



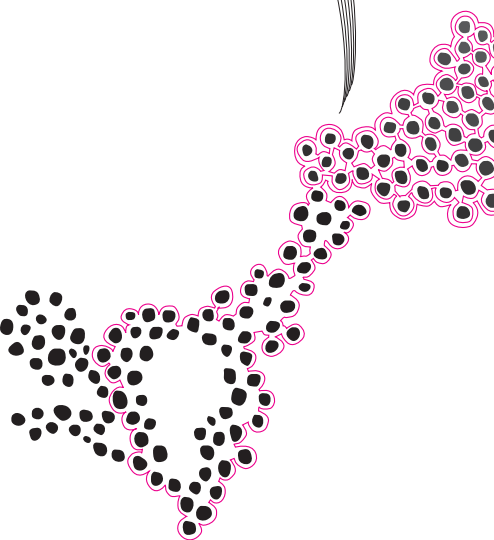
BSc Thesis Applied Mathematics

**Planning interdependent tasks in  
house construction  
- an extension to the critical path  
method**

Esmee Nijhof

Supervisor: Gerhard Post

July 22, 2022



Department of Applied Mathematics  
Faculty of Electrical Engineering,  
Mathematics and Computer Science

## **Preface**

This report was written as a bachelor assignment in the study Applied Mathematics in the department of discrete mathematics and mathematical programming. I would like to thank my supervisor, Gerhard Post, who guided and supported me through the process of writing a bachelor thesis. I have learnt a lot about the mathematics behind planning, scheduling and about research itself, and this can be very valuable over the remaining course of my studies.

## **Abstract**

House construction can often be seen as projects with tasks need to be done. Naturally, tasks have a certain order, so-called task dependencies. For example, a foundation has to be poured before walls can be built. Project based scheduling started with the Critical Path Method (CPM) during the late 1950's and ever since, this method was used as a basis for many ways of project scheduling [9]. CPM respects the interdependencies between the tasks, but assumes unlimited resources, i.e. workers that can execute tasks of a project. Also, CPM is unable to consider tasks that become available during the project and it does not take into account that some tasks may have to be finished before a deadline within the project. This paper aims to find a method which considers the interdependencies and these factors simultaneously. This is done by splitting the tasks between the number of resources and then sequencing the tasks per resource. The found method uses the well-known multiway number partition problem and the CPM in order to divide the tasks among the resources. In addition, by allowing pre-emption, the method is able to consider unequal release dates of the materials. The method deals with NP-hard problems, so they can probably not be solved in polynomial time. Additionally, the limitations of the method are explained and demonstrated. More research is required to find methods that are able to schedule projects with the use of the CPM, while considering resource constraints in polynomial time.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Linear Program</b>	<b>6</b>
<b>3</b>	<b>Critical Path Method without resource constraints</b>	<b>8</b>
<b>4</b>	<b>Critical Path Method with resource constraints</b>	<b>10</b>
4.1	CPM with m resources and no release dates . . . . .	10
4.1.1	Method . . . . .	10
4.1.2	Numerical example . . . . .	12
4.1.3	Illustration of flaws . . . . .	14
4.2	CPM with m resources and unequal release dates . . . . .	15
4.2.1	Method . . . . .	15
4.2.2	Numerical example . . . . .	16
<b>5</b>	<b>Conclusion</b>	<b>18</b>
<b>6</b>	<b>Discussion</b>	<b>19</b>

# 1 Introduction

Often, in project based scheduling, there are many different tasks that need to be done in a certain sequence. It can be a challenge to plan those tactically, as there are many factors to consider in this process, such as the availability of resources with regards to workers and materials. In addition, some tasks naturally have a well defined order, for instance, in house construction, the foundation of a house must be poured before the building of the walls can start, or painting the walls and frames can only be done after door frames are placed. Though obtaining an optimal project schedule may be hard, it has great benefits for both the management of the project and the workers. It makes it easier to identify the progress of the overall project and in case of unexpected challenges, it helps to see what tasks should be prioritized [10].

The desire for optimal planning of large projects originated from the second world war. For instance, in a conflict the defender needs to choose against which offensive tools from the attacker they should make countermeasures first, while considering the costs and damages that come with this. In addition, the defender needs to decide how to divide the resources strategically, as they are limited. This can be seen as a network optimization problem if the outcome of events, such as the duration of counterattacks, is deterministic. Also, if the strategic plan of the attacker is predictable, then this can also be seen as deterministic events [6]. From then on, scheduling methods and tools that help solve these kind of problems have been developed. PERT, for example, is a scheduling tool for probabilistic problems [11]. One of the first methods was the Critical Path Method (CPM). This method aims to minimize the total project duration in which a project can be finished, while respecting the order in which tasks have to be done [10]. The CPM is suitable for house construction projects, as it is a deterministic model and construction tasks have a deterministic duration.

This method was published shortly after the war and opened many doors in the field of scheduling and planning. Even though, the method as designed in 1959 is as a solid foundation for scheduling projects, it has some limitations. For instance, it does not consider any resource constraints, i.e. the number of workers that are available. Furthermore, CPM is also unable to consider release dates of tasks. It may occur that materials are not available, which makes a task available during the project, instead of at the start of the project. Ideally, resources and materials have unlimited availability, but in practice, this is hardly ever the case. Note that in this paper, a company's workers are considered as resources and this does not entail any materials needed for a task.

