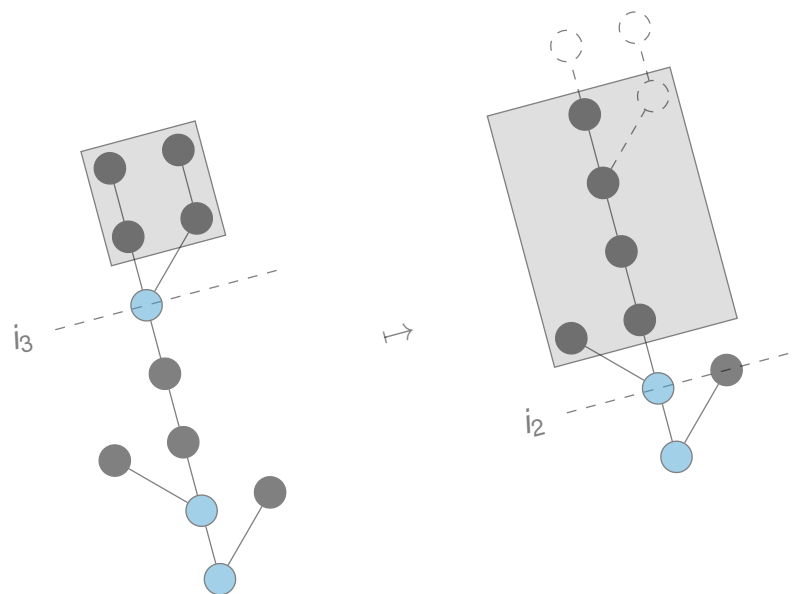




Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen

Stochastic Scheduling with Precedence Constraints

Chris Pinkau

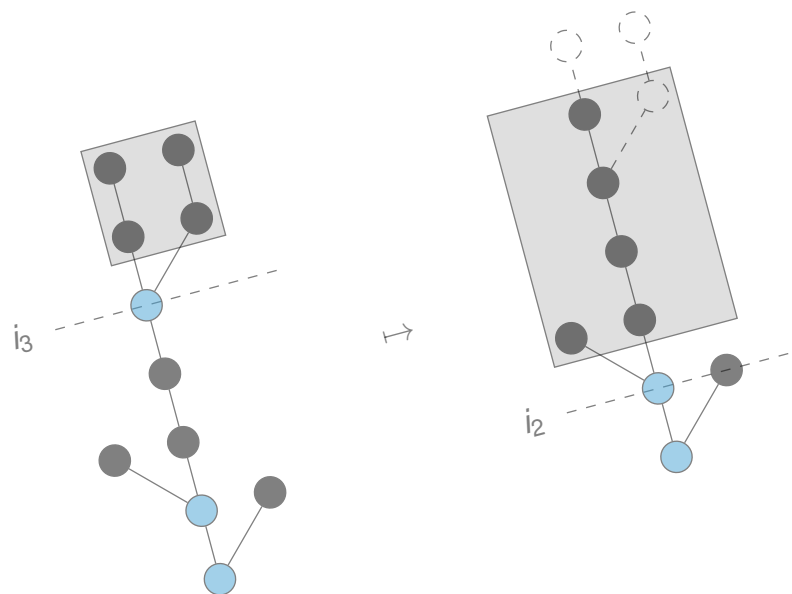




Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen

Stochastisches Scheduling mit Präzedenz-Relationen

Chris Pinkau





Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen

Stochastic Scheduling with Precedence Constraints

Chris Pinkau

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Tobias Nipkow, Ph.D.

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Ernst W. Mayr
2. Univ.-Prof. Dr. Helmut Seidl

Die Dissertation wurde am 13.06.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.01.2017 angenommen.

Abstract

Scheduling partially ordered tasks on several parallel processors has been a topic of many scientific articles over the past 40 years. Mokotoff, Pinedo, and Lawler et al give many prominent examples, some of them polynomially optimizable, others **NP**-hard. The landscape of scheduling is vast, changing just one variable in the problem statement can dramatically alter the process of finding results and the results themselves. One special branch of scheduling that has been considered is stochastic scheduling, with stochastic processing times for the tasks, subject to some probability distributions.

