (on the bottom) of Algorithm 3 for randomly generated high density and low density instances.

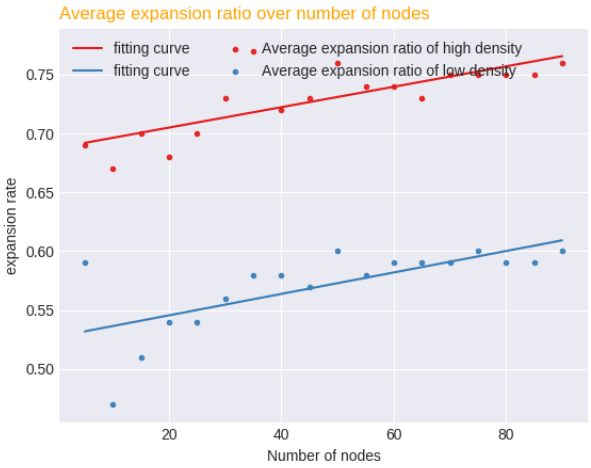


Figure 6.11: Average ratio $r(n) = \frac{\sum_{i=1}^{n} K_i^\star}{\sum_{i=1}^{n} N_i}$ for the partial expanded graph computed by Algorithm 3 for randomly generated high density and low density instances.

7. Global conclusions and perspectives

7.1 Fixed-parameter tractable algorithms for fundamental scheduling optimization problem

Discover the optimal schedule has always been a hard question. Fixed-parameter tractable (FPT) algorithm is one of the methods to get an optimal solution of a hard problem in a reasonable time. Many attempts were made to solve scheduling problems with FPT algorithms with different parameters recently like width of precedence graph, numbers of chains etc. However, there are few results of scheduling problems with communication delays, despite the trend of longer and longer communication time in super computer and data centers nowadays.

In chapter 4, Algorithm 1 of UET-UCT scheduling problem i.e. $\overline{P}|r_i, prec, p_i = 1, c_{ij} = 1|C_{max}$ was introduced and was proved to be a fixed-parameter tractable. The time complexity of Algorithm 1 is $\mathcal{O}(n^3 \cdot pw(\overline{C}) \cdot 2^{2pw(\overline{C})})$, where $pw(\overline{C})$ is the pathwidth of the interval graph associated to the time windows $[r_i, d_i]$, $i \in \mathcal{T}$ with an upper bound of makespan $\overline{C}$.

In chapter 5, the scheduling problem $P|r_i, prec, p_i = 1, c_{ij} = 1|L_{\max}$ was introduced and Algorithm 2 was developed and proved to be fixed-parameter tractable in the pathwidth $pw(\bar{L})$ associated to an upper bound $\bar{L}$ of the maximum lateness. The time complexity of Algorithm 2 is $\mathcal{O}(n^2 + n \times pw(\overline{L}) \times 2^{3pw(\overline{L})})$, where $pw(\overline{L})$ is the pathwidth of the interval graph associated to the time windows $(r_i, d_i + \overline{L})$, $i \in \mathcal{T}$.

These algorithms are inspired by the work of Alix Kordon-Munier [72]. These are the first fixed-parameter tractable algorithms for scheduling with communication delays. These results opens the discussion of the parameter: pathwidth of interval graph of time windows. For example, can this parameter be extended to other scheduling problems such as large communication delay problems, problems on multiple types of machines system? Can the parameter be simplified to numbers of chains, width or treewidth of precedence graph etc? Can it be extended to other

optimization criteria like total weighted tardiness and total weighted completion time? Besides, researches on reducing the size of time windows are important for this algorithm to limit pathwidth. In parameterized complexity theory, Courcelle's theorem identifies a set of problems on undirected graphs that can be solved with FPT algorithms with treewidth as the parameter. However, it remains open if Courcelle's theorem be extended to scheduling problems with pathwidth as the parameter.

In application point of view, when the parameter is bounded, comparison of the performance between our algorithms and branch and bound method, integer linear programming method is interesting. Industrial cases in which pathwidth can be tractable still need to be identified. Experiments on industrial cases with bounded pathwidth are to be conducted.

7.2 Calculation of age latency in real time systems

The goal of this part of thesis is to develop mathematical and algorithmic tools to compute efficiently the age latency of a real-time system of multi-periodic communicating tasks under LET assumption. It is answered with an original method that computes efficiently the age latency of a general task communication graph G of a real-time system without circuit. A wide number of methods to this problem limit their study to chains, but they can not be applied efficiently on general graph by simply enumerate them because the number of paths between two nodes is potentially exponential.