This paper investigates whether the CPM can be extended or adapted in such a way that it also considers resource constraints. Firstly, the problem is defined as a linear program, which is solvable under certain assumptions. We then discuss the general CPM method and its limitations with regard to resource constraints. Additionally, an alternative method in case of  $m$  workers is given. The problems involved in this method are NP-hard, which complicates finding a solution for the problem. The discussion elaborates on this in more detail.

## 2 Linear Program

Before the CPM and its limitations are explained, a linear program defines the problem. The input is a set of tasks  $J = \{1, 2, \dots, n\}$ , where  $n$  is the number of tasks. In addition, each task,  $i \in J$ , has an associated duration  $d_i \in \mathbb{N}$ , which is known. The linear program indicates time units  $t \in T$ , with  $T = \{1, 2, \dots, t_p\}$ , where  $t_p = \sum_{i \in J} d_i$ . The linear program aims to minimize the total project duration  $M$ , while considering the predefined order in which the tasks need to be performed, which are also called precedence constraints [15, 17]. These are denoted as follows:  $i \rightarrow j$ . In case of a precedence constraint, task  $i$  has to be completed before task  $j$  can start. Additionally, each task  $i \in J$  has a release date  $r_i$ . This indicates when a task is available. If all tasks are available from the start of the project, all  $r_i$  can be set to 0. Furthermore, pre-emption is not allowed, which means that a task has to be done without any interruptions. Next to that, there are resource limitations, as there are a limited number of workers available at a time. The variable  $m$  defines the number of resources available. It assumes that workers are always available during the project.

The decision variable is defined as follows:

$$x_{it} = \begin{cases} 1 & \text{if task } i \text{ finishes at time } t \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Also, to increase the readability of the linear program, we introduce another variable: the number of tasks that are active in the time interval  $t - 1$  to  $t$ .

$$n_t = \sum_i \sum_{t=t+d_i-1} x_{it} \quad \forall i \in J, \forall t \in T \quad (2)$$

The minimization problem is summarized into the following LP:

$$\begin{aligned} & \text{minimize} && M \\ & \text{subject to} && \sum_t x_{it} = 1 && \forall i \in J && (i) \\ & && n_t \leq m && \forall t \in T && (ii) \\ & && \sum_t t x_{it} \leq \sum_t t x_{jt} - d_j && \forall i \rightarrow j \text{ with } i, j \in J && (iii) \\ & && x_{it} = 0 && \forall i, \forall t < d_i + r_i && (iv) \\ & && \sum_t t x_{it} \leq M && \forall i \in J && (v) \\ & && x_{it} \in \{0, 1\} && \forall i \in J, \forall t \in T && (vi) \end{aligned}$$

Here the objective is to minimize the total project duration by ensuring that each task is finished at the earliest possible time. Note that  $\sum_{t \in T} t x_{it}$  is the time at which task  $i$  is finished [16]. Constraint (i) ensures that each task is scheduled once, by allowing exactly one finish time for each job. Moreover, Constraint (ii) considers the resource constraint. This makes sure that the number of active tasks in time interval  $[t - 1, t]$  does not exceed the number of workers available at each time interval. Furthermore, the precedence constraints are considered in Constraint (iii). For each relation  $i \rightarrow j$ , task  $j$  can only start after task  $i$  has finished [7]. Note that the start time of a task can be calculated by subtracting the duration from the finish time. Also, the earliest time a task can be finished is after the release time and

duration have passed, so if  $t \geq r_i + d_i$ , which is guaranteed by Constraint (iv). In addition, Constraint (v) states that every task should be finished before the total project duration has passed. Constraint (vi) defines the decision variable for each job, like Equation (1).

This linear program can be easily adapted, so that the availability of the resources can be non-constant by making variable  $m$  dependent on time. Also, by changing the LP, it can also consider task deadlines, which means that a task must be finished before some time  $t$ . If a task is not finished before the deadline, the task is late and this should be minimized. In principle, this does not change the LP a lot, except if more tasks than available resources can be chosen to be scheduled at the same time. Due to the deadlines, the task which is the least tardy should be chosen in this case.

### 3 Critical Path Method without resource constraints

This section explains the classic CPM and ways in which this method fails to consider resource constraints are demonstrated.

The CPM aims to find the minimal total project duration while considering the precedence constraints of the tasks when a list of tasks and their duration is given. This is done with a clear project structure by representing all tasks and their interrelations [10]. For each task, it should be considered which tasks immediately precede this task, which tasks immediately follow this task, and which tasks can be done parallel to this task. From the project structure, calculations will be done in order to find the longest sequence of tasks, which is called the critical path. These tasks are essential to finish the project within the time frame that the CPM indicated. If one of the tasks on the critical path is delayed, the entire project duration increases as well, whereas non-critical tasks do not necessarily delay the entire project.