This thesis starts by examining the scheduling problem given by two processors, exponentially distributed processing times,intree precedence constraints, and (expected) makespan optimization. This specific problem has been polynomially solved many years ago by Chandy and Reynolds, but, to our knowledge, has not been generalized to other distributions yet. This thesis describes an optimal solution for the geometric distribution, and provides insights and experimental results for the uniform and Erlang distributions.

We shed light on some strategies for general precedence constraints and the extension to three processors by indicating characterizations of the proposed scheduling problems that may lead to finding optimal strategies in future work.

Zusammenfassung

Seit nun schon über 40 Jahren ist Scheduling von partiell geordneten Prozessen auf parallelen Systemen Thema zahlreicher wissenschaftlicher Arbeiten. Mokotoff, Pinedo, und Lawler et al nennen einige der berühmtesten Beispiele, viele davon lösbar in Polynomialzeit, andere wiederum sind **NP**-schwer. Das Spektrum von Scheduling ist reichhaltig, das Ändern einer Variable in der Problembeschreibung kann zu großen Unterschieden in den Ergebnissen führen. Ein großes Teilgebiet ist dabei das stochastische Scheduling, bei dem die Ausführungszeiten der Prozesse stochastischer Fluktuation unterliegen, d.h., sie sind nicht deterministisch gegeben, sondern durch Zufallsvariablen mit zugrundeliegenden Wahrscheinlichkeitsverteilungen beschrieben.

Das Ausgangsproblem dieser Arbeit ist das Scheduling-Problem, das gegeben ist durch zwei Prozessoren, exponentiell verteilte Prozesszeiten, Präzedenzen in Form von Intrees, und Optimierung der totalen (erwarteten) Laufzeit. Zu diesem Problem gibt es schon seit einiger Zeit eine Polynomialzeit-Lösung von Chandy und Reynolds, doch ist es bisher (nach unserer Erkenntnis) noch nicht auf andere Verteilungen erweitert worden. Diese Arbeit zeigt den Beweis einer optimalen Lösung für geometrisch verteilte Prozesszeiten, und fasst Einblicke und experimentelle Ergebnisse für gleichverteilte und Erlang-verteilte Prozesszeiten zusammen.

Außerdem befassen wir uns mit verschiedenen Strategien für allgemeine Präzedenzen und die Erweiterung des Problems auf mehr als zwei Prozessoren, bei denen die genaue Untersuchung von nicht-optimalen Strategien und deren Charakteristika zum Auffinden einer optimalen Strategie für zukünftige Arbeiten führen könnte.

Contents

Abstract	ii
List of Figures	ix
Part I Introduction & Motivation	1
Part II Definitions & Concepts	5
1 General Scheduling Concepts	7
2 Task Characteristics	9
3 Optimality Criteria	10
4 Notation	12
5 Reductions between Scheduling Problems	12
6 The Highest-Level-First Scheduling Strategy	13
7 Configuration	14
8 Profile	15
Part III Mathematical Concepts	17
9 Sums and Series	19
10 Power Series and Generating Functions	21
11 Hypergeometric Summation	23
12 Integrals	24
13 Probability Theory	24
Part IV The Expected Makespan	29
14 Discrete Case	31
15 Continuous Case	32