A new definition of the dependence between the successive executions of two tasks communicating in LET paradigm was provided This definition was exploited to build a partial expanded graph $P_K(\mathcal{G})$ associated to any vector $K \in \mathbb{N}^{\star n}$ for the computation of a lower bound of the age latency. A greedy algorithm to compute the optimal value of the age latency was developed and tested on random realistic instances. This algorithm also allows to identify the critical paths of the graph $\mathcal{G}$.

This research is limited to real-time systems with LET paradigm, extension to other real-time system communication paradigm is waiting to be researched. Cooperation with automotive design

industry is needed to test the method in a more complex environment.

REFERENCES

[1] Autosar.

[2] Massinissa Ait Aba, Alix Munier-Kordon, and Guillaume Pallez. Scheduling on two unbounded resources with communication costs. In *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26-30, 2019, Proceedings*, pages 117–128, 2019.

[3] Abdessamad Ait El Cadi, Rabie Ben Atitallah, Said Hanafi, Nenad Mladenovic, and Abdelhakim Artiba. New MIP model for multiprocessor scheduling problem with communication delays. *Optimization Letters*, page 15, 2014.

[4] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems*, 11 2021.

[5] Theodore Andronikos, Nectarios Koziris, George Papakonstantinou, and Panayiotis Tsanakas. Optimal scheduling for uet/uet-uct generalized n-dimensional grid task graphs. *J. Parallel Distrib. Comput.*, 57(2):140–165, 1999.

[6] Stefan Arnborg. Efficient algorithms for combinatorial problems with bounded decomposability - A survey. *BIT*, 25(1):1–23, 1985.

[7] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[8] Kenneth R Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[9] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169, Aug 2016.

[10] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104 – 113, 2017.

[11] Alessandro Biondi and Marco Di Natale. Achieving predictable multicore execution of automotive applications using the LET paradigm. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*, pages 240–250, 2018.

[12] Bruno Bodin, Alix Munier-Kordon, and Benoît Dupont de Dinechin. K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph. In *2012 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII, Samos, Greece, July 16-19, 2012*, pages 152–159, 2012.

[13] Bruno Bodin, Alix Munier-Kordon, and Benoît Dupont de Dinechin. Optimal and fast throughput evaluation of CSDF. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 160:1–160:6, 2016.

[14] Hans Bodlaender, Jens Gustedt, and Jan Telle. Linear-time register allocation for a fixed number of registers. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 09 2001.

[15] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998.

[16] Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

[17] Hans L. Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays, 2020.

[18] Jonathan F. Buss and J. Goldsmith. Nondeterminism within p. *SIAM J. Comput.*, 22:560–572, 1993.

[19] Aurélien Carlier, Claire C. Hanen, and Alix Munier-Kordon. The equivalence of two classical list scheduling algorithms for dependent typed tasks with release dates, due dates and precedence delays. *Journal of Scheduling*, pages 1–9, 2017.

[20] Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Peter Rossmanith, and Ping-Chen Su. Fixed-parameter algorithms for vertex cover p3. *Discrete Optimization*, 19:12–22, 2016.

[21] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. 1998.

[22] Philippe Chrétienne. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research*, 43(2):225–230, 1989.

[23] Philippe Chrétienne and Christophe Picouleau. *Scheduling with communication delays: A survey*, pages 65–90. 1995.

[24] J. Y. Colin and P. Chrétienne. C.p.m. scheduling with small communication delays and task duplication. *Oper. Res.*, 39(4):680–684, aug 1991.

[25] Richard Walter Conway, William L Maxwell, and Louis W Miller. *Theory of scheduling*. Courier Corporation, 2003.

[26] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[27] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[28] Tatjana Davidović, Leo Liberti, Nelson Maculan, and Nenad Mladenovic. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. *In Proc. 3rd Multidisciplinary Int. Conf*, 01 2007.

[29] Sami Davies, Janardhan Kulkarni, Thomas Rothvoss, Jakub Tarnawski, and Yihao Zhang. Scheduling with communication delays via LP hierarchies and clustering. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 822–833. IEEE, 2020.

[30] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4), oct 2011.