Mathematically, we can define the tasks of a construction project as a set  $J = \{1, 2, \dots, n\}$ , where  $n$  is the number of tasks. The relations of the tasks are represented as follows:  $i \rightarrow j$ , with  $i, j \in J$ , meaning that task  $i$  has to be fully completed before task  $j$  can start. In addition, the duration of each task is known and given by  $d_i \in \mathbb{N}$  with  $i \in J$ .

The CPM starts with graphically representing the project. In order to do this, each task is represented as a node. In this example, the set of nodes is  $J = \{1, \dots, 5\}$ . To illustrate, in Figure 1, task 3 can only be started after task 2 and 4 are finished. If there is no relation to any of the other tasks, then the task can be executed anywhere on the timeline. Note that the notation differs from the typical CPM. Usually in CPM, the tasks are represented as arrows and the nodes are so-called events where tasks start and finish.

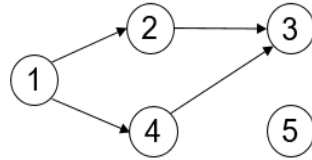


FIGURE 1: Example of a graphical representation of a project

Typically, CPM continues with forward and back tracking the completion times of the project to find the critical path. In this way, it yields the shortest possible project completion time, so the longest sequence of tasks in duration. This algorithm runs in polynomial time, so the problem is easily solved [2, 4].

However, CPM does not consider release dates of tasks in case that a task can only be started after some period of time. This might be due to the availability of materials. Also, in CPM, there are no deadlines for tasks. Realistically, it may occur that certain tasks have to be finished before a certain time, as it, for instance, can incur costs if the task is late, but CPM does not consider this. Furthermore, CPM cannot take into account any resource constraints. It may happen that the shortest project completion time is very small, as the dependencies allow all tasks to be planned parallel to each other, but if the project has fewer resources than tasks that need to be executed at the same time, then the project cannot be finished in the completion time calculated by the CPM. This is demonstrated with a small example in Figure 2. Here, the duration of each task is denoted under each node. The critical path, also the longest duration of a path, is indicated in red. The critical path goes along nodes  $1 \rightarrow 3 \rightarrow 5$



and the project completion time is the sum of the corresponding durations, so 13 time units. With unlimited resources, the project can be finished within this time. However, assume that there are only 2 workers available during the execution of the project. One of them needs to execute the tasks from the critical path, leaving the other with tasks 2 and 4 which have a total duration of 14 time units. The non-critical tasks together have a longer duration than the critical path itself, so the actual minimal project duration is 14 time units. This means that the project completion time calculated by the CPM was too low considering the resource constraints, and therefore it does not represent the actual project duration. In general, it is clear that the completion time is always at least  $\sum_{i \in J} \frac{d_i}{m}$ , rounded up, if  $m$  is the number of resources.

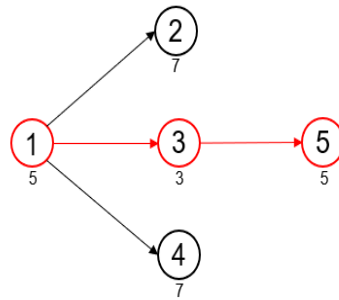


FIGURE 2: Example of a case where CPM fails to consider resource constraints

## 4 Critical Path Method with resource constraints

This section investigates whether the critical path method can be altered in such a way that it respects possible resource constraints as well. We consider the general case, where  $m$  resources are available during project. In order to schedule the tasks, they need to be divided among the resources. This is done with a multiway number partition problem. After that, the tasks are scheduled in an optimal sequence per resource. In addition, we look at a case where tasks have release dates. This means that tasks may become available during the project, instead of all tasks being available at the start of the project. As before, this procedure yields a sequence in which the tasks need to be performed and a total project completion time, but it allows pre-emption in the last step of the procedure.

### 4.1 CPM with $m$ resources and no release dates

In this particular case, we consider  $m$  resources at all times and each resource can execute any task, so there are no specialised resources. Also, it is assumed that only one resource at a time is allowed to work on a task and pre-emption is not allowed and that all tasks are available from the start of the project. The aim is planning the tasks in such a way that the total project completion time is minimized. By using a multiway number partition problem, the tasks can be sorted into  $m$  sets of independent tasks. Also, each task has a deadline  $p_i$ , which indicates when a task is considered late. This will be needed when choosing tasks in one of the next steps when the tasks will be sequenced.

#### 4.1.1 Method

In order to separate the tasks into  $m$  independent sets, we consider the maximal resource duration to be  $K$  time units [3]. This means that the sum of the durations of the tasks scheduled at one resource cannot exceed  $K$ . It does not imply that the resource has to be occupied from  $t = 0$  to  $t = K$ . The size of  $K$  can be determined by using a bisection method [5]. Each step bisects the interval  $\sum_{i \in C} d_i$  and selects the largest interval. This continues until it is large enough to fit the set of tasks. The size of  $K$  cannot exceed  $\sum_{i \in C} d_i$ , as this is the sum of all the task durations which are all scheduled at one resource. Note that, if it is possible to plan the tasks in exactly  $\sum_{i \in C} \frac{1}{m} d_i$ , then this is an optimal solution. This means that all resources are in use at all times, so the project cannot be completed any faster without increasing resources.