Part V The Exponential Distribution	33
16 Calculating the Expected Makespan	36
17 The Optimal Scheduling Strategy	38
18 Formulas for the Expected Makespan	44
18.1 Naive Approaches	45
18.2 Using the Configuration Relation	48
18.3 Using the Profile Relation	51
18.4 Using the $\{h, b\}$ values	52
18.5 One Chain	57
18.6 Two Chains	57
18.6.1 Using the Combinatorial Approach to Find a Closed Form	57
18.6.2 Using Generating Functions to Find a Closed Form	60
18.7 k Chains	66
18.7.1 Using the Combinatorial Approach	66
18.7.2 Using Generating Functions for Specific Examples	70
18.7.3 Using a Stair Function for the General Case	71
18.8 Y-graphs and Psi-graphs	80
18.9 Intree Decomposition	81
Part VI The Geometric Distribution	87
19 Calculating the Expected Makespan	90
20 The Optimal Scheduling Strategy	93
Part VII The Uniform Distribution	97
21 Discrete Case	99
22 Continuous Case	102
23 Calculating the Expected Makespan	104
23.1 Discrete Case	105
23.2 Continuous Case	108
24 HLF and Uniform Processing Times	110

Part VIII The Erlang Distribution	113
25 Calculating the Expected Makespan	116
26 The Generalized Erlang Distribution	120
27 <i>HLF</i> and Erlang processing times	121
Part IX General Precedence Constraints	129
28 Directed Acyclic Graphs	131
28.1 The Coffman-Graham Algorithm is not Optimal	131
28.2 The Chandy-Reynolds Algorithm is not Optimal	137
28.3 Static, Semi-Static, and Dynamic Scheduling Strategies	139
28.3.1 Level and Number of Descendants of a Task	143
28.3.2 Number of Induced Paths of a Task	147
28.4 Choosing Pairs of Sources	149
29 Calculating the Expected Makespan	154
30 sp-graphs	157
Part X Three Processors	161
31 Difference to the Deterministic Setting	163
32 Minimal Counterexamples	163
32.1 Supergraphs of Counterexamples	166
32.2 Y-subgraphs of Counterexamples	167
32.2.1 Fix one Source	167
32.2.2 Fix two Sources	168
32.2.3 Probability of Reaching Certain Y-subgraphs under Different Strategies	170
33 Times of Busy and Idle Processors	173
34 More Differences to the Two Processor Case	174
35 <i>HLF</i> is Asymptotically Optimal	174

Conclusion	177
Appendix Detailed Calculations	179
A Equivalence of Approaches for Exponential Processing Times	180
B Equivalence of Approaches for Geometric Processing Times	181
C Induction Bases for Exponential Processing Times	183
D Induction Bases for Geometric Processing Times	185
E Induction Bases for Uniform Processing Times	188
F Induction Bases for Erlang Processing Times	192

List of Figures

Fig. 1	A Gantt chart.	8
Fig. 2	An intree with 10 tasks.	9
Fig. 3	An inforest and the corresponding intree with a supersink.	10
Fig. 4	Examples of configurations.	15
Fig. 5	Intrees with different profiles.	16
Fig. 6	Crucially different dominating paths.	40
Fig. 7	The profile does not say anything about the number of available tasks.	40
Fig. 8	Four intrees with $G_2 \propto G_1$, $G_2 \propto G_3$, $G_1 \sim G_3$, and G_4 is incomparable.	41
Fig. 9	G and G' and their respective successor configurations.	42
Fig. 10	Different heads of the chain for the same intree.	45
Fig. 11	An intree with profile $(1, 2, 3, 3)$	46
Fig. 12	Hasse diagram for the configuration relation of the intree in Fig. 2.	49
Fig. 13	Four isomorphic intrees.	50
Fig. 14	A configuration and its successor configurations.	51
Fig. 15	Successor configurations can have different profiles.	52
Fig. 16	Hasse diagram for the profile relation of the intree in Fig. 2.	53
Fig. 17	Two different intrees with the same $\{h, b\}$ values.	54
Fig. 18	Hasse diagram of the $\{h, b\}$ -relation for the intree in Fig. 11.	56
Fig. 19	The lattice for the computation of a_{ij} via dynamic programming.	61
Fig. 20	Two configuration paths that lead from $[2, 1, 1]$ to $[1, 0, 0]$	67
Fig. 21	The chain grid for $[5, 5, 4, 2]$	72
Fig. 22	Corresponding grids to some examples.	74
Fig. 23	The base cases of the recurrence relation	75
Fig. 24	Examples for (18.21) and (18.22) on the left and right, respectively.	75
Fig. 25	The grids to be considered in the example from Fig. 21.	76
Fig. 26	The configuration graph for $[4, 4, 3, 2]$	79
Fig. 27	A Y-graph $Y_{r,s,t}$	80
Fig. 28	A Psi-graph Ψ_{r,s_1,\dots,s_k}	81
Fig. 29	An intree with width 2 and three join nodes on join levels i_1 , i_2 , and i_3	82
Fig. 30	HLF applied to the intree in Fig. 29.	82
Fig. 31	An intree with decomposition (G_1, \dots, G_4)	83
Fig. 32	The chain grid for $[4, 3, 3, 2]$ and its simplified version without labels.	85
Fig. 33	G and G' and their respective successor configurations.	95
Fig. 34	Case A : $i < a$ and $i > b - a$	101
Fig. 35	Case B : $i < a$ and $i < b - a$	101
Fig. 36	Case C : $i \geq a$ and $i > b - a$	101
Fig. 37	Case D : $i \geq a$ and $i < b - a$	101
Fig. 38	Case E : $i < a$ and $i = b - a$	102
Fig. 39	Case F : $i \geq a$ and $i = b - a$	102