[31] Robert de Groote. *On the analysis of synchronous dataflow graphs: a system-theoretic perspective*. PhD thesis, University of Twente, 2016.

[32] R.G. Downey and M.R. Fellows. Fixed-parameter intractability. In *[1992] Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 36–49, 1992.

[33] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In Klaus Ambos-Spies, Steven Homer, and Uwe Schöning, editors, *Complexity Theory: Current Research, Dagstuhl Workshop, February 2-8, 1992*, pages 191–225. Cambridge University Press, 1992.

[34] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[35] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. 1999.

[36] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. 2013.

[37] Maciej Drozdowski. *Scheduling for Parallel Processing*. Springer, 2009.

[38] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *IEEE Real-Time Systems Symposium, November 30-December 3*. IEEE Communications Society, 2009.

[39] Afonso G. Ferreira and Siang W. Song. Achieving optimality for gate matrix layout and pla folding: a graph theoretic approach. In *of Lecture*, pages 173–195, 1992.

[40] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[41] Julien Forget, Frédéric Boniol, and Claire Pagetti. Verifying end-to-end real-time constraints on multi-periodic models. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017, Limassol, Cyprus, September 12-15, 2017*, pages 1–8, 2017.

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

[43] Bernard Giffler and Gerald Luther Thompson. Algorithms for solving production-scheduling problems. *Operations research*, 8(4):487–503, 1960.

[44] Rodolphe Giroudeau and Jean-Claude König. Scheduling with Communication Delay. In *Multiprocessor Scheduling: Theory and Applications*, pages 1–26. ARS Publishing, December 2007.

[45] Rodoplhe Giroudeau and Jean-Claude König. Scheduling with communication delays. In Eugene Levner, editor, *Multiprocessor Scheduling*, chapter 4. IntechOpen, Rijeka, 2007.

[46] R. L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.

[47] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. Communication centric design in complex automotive embedded systems. In *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*, pages 10:1–10:20, 2017.

[48] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, Falk Wurst, and Dirk Ziegenbein. Waters industrial challenge 2017. Bosch GmbH,Inria Grenoble – Rhône-Alpes, 2017.

[49] C Hanen and A Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, 108(3):239 – 257, 2001.

[50] Claire Hanen, Alix Munier-Kordon, and Theo Pedersen. Two Deadline Reduction Algorithms for Scheduling Dependent Tasks on Parallel Processors (extended version). Research report, LIP6, Sorbonne Université, April 2021.

[51] Claire Hanen and Yakov Zinder. The worst-case analysis of the garey-johnson algorithm. *J. Sched.*, 12(4):389–400, 2009.

[52] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

[53] Thomas A. Henzinger, Christoph M. Kirsch, Marco A.A Sanvido, and Wolfgang Pree. From control models to real-time code using Giotto. *IEEE Control Systems Magazine*, 23(1):50–64, Feb 2003.

[54] Petr Hliněny. Crossing-number critical graphs have bounded path-width. *Journal of Combinatorial Theory, Series B*, 88(2):347–367, 2003.

[55] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3):129 – 137, 1994.

[56] Klaus Jansen, Oliver Sinnen, and Huijun Wang. An EPTAS for scheduling fork-join graphs with communication delay. *Theor. Comput. Sci.*, 861:66–79, 2021.

[57] Sana Jeddi and Wahid Nasri. A poly-algorithm for efficient parallel matrix multiplication on metacomputing platforms. In *2005 IEEE International Conference on Cluster Computing*, pages 1–9, 2005.

[58] Jad Khatib, Alix Munier-Kordon, Enagnon Cédric Klikpo, and Kods Trabelsi-Colibet. Computing latency of a real-time system modeled by synchronous dataflow graph. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, pages 87–96, 2016.

[59] Christoph M. Kirsch and Ana Sokolova. The logical execution time paradigm. In Samarjit Chakraborty and Jörg Eberspächer, editors, *Advances in Real-Time Systems*, pages 103–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[60] Dzmitry Kliazovich, Johnatan E. Pecero, Andrei Tchernykh, Pascal Bouvry, Samee Ullah Khan, and Albert Y. Zomaya. CA-DAG: modeling communication-aware applications for scheduling in cloud computing. *J. Grid Comput.*, 14(1):23–39, 2016.

[61] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. Robert Bosch GmbH, 2015.