Let us start with a maximal resource duration of size  $K$ . If all tasks are independent, then the problem reduces to a multiway number partition problem. This is a well-known mathematical problem, which splits a set of non-negative integers in such a way that the difference of the sum of the sets is as small as possible. This problem is an NP-hard problem, but there exist greedy algorithms that can solve it [8, 14, 18]. The subset with the largest sum of task durations is defined as the critical path and the sum of durations defines the total project completion time.

We are particularly interested in dependencies between tasks. In order to divide those among the resources, the tasks are made independent of each other, which reduces the problem with dependencies to the multiway number partition problem. A dependency, such as  $i \rightarrow j$  can also be considered as a merged, larger task  $(i, j)$ , with the associated duration of  $d_{i,j} = d_i + d_j$ . In case of a longer chain of dependencies, they can also be merged together into one larger task. For instance, suppose  $i_1 \rightarrow \dots \rightarrow i_n$  can be merged into task  $(i_1, \dots, i_n)$  with duration  $d_{1, \dots, n} = \sum_{i=1}^n d_i$ . Note that chaining together tasks is only done when tasks  $\{i_2, \dots, i_n\}$  have in-degree of exactly one.

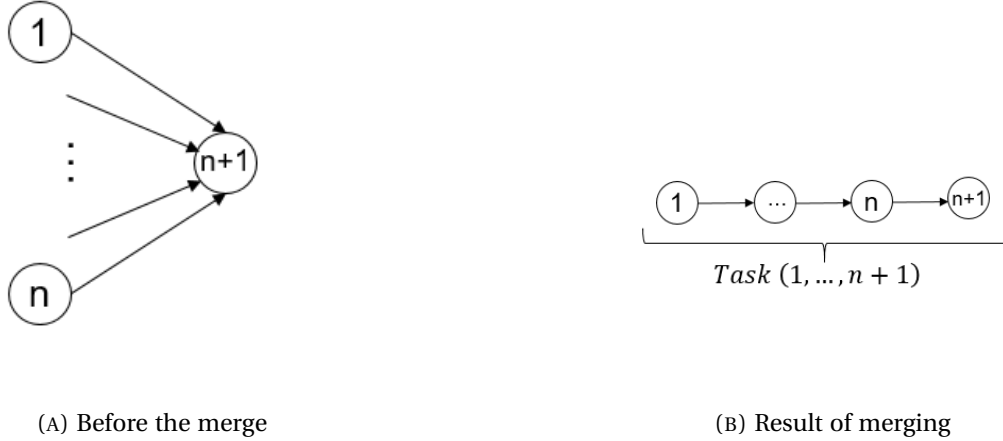


FIGURE 3: Merging a tree-like structure of dependent tasks

It can also happen that a task has multiple immediately preceding tasks, implying that a node has an in-degree of more than 1. This is also merged into one larger task by putting the preceding tasks in a row like before. To visualize this idea, node  $n + 1$  with in-degree  $n$ , see Figure 3(A), is merged to task  $(1, \dots, n + 1)$  by placing all parallel tasks behind each other, as can be seen from Figure 3(B). The order in which this happens does not matter for the total project. It can be speculated whether moving tasks behind each other is the best approach, as the preceding tasks could have been done parallel as well, but this topic will be covered later. It is also possible to merge more complicated structures, as they are always a combination of chains and tree-like structures, which was explained earlier.

By merging all the dependent tasks, only independent tasks will be left over. We end up with the partition problem that has been explained before. It will yield  $m$  independent sets of tasks. As there are  $m$  resources, each resource will handle one set of tasks.

Now that the tasks have been divided among the resources, it is a matter of deciding the order of the execution of the tasks. To determine the order, a sequencing algorithm of a single machine is used [12].

As mentioned earlier, each task also has an associated deadline,  $p_j$ . If a task is not finished by that time, the task is late and lateness should be minimized. Deadlines are used to decide the order of the tasks when there are multiple candidates that can be planned at a certain time. This can be represented by a cost function depending on time,  $c_j(t)$  as can be seen in Equation (3) and Figure 4. Here, the cost is the number of time units the task is tardy.

$$c_j(t) = \begin{cases} t - p_j, & t \geq p_j \\ 0, & t < p_j \end{cases} \quad (3)$$

The maximal lateness,  $C_{\text{total}}$ , of the project must be minimized, so that the project completion time is minimized. This is also written as  $C_{\text{total}} = \max_{i \in C} \{\min_i \{c_i(t_i)\}\}$ , where  $c_i(t_i)$  are the costs that are incurred for finishing task  $i$  at time  $t_i$ . By Lawler, the minmax can be found by applying the "last-to-first"-rule. This means that the tasks are sequenced from the latest to the earliest deadline [13]. So, the currently available task, i.e. tasks without successors, with the