Fig. 40 Case A : $i < a$ and $i > b - a$.	103
Fig. 41 Case B : $i < a$ and $i \leq b - a$.	104
Fig. 42 Case C : $i \geq a$ and $i > b - a$.	104
Fig. 43 Case D : $i \geq a$ and $i \leq b - a$.	105
Fig. 44 The reduction from Erlang to exponential processing times.	121
Fig. 45 The configurations G and G' and their respective successor configurations.	124
Fig. 46 G , G' , and their successor configurations.	124
Fig. 47 G , G' , and their successor configurations.	125
Fig. 48 G , G' , and their successor configurations.	126
Fig. 49 G , G' , and their successor configurations.	127
Fig. 50 A <i>dag</i> and its corresponding optimal schedule.	132
Fig. 51 Part of the configuration graph for the Coffman-Graham scheduling strategy.	133
Fig. 52 Part of the configuration graph for the optimal scheduling strategy.	134
Fig. 53 Two degenerated configurations.	135
Fig. 54 A <i>dag</i> 's χ -blocks are processed in order in an optimal schedule.	136
Fig. 55 Part of the configuration graph for an <i>HLF</i> .	138
Fig. 56 Different <i>HLFs</i> result in different makespans.	139
Fig. 57 The levels are static values.	140
Fig. 58 The α -values are static values.	140
Fig. 59 The number of tasks on the same level is semi-static.	140
Fig. 60 The number of ancestors of a task is semi-static.	141
Fig. 61 The tuple of the number of ancestors and descendants of a task is semi-static.	141
Fig. 62 Part of a configuration graph for the strategy using the number of tasks on the same level.	142
Fig. 63 Part of a configuration graph for the strategy using the number of ancestors.	143
Fig. 64 Part of a configuration graph for a strategy always choosing one of the highest sources.	144
Fig. 65 The tuple of the level and the number of descendants of a task.	144
Fig. 66 The optimal strategy for these graphs implies that $2 3 \prec 3 2$.	145
Fig. 67 Four graphs that show (from left to right): $2 4 \prec 3 3$, $3 3 \prec 2 5$, $2 5 \prec 3 4$, and $3 4 \prec 2 6$.	145
Fig. 68 The order of some $x y$ values.	146
Fig. 69 The four different possibilities for a task with values $3 4$.	146
Fig. 70 The tuples $5 5$ and $3 8$ are incomparable.	147
Fig. 71 Only one edge makes the difference for the optimal strategy.	147
Fig. 72 Even for a big difference in the number of edges, $3 16 \prec 3 17$.	148
Fig. 73 The number of induced paths of a task as a priority.	148
Fig. 74 Part of the configuration graph of an optimal <i>HLF</i> schedule for the left graph in Fig. 71.	150
Fig. 75 Part of the configuration graph of a non-optimal <i>HLF</i> schedule for the right graph in Fig. 71.	151
Fig. 76 Part of the configuration graph of an optimal schedule for the right graph in Fig. 71.	152