[62] Edward A. Lee and David G. Messerschmitt. Synchronous data flow. *Proceeding of the IEEE*, vol. 75(no. 9):pp. 1235–1245, 1987.

[63] Jan Karel Lenstra, Marinus Veldhorst, and Bart Veltman. The complexity of scheduling trees with communication delays. *J. Algorithms*, 20(1):157–173, January 1996.

[64] Allen Leung, Krishna V. Palem, and Amir Pnueli. Scheduling time-constrained instructions on pipelined processors. *ACM Trans. Program. Lang. Syst.*, 23(1):73–103, 2001.

[65] Qing Li and Caroline Yao. *Real-time concepts for embedded systems*. Taylor and Francis, Hoboken, NJ, 2014.

[66] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[67] Quanquan C. Liu, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua R. Wang. Scheduling with communication delay in near-linear time, 2021.

[68] A.D. Lopez and H.-F.S. Law. A dense gate matrix layout method for mos vlsi. *IEEE Transactions on Electron Devices*, 27(8):1671–1675, 1980.

[69] Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2244–2254, Nov 2018.

[70] Matthias Mnich and René Van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100, 2018.

[71] Alix Munier and Jean-Claude König. A heuristic for a scheduling problem with communication delays. *Operations Research*, 45(1):145–147, 1997.

[72] Alix Munier-Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discrete Applied Mathematics*, 290:1–6, 2021.

[73] Rolf H. Möhring and Markus W. Schäffter. Scheduling series–parallel orders subject to 0/1-communication delays. *Parallel Computing*, 25(1):23 – 40, 1999.

[74] T. Ohtsuki, H. Mori, E. Kuh, T. Kashiwabara, and T. Fujisawa. One-dimensional logic gate assignment and interval graphs. *IEEE Transactions on Circuits and Systems*, 26(9):675–684, 1979.

[75] Michael Orr and Oliver Sinnen. Optimal task scheduling benefits from a duplicate-free state-space. *Journal of Parallel and Distributed Computing*, 146, 2020.

[76] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The ROSACE case study: from simulink specification to multi/many-core execution. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 309–318, April 2014.

[77] C. Picouleau. New complexity results on scheduling with small communication delays. *Discrete Applied Mathematics*, 60(1):331 – 342, 1995.

[78] V.J. Rayward-Smith. Uet scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18(1):55 – 71, 1987.

[79] Neil Robertson and P.D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[80] Kana Shimada, Ittetsu Taniguchi, and Hiroyuki Tomiyama. Communication-aware scheduling for malleable tasks. In *2019 International Conference on Platform Technology and Service (PlatCon)*, pages 1–6, 2019.

[81] Oliver Sinnen. Reducing the solution space of optimal task scheduling. *Computers & Operations Research*, 43:201–214, 2014.

[82] Jorge P. Sousa and Laurence A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming volume*, (54):353—-367, 1992.

[83] Matthew Suderman. Pathwidth and layered drawings of trees. *International Journal of Computational Geometry and Applications*, 14, 01 2003.

[84] Ning Tang and Alix Munier-Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks with unit communication delays. In Leonel Sousa, Nuno Roma, and Pedro Tomás, editors, *Euro-Par 2021: Parallel Processing - 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1-3, 2021, Proceedings*, volume 12820 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2021.

[85] Sun Tzu. *The Art of War*. 5th century BC.

[86] René van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width, 2016.

[87] B Veltman, B.J Lageweg, and J.K Lenstra. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16(2):173 – 182, 1990.

[88] Bart Veltman. *Multiprocessor scheduling with communication delays*. PhD thesis, Eindhoven University of Technology, 1993.

[89] S. Venugopalan and O. Sinnen. Ilp formulations for optimal task scheduling with communication delays on parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):142–151, 2015.

[90] Rémy Wyss, Frédéric Boniol, Claire Pagetti, and Julien Forget. End-to-end latency computation in a multi-periodic design. In *28th Symposium On Applied Computing (SAC'13)*, pages 1682–1687, Coimbra, Portugal, April 2013.

[91] Frederic Vivien Yves Robert. *Introduction to Scheduling*. CRC Press, Boca Raton, 2009.

[92] Yakov Zinder, Bo Su, Gaurav Singh, and Ron Sorli. Scheduling uet-uct tasks: Branch-and-bound search in the priority space. *Optimization and Engineering*, 11:627–646, 2010.

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page