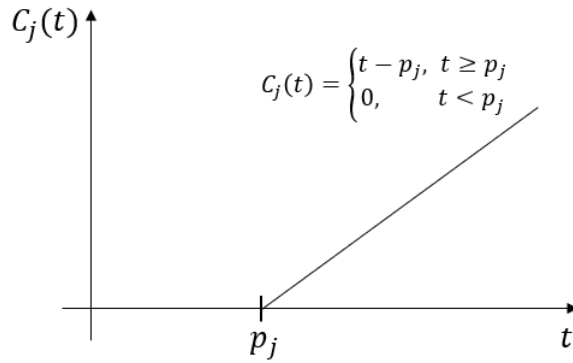


FIGURE 4: Example of a case where CPM fails to consider resource constraints

latest possible deadline is chosen. The algorithm can be described as follows: let  $S$  denote the subset of tasks which can be performed last, implying that these tasks are not preceding other tasks and let  $D$  be the sum of durations of all tasks in this set, so  $D = \sum_{i \in C} d_i$ . Compute for each task  $j$  in  $S$  such that  $c_j(D) = \min_{i \in S} \{c_i(D)\}$ . Compare their tardiness at time  $D$ , so  $c_j(D)$ , and choose the task  $k$  with the lowest value. Then repeat these steps for the remaining tasks in the set, namely  $C \setminus \{k\}$ , and consider the costs at time  $D - d_k$ . Continue until all tasks have been chosen. This algorithm creates an optimal sequence in  $\mathcal{O}(n)$ , where  $n$  is the number of tasks. Also, note that this way of sequencing is independent of the duration of the task.

#### 4.1.2 Numerical example

In order to illustrate the above explained procedure, a small numerical example is demonstrated here. Suppose that the project has a set of tasks  $J \in \{1, 2, \dots, 10\}$ , which have the given interdependencies as can be seen in Figure 5. Also, the durations,  $d_i$ , and deadlines,  $p_i$ , of each task  $i \in J$  are given in Table 1. All tasks are released at the start of the project, so all tasks are always available.

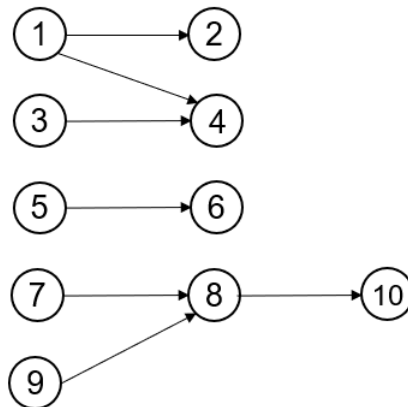


FIGURE 5: Numerical example: interdependencies between tasks

Node	1	2	3	4	5	6	7	8	9	10
$d_i$	2	4	4	2	1	4	1	2	2	2
$p_i$	3	8	6	6	3	9	2	4	2	7

TABLE 1: Numerical example: task durations  $d_i$  and deadlines  $p_i$  for each task  $i \in J$

First, the tasks are split into two independent sets of tasks by merging all chains and tree-like structures. This yields the project structure as shown in Figure 6. Accordingly, their durations are added and mentioned next to each group of tasks. The sum of all durations  $D = \sum_{i \in J} d_i = 24$  time units, so we take a maximal resource duration of 12 time units. From Figure 6, the group of tasks  $\{1,2,3,4\}$  can be placed into one set which has a total duration of 12 time units, while the other two sets  $\{5,6\}$  and  $\{7,8,9,10\}$  are combined in the other set, which has the same total duration. Note that these two sets are independent of each other, as desired.

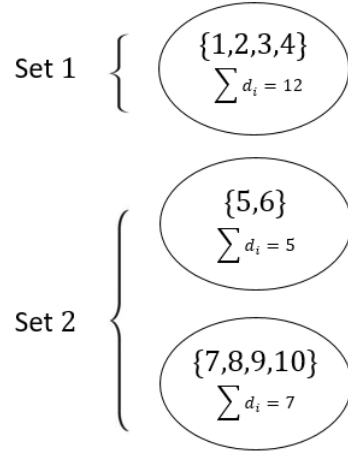


FIGURE 6: Numerical example: result after merging the tasks and their combined durations

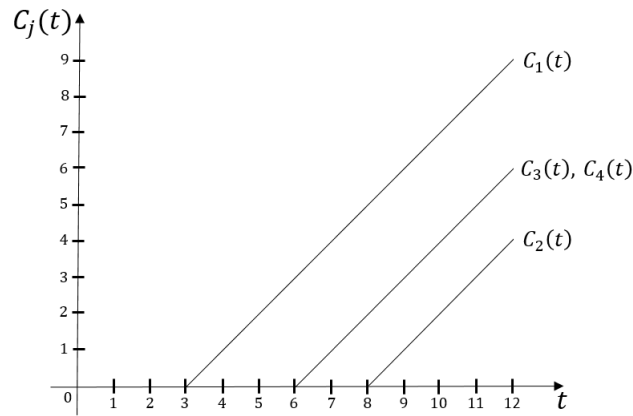


FIGURE 7: Numerical example: graph of the cost function of each task in Set 1

Next, the tasks within the set need to be sequenced. Firstly, we only focus on the set with tasks  $\{1, 2, 3, 4\}$ . To sequence them, the maximum lateness should be minimized by using the "last-to-first"-rule, while respecting the interdependencies. Therefore, it is necessary to undo the merging of the tasks, so that the interdependencies within the tasks are shown again. Additionally, the cost functions, based on the deadlines, are shown in Figure 7. At the last time point,  $t = 12$ , the candidate tasks are 2 and 4, as they do not have any successors. The tardiness of these tasks is determined by:

$$\begin{cases} c_2(12) = 4 \\ c_4(12) = 6 \end{cases}$$

Task 2 is the least tardy, and therefore most suitable to be performed last. The next candidates are tasks 1 and 4. Note that task 1 has become available, as its successor, task 2, has been finished. The lateness of the candidates is determined at time  $t = 8$ , which is the end time subtracted by the duration of the chosen task, which is  $d_2$ .