Fig. 77	A <i>dag</i> with 16 tasks, which is optimally scheduled by a non- <i>HLF</i> .	153
Fig. 78	Induced subconfigurations of pairs of sources do not hold any value as a priority.	153
Fig. 79	An <i>M</i> -graph and <i>i</i> singular tasks.	154
Fig. 80	A <i>dag</i> consisting of two <i>M</i> -graphs.	154
Fig. 81	The recursive reduction of a sequence of three <i>M</i> -graphs.	156
Fig. 82	An incomplete bipartite graph with $j + i$ nodes.	157
Fig. 83	The level-2 task with 16 successors is prioritized over the level-7 tasks, i.e., $7 \mid 6 \prec 2 \mid 16$.	157
Fig. 84	The parallel and series compositions of two different two-terminal sp-graphs.	159
Fig. 85	A counterexample to <i>HLF</i> for sp-graphs.	159
Fig. 86	Not all <i>dags</i> can be created as an sp-graph.	160
Fig. 87	One of the minimal counterexamples to <i>HLF</i> .	164
Fig. 88	Another smallest counterexample to <i>HLF</i> .	165
Fig. 89	Supergraphs of the graph in Fig. 87 which are also not optimally scheduled by <i>HLF</i> .	166
Fig. 90	Supergraphs of the graph in Fig. 88 which are also not optimally scheduled by <i>HLF</i> .	166
Fig. 91	$G_{d,\ell}$.	167
Fig. 92	The product of probability and sum of makespans does not work as a priority.	168
Fig. 93	The sum of makespans alone does also not work as a priority.	169
Fig. 94	The values for the example graph from Fig. 92, this time with two fixed sources.	169
Fig. 95	The values for the example graph from Fig. 93, this time with two fixed sources.	170
Fig. 96	Part of the configuration graph of the graph in Fig. 88 using <i>HLF</i> .	171
Fig. 97	Part of the configuration graph of the graph in Fig. 88 using the optimal strategy.	172
Fig. 98	Optimizing T_1 or T_3 does not necessarily yield an optimal schedule.	173
Fig. 99	Same profiles do not infer same optimal strategies.	174
Fig. 100	The intree [5, 3].	180
Fig. 101	An intree with expected makespan $\frac{190}{27}$.	181

Listings

18.1	Mathematica code for (18.5). The execution yields the expression in (18.6).	60
18.2	Mathematica code for (18.7), resulting in (18.8).	60
25.1	Mathematica code for reproducing the formulas from Part V by setting $k = 1$ and/or $t = 0$ in the formulas from above.	120
F.1	Mathematica expressions for the values from case $k = 2$ from above.	194
F.2	Mathematica expressions for the values from case $k = 3$ from above.	195
F.3	Mathematica expressions for the values from Calculations 25 and 26.	197
F.4	Mathematica expressions for the values from Calculations 27 and 28.	200



Introduction & Motivation

Suppose we are given several tasks that need to be processed on several machines or processors. Then we require some *scheduling strategy* that provides an algorithm that determines the order and on which processors the tasks should be processed. The result, i.e., a listing of what every processor is doing at each time point, is called a *schedule* (More mathematically formal definitions will follow later). The landscape of scheduling is vast, Mokotoff [42] compiled a survey covering many important deterministic scheduling problems, the same is true for Lawler et al [37]. For a quick overview of deterministic as well as stochastic scheduling we refer to a book of Pinedo [47]. There are many different variables that can be tempered with to obtain thousands of very different scheduling problems.

Consider a very basic setting: there are several parallel and identical processors available, as well as some number of tasks, and the objective is to derive a schedule that minimizes the total time it takes to process all tasks on those processors. This problem is **NP**-hard [26]. Even the restriction to only two processors is **NP**-hard [39]. In fact, both problems belong to a class of combinatorial optimization problems, for many of which the hardness was also proven [16, 33]. However, restricting these problems further in certain directions may very well make them tractable, for example the problem with a fixed number of processors, unit processing times, and general precedences is still open and might be polynomially solvable, whereas the variant with the same restrictions but arbitrary number of processors is proven to be **NP**-hard [55].

All of the aforementioned results only deal with deterministic scheduling, whereas, in this work, we focus on stochastic scheduling, meaning that the processing times of the tasks – i.e., the time needed for the tasks to be fully processed – are not fixed constant values, but rather take values according to some probability distributions. The two easiest distributions (apart from trivial distributions) that can be taken into account are the geometric distribution in the discrete case and the exponential distribution in the continuous case. The simplicity comes from the fact both are memoryless (see Parts V and VI for a definition and further results), and they are the only two distributions that have this property.

Furthermore, it may be the case that not all the tasks are available in the beginning, so a valid scheduling strategy may not choose from all tasks, but only from all available tasks. We impose certain restrictions on the set of tasks, called *precedence constraints* (or *precedence relations* or just *precedences*), which describe that some tasks need others to be fully processed before they can be chosen for being processes themselves. As an example, consider a cooking recipe. For some steps in the recipe, others have to precede them, e.g. noodles may only be put into the pot when the water is already boiling, so the task “heat water until it boils” has to precede the task “put noodles into water”, which in turn has to precede “mix noodles with sauce”, and so on. There are many different scenarios for precedence constraints as well. Some task may be allowed to depend on more than some other task, or may be critical for more than one other task as well. However, one relationship should not be allowed for the precedence constraints: a mutual dependence of two (or more) tasks, meaning that one task would need the other to be fully processed to be available, and vice versa.

The overall goal of such a scheduling strategy is to optimize some objective function. There are many possibilities to define such a function. Probably, the most natural way would be to try to minimize the total time the schedule needs, i.e., the time point at which the last task is finished on any processor. This is often called the *makespan*, and is the objective function we will use in all of our problems. We will mention a few more common objective functions later on in Chapter 3.

In all cases in this work, we will not impose more restrictions on our problems. So, for example, we will not allow:

- *preemption* (a processor that is processing a task may interrupt its current work to process another task and it or another processor continues processing the interrupted task later on),
- *shop* scheduling (a task has to be processed by several processors in a certain order),
- that the processors have different speeds with which they can process the tasks,
- that some tasks may only be scheduled on certain processors,
- deadlines, meaning that tasks have to be finished by a certain time point.

The basic scheduling problem that is the origin of all this work is the case with two parallel, identical processors, processing times which are all independent and identically distributed according to an exponential distribution with mean value 1, and an underlying precedence relation which forms an *intree* (for the definition, see Chapter 2). This means that no task can have more than one task that directly depends on it, but may itself depend on any number of other tasks. The objective is to minimize the expected makespan, i.e., the total expected time needed to completely process all tasks. The optimal scheduling strategy for this particular problem was proven some decades ago by Chandy and Reynolds [12], but many similar problems are still open. We try to find simplified and/or closed formulae for the already proven parts and solutions for open problems. These open problems include assuming other precedence relations, other distributions for the processing times, a higher number of processors, or a combination of those. The first part of this thesis provides an introduction to the topic of scheduling. In the second part, we define the basic concepts of scheduling theory and introduce our notations of these concepts. Part three deals with the mathematical basics such as sums, series, or probability theory. The next short part is devoted to the general way of calculating the expected makespan. With part five, we begin to examine certain stochastic scheduling problems, with the two processor problem using exponentially distributed processing times and intree precedence constraints being the first to be considered in this part. The next part is a brief discussion of its discrete counterpart, using geometrically distributed processing times instead. Parts seven and eight consider yet two other distributions and their implications: the uniform distributions, and the Erlang distribution. Part nine follows with a discussion and simulations for general precedence constraints, followed by part ten which is about using three processors. The last part sums up and concludes this work.



Definitions & Concepts

Table of Contents

1	General Scheduling Concepts	7
2	Task Characteristics	9
3	Optimality Criteria	10
4	Notation	12
5	Reductions between Scheduling Problems	12
6	The Highest-Level-First Scheduling Strategy	13
7	Configuration	14
8	Profile	15

1 General Scheduling Concepts

First, we will have some definitions to properly classify our problem(s).

Definition 1 (Task). A *task* (or *job*) is given by an identifier (*id*) or a name and its *processing time* which describes the time it takes a processor of unit speed to fully process this task.

Usually, the tasks are given as a set, and their names are identical to their respective *ids*, and given as a list (x_1, \dots, x_n) . Throughout this work we use p_i as the processing time of the task with *id* x_i (they share the index i), and n as the total number of tasks (unless specifically stated otherwise). The set of all tasks is denoted by \mathcal{T} .

A task is called *available* at a given time point if it can be chosen to be processed by a processor at that time point. Reasons why a task might not be available can be seen in Chapter 2. A task is called *active* at a given time point if it is currently being processed by a processor at that time point.

In deterministic scheduling, the processing times of the tasks are fixed. They may differ from one task to another, but they will always be known values. In the case of stochastic scheduling, the processing times of the tasks are given by random variables, and may take on many different values in general. Scheduling problems may differ greatly whenever the constraints on the tasks' processing times are altered (in deterministic and in stochastic scheduling). In this work, the random variables are all independent and identically distributed according to some well-known probability distributions. Different distributions are studied, as well as their commonalities and differences.

What follows next is a definition that deals with the question of when exactly an algorithm has to choose a task.

Definition 2 (Decision Point). A *decision point* is a point in time when a processor finishes processing a task and becomes ready for processing a next one. A processor which is ready for processing a task is called *idle*.

In preemptive scheduling, every time point can potentially be a decision point. However, when preemption is not allowed, the above definition completely characterizes decision points. In the following, we will use that definition.

Definition 3 (Scheduling Strategy). Let \mathcal{T} be a set of tasks. Given a set of m processors, a *scheduling strategy* determines, at any decision point, which available task (if any) is assigned to which idle processor.

In all of our cases, we will deal with identical processors. Because of this, it is of no use for us to distinguish between them. Although a general scheduling strategy would assign an available task to a specific idle processor, we just say that a task is being scheduled and processed by some processor. There may also be another layer introduced in this process by having a *Processor Allocation Strategy* that assigns chosen available tasks to processors in a specific manner. As mentioned before, this extra layer could be useful when dealing with non-identical

processors. In our case, we can have any processor allocation strategy, which is why we do not mention it any further at all.

Definition 4 (Schedule). A *schedule* is a map $S : \mathbb{R}_0^+ \mapsto (\mathcal{T} \cup \{\emptyset\})^m$, that is determined by an underlying scheduling strategy and denotes, at any point in time, the allocation of the processors to the tasks.

In our case of non-preemptive scheduling, a schedule could also be defined as a list for each processor, where each list contains the starting times of all tasks that are processed on this processor along with their ids.

Schedules are often depicted by *Gantt charts* (invented/formalized by Clark in [13]). The Gantt chart of a possible schedule for some set of tasks $\mathcal{T} = \{x_1, \dots, x_{10}\}$ can be seen in Fig. 1.

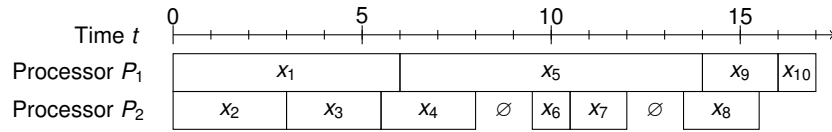


Fig. 1: A Gantt chart.