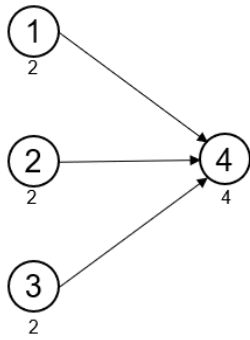
$$\begin{cases} c_1(8) = 5 \\ c_4(8) = 0 \end{cases}$$

Task 4 can be finished before the deadline, whereas task 1 is late. So task 4 is the best choice here. Now, task 1 and 3 are candidate. By computing the lateness at time  $t = 6$ , we get that task 1 is 3 time units late, whereas task 3 is executed before the deadline. Lastly, task 1 is left and will be performed first. This is before its deadline, and will not yield any lateness. After these calculations, the sequence is as follows:  $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$ . Only task 1 is late with 4 time units.

The same procedure can be applied to the other set of tasks, so  $\{5, 6\} \cup \{7, 8, 9, 10\}$ . First, the least tardy task from the candidate tasks at time  $t = 12$  is task 6 with a lateness of 3 time units. Next, task 5 or 10 can be chosen. They have the same lateness, but task 10 has the latest deadline and therefore, is the chosen task. Continuing this, it yields the following sequence:  $7 \rightarrow 9 \rightarrow 5 \rightarrow 8 \rightarrow 10 \rightarrow 6$  with a total project set duration of 12 and a tardiness of 8 time units.

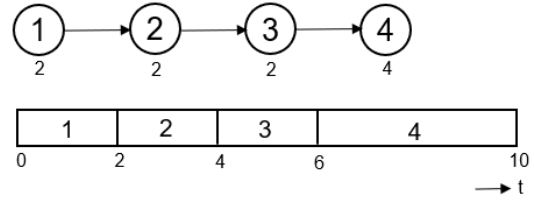
#### 4.1.3 Illustration of flaws

However, there are also cases in which this procedure does not provide an optimal solution. This happens, for instance, if there are many interdependencies in the project scheme. Suppose there are 4 tasks, each with their duration given below the node, and all tasks are precedent to task 4, see Figure 8. Suppose that there are 2 resources and all tasks are released at the start of the project. During the merging of the dependent tasks, all the tasks will be grouped together. This results into one resource which is fully occupied all the time, and one resource which is not in use at all. This makes the project completion time 10 time units. However, an optimal solution utilizes the second resource as well. This reduces the project completion time to 8 time units.



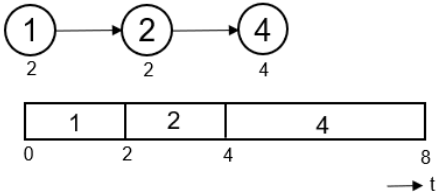
(A) Project structure

Resource A:



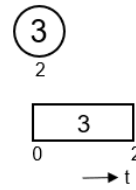
(B) Solution obtained with method from Section 4.1

Resource A:



(C) Optimal solution: Resource A

Resource B:



(D) Optimal solution: Resource B

FIGURE 8: Instance where splitting tasks does not yield an optimal solution

## 4.2 CPM with $m$ resources and unequal release dates

In practice, not all materials are available at the beginning of the project. It could happen that materials have certain so-called release dates, which restricts the earliest start time of a task. In order to take this into account, we allow pre-emption, i.e. unlimited interruptions and resuming at a later moment in time [1]. Obviously, the precedence constraints are still considered, so in case a task immediately follows another task, the precedent task has to be fully completed before the next task can start. Each task also has a deadline, which means that the task has to be completed before this time, or it is considered late. Also, lateness should be kept to a minimum. In total, there are  $m$  available resources.

### 4.2.1 Method

As done in Subsection 4.1, the tasks will be divided into  $m$  independent subsets by using the multiway number partition problem. The dependencies within the set can again be represented in a graph and each task,  $i$ , has out-degree,  $o_i$ . The cost functions  $c_i(t) \forall i \in J$  are described in Equation (3). Different from the previous section, each task  $i$  has an associated release date  $r_i$ , which indicates when a task becomes available.

In order to deal with the precedence constraints, the release dates are modified in such a way that the release date added to the task duration of the precedent task does not exceed the

release date of the successive task. Therefore, in case  $i \rightarrow j$  is a dependency, then  $r_i + d_i \leq r_j$ . This is a natural way of dealing with the precedence constraints. So, set  $r_j = \max\{r_j, \max\{r_i + d_i | i \rightarrow j\}\}$  for  $j = 2, \dots, n$ . Then order the tasks according to non-decreasing  $r_j$ . This creates a schedule consisting out of blocks, where a block  $B \subseteq C$  is defines as a set of tasks which can be executed consecutively without idle time. Now, for each block, update the out degree of each task within the block. If a block consists of only 1 task, then this is an optimal part of the schedule. In case a block consists of more than one task, make a sub-block  $B'$  with tasks that have out degree 0, i.e the task without successors. Select the task which is the least tardy at the completion time of block  $B$ . This task will be scheduled with pre-emption, meaning that this task will be scheduled in case no other task is available. The remaining tasks are scheduled as soon as they become available and the worker is available. New blocks will be created in the period where the pre-emptive task is interrupted. The same procedure applies until there are only optimal blocks left and this yields the optimal sequence of tasks with a minimal maximum of lateness.

#### 4.2.2 Numerical example

In order to illustrate this method, a small numerical example is given. Suppose the set of tasks  $J = \{1, 2, 3, 4, 5\}$  has interdependencies as depicted in Figure 9. The release dates, task durations, deadlines and initial out-degrees are given in Table 2(a).

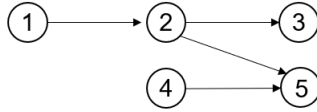


FIGURE 9: Numerical example: task interdependencies

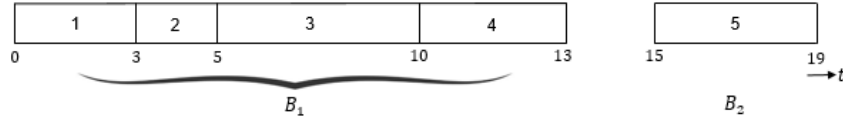
		1	2	3	4	5
(a)	$r_i$	0	0	8	4	15
	$d_i$	3	2	3	5	4
	$p_i$	5	6	11	16	20
	$o_i$	1	2	0	1	0
(b)	updated $r_i$	0	3	8	4	15
(c)	updated $o_i$ of $B_1$	1	1	0	0	-
(d)	updated $o_i$ of $B_{11}$	1	0	-	-	-

TABLE 2: Numerical example: given parameters for numerical example

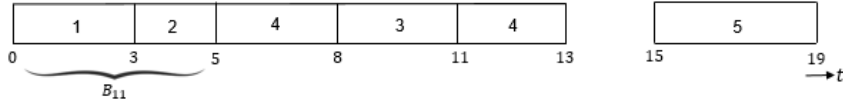
We begin with modifying the release dates, so that  $r_j \geq r_i + d_i$  if  $i \rightarrow j$ . The outcome is given in row (b) of Table 2. As we have taken care of the precedence constraints, the tasks can now be scheduled in order of increasing release dates, so in the order: 1, 2, 4, 3, 5. This is illustrated in Figure 10(A). The initial schedule can be split into different blocks:  $B_1 = \{1, 2, 3, 4\}$  from  $t = 0$  to  $t = 13$  and  $B_2 = \{5\}$  from  $t = 15$  to  $t = 19$ . The blocks are separated by idle time of the resource. Note that the block consisting of exactly one task is optimally scheduled.

The planning inside block  $B_1$  can still be optimized by using pre-emption. The updated out-degrees within this block are given in Table 2(c). Task 3 and 4 have out-degree 0, which means that they do not have any successors within the block. Therefore, they are candidates to be the





(A) Initial schedule of numerical example



(B) Updated schedule of numerical example

FIGURE 10: Numerical example: task scheduling with pre-emption

pre-emptive task, i.e the task that is scheduled if no other task is available. The pre-emption is used to reduce the lateness within a block. To decide, the lateness of the candidate tasks is compared at  $t = 13$ :

$$\begin{cases} c_3(13) = 2 \\ c_4(13) = 0 \end{cases}$$

Task 4 is the best choice, as it will be executed before the deadline, whereas task 3 would be late. The initial schedule can be updated by scheduling tasks 1, 2 and 3 as soon as they become available and scheduling task 4 if no other task is available. This can be seen in Figure 10(B). Note that the execution of task 4 has been split up into two intervals.

Next, we look for sub-blocks that can be optimized. These are created in the intervals where the pre-emptive task is not scheduled. Within block  $B_1$ , a sub-block  $B_{11}$  from  $t = 0$  to  $t = 5$  is created. The updated out-degrees in this sub-block are given in Table 2(d). Task 2 is the only task without any successors. Therefore, this task is the pre-emptive task of this sub-block. However, it cannot be optimized further as task 1 and 2 are already scheduled as soon as possible. Therefore, Figure 10(B) is the optimal sequence of this part of the project. Note that allowing pre-emption reduces the tardiness of the project. If we would schedule according to increasing release dates without pre-emption, we obtain the schedule as shown in 10(A). In this schedule, task 3 is late with 2 time units, whereas in the updated schedule as shown in 10(B), all tasks are performed before their deadlines. Therefore, allowing pre-emption is beneficial for the scheduling of projects with deadlines.