Using the mathematical definition above, the schedule seen in Fig. 1, can also be given by

$$S(t) = \begin{cases} (x_1, x_2) & \text{for } t \in [0, 3) \\ (x_1, x_3) & \text{for } t \in [3, 5.5) \\ (x_1, x_4) & \text{for } t \in [5.5, 6) \\ (x_5, x_4) & \text{for } t \in [6, 8) \\ (x_5, \emptyset) & \text{for } t \in [8, 9.5) \\ (x_5, x_6) & \text{for } t \in [9.5, 10.5) \\ (x_5, x_7) & \text{for } t \in [10.5, 12) \\ (x_5, \emptyset) & \text{for } t \in [12, 13.5) \\ (x_5, x_8) & \text{for } t \in [13.5, 14) \\ (x_9, x_8) & \text{for } t \in [14, 15.5) \\ (x_9, \emptyset) & \text{for } t \in [15.5, 16) \\ (x_{10}, \emptyset) & \text{for } t \in [16, 17) \\ (\emptyset, \emptyset) & \text{for } t \in [17, \infty). \end{cases}$$

Equivalently, the list of starting times for this schedule are

$$(0, x_1), (6, x_5), (14, x_9), (16, x_{10})$$

for the first processor P_1 , and for the second processor P_2 :

$$(0, x_2), (3, x_3), (5.5, x_4), (9.5, x_6), (10.5, x_7), (13.5, x_8).$$

2 Task Characteristics

Definition 5 (Precedence Relation). Let \mathcal{T} be a set of tasks. A *precedence relation* (or *precedence constraints* or just *precedences*) R is a partial order on the set \mathcal{T} . If a task i has to be finished before another task j becomes available to be processed, then $(i, j) \in R$. In this case, j is a (direct) successor of i , and i is a (direct) predecessor of j .

If R contains a sequence $(k_1, k_2), \dots, (k_{r-1}, k_r)$ with $i = k_1$ and $j = k_r$ for some $r \geq 2$, then we say that j is a *descendant* of i , and i is an *ancestor* of j .

Usually, we picture a precedence relation by its corresponding *Hasse diagram* [6] with an arc from task i to task j if $(i, j) \in R^*$, i.e. (i, j) is in the transitive reduction of R . The convention is that the direction of the arcs in the diagram go from the top to the bottom, thus the arcs can be drawn as normal edges, see Fig. 2 for an example. Available tasks, i.e., tasks without predecessors, are also called *sources*.

In general, the corresponding graph to a precedence relation can be a directed acyclic graph, or *dag* for short. It is directed as a precedence relation should not be symmetric, and it is acyclic, because a cycle within a precedence relation implies that two tasks depend on each other, a situation which can never be solved. However, for most of this work, we restrict ourselves to a special case of a *dag*.

Definition 6 (Intree). Let $G = (V, E)$ be a graph. G is an *intree*, if it is acyclic and connected, and every node in V has outdegree at most 1.

Each intree G contains one distinct node with outdegree 0, called the *root* (or *sink*) of G . See Fig. 2 for example.

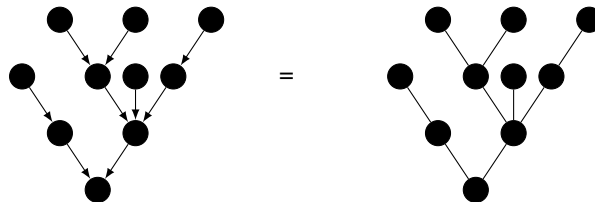


Fig. 2: An intree with 10 tasks.

Definition 7 (Inforest). A graph whose connected components are intrees is called an *inforest*.

An inforest may have several nodes with outdegree 0 and might no longer be connected, see Fig. 3. Connecting all roots/sinks of an inforest to a common supersink results in an intree again. This supersink is depicted by the white node in Fig. 3.

In the following, we write about a *task* whenever we mean the scheduling concept, and write about a *node* whenever we mean the graph theoretical object. As each task is represented by a node in the figures, sometimes these two notions can be used interchangeably.

Notice that