## 5 Conclusion

The classical CPM is a useful tool for scheduling projects, such as house construction projects, but it would be more useful if it can also consider practical limitations. For example, it is not realistic to assume unlimited resources during the entire project or to assume that all tasks are available from the start of the project. Therefore, the CPM can be altered such that it can consider these factors. For a resource limitation of  $m$  workers, the tasks should be divided among the workers strategically, while considering the precedence constraints. This can be done by splitting the tasks into  $m$  independent sets, so that the workers can execute their tasks independently of each other and finish the project in the shortest possible time. In order to do so, the well-known mathematical concept of the multi-way number partition problem is used. If pre-emption, i.e. interruption of tasks, is allowed, it is fairly easy to consider the release dates of tasks as well. Though this is still an NP-hard problem and it might not work for some instances, it can still be used as a tool for scheduling tasks with  $m$  resources. However, it is not always optimal, so future studies may research methods in which it is always optimal. Also, future research can attempt to find methods that omit NP-hard problems.

## 6 Discussion

Though this paper has found a useful way to schedule a task-based project with precedence constraints and release dates in the case of limited resources, it still has its limitations and opportunities for further research. These will be discussed in this section.

### Limitations of the method

As mentioned before, the method of splitting tasks between the resources may not work for all kinds of project structures. In Section 4.1.3, it is already demonstrated that the method fails in case all tasks are dependent of each other. By using the method, it will merge all tasks, leaving other resources fully unused. This can be solved by adding an extra step after the splitting of the tasks. This checks whether resources have some unscheduled time and if a task can be moved from one resource to the other. Note that this task has possible dependencies on other tasks in the set where it was initially scheduled. These still need to be considered while rescheduling.

Similar to this, the method does not schedule tree-like structures, as depicted in Figure 3a, optimally. During the merging of these kind of structures, all tasks involved are placed in a row in order to make two independent sets of tasks. The tasks could have been executed parallel if the number of resources allows it, but this implies that the sets of tasks are not independent of each other. The method could be adapted in such a way that this is allowed, which may reduce the minimum project time.

In addition, the computational boundaries of this method are not tested, so the size of the project structures the method can handle is unclear. It should be mentioned that the method is NP-hard, so it can in principle not be calculated in polynomial time. Future research may look into heuristics that make scheduling possible in acceptable time.

### Opportunities for further research

Even though this method has added to the field of scheduling, there are still a lot of, perhaps even endless, opportunities for future research. Realistically, workers are often specialized, meaning that not every worker is available for every task. Moreover, some tasks may require more than 1 worker at a time, so a task may occupy multiple resources at the same time. To further extend this, the number of resources that are working on a task may also be a choice. The more resources on a task, the faster the task is done. It may be beneficial to schedule more resources for a certain task with a strict deadline for example. These extensions may be considered during the scheduling in order to portray the reality of a project better.

## References

- [1] K.R. Baker et al. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 31:381–386, 1981.
- [2] Paul E. Black. Np-complete. <https://www.nist.gov/dads/HTML/npcomplete.html>, 2021.
- [3] G. Borradaile et al. The knapsack problem with neighbour constraints. 2018.
- [4] Pawel Chanas, Stefan; Zielinski. The computational complexity of the criticality problems in a network with interval activity times. *European Journal of Operational Research*, 136:541–550, 2002.
- [5] Arun Dharavath. Bisection method. <https://protonstalk.com/polynomials/bisection-method/>, 2022.
- [6] Moshe; Uriel G.Rothblum; Michal Penn Golany, Boaz; Kress. Network optimization models for resource allocation in developing military countermeasures. *Operations Research*, 60(1):48–63, 2012.
- [7] Neos Guide. Project scheduling with the critical path method. <https://neos-guide.org/content/project-scheduling-critical-path-method>, 2020.
- [8] Brian Hayes. Computing science: The easiest hard problem. *American Scientist*, 90:113–117, 2002.
- [9] John S. Sayer James E. Kelley, Morgan R. Walker. The origins of cpm: a personal history. *PM network*, 3:7–22, 1989.
- [10] Morgan R. Kelly, James E.; Walker. Critical-path planning and scheduling. *Proceedings of the Eastern joint computer conference*, pages 160–173, 1959.
- [11] Caron M. Kopp. Program evaluation review technique (pert) chart. <https://www.investopedia.com/terms/p/pert-chart.asp>, 2022.
- [12] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management science*, 19:544–546, 1973.
- [13] E.L.; Morre J.M. Lawler. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16:77–84, 1969.
- [14] Ethan L. Schreiber et al. Optimal multi-way number partitioning. *Journal of the ACM*, 65(4):1–61, 2018.
- [15] C.M.F. Swennenhuis. Parallel machine scheduling with partition constraints. Master’s thesis, TU Delft, 2018.
- [16] Bernard W. Taylor. Formulating the CPM/PERT Network as a Linear Programming Model. <https://flylib.com/books/en/3.287.1.114/1/>, 2006.
- [17] C.A.J Vredeveld, T; Hurkens. Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. *Informatics Journal on Computing*, 14:176–189, 2002.
- [18] Liang Zhou et al. On generalized greedy splitting algorithms for multiway partition problems. *Discrete Applied Mathematics*, 143(1-3):130–143, 2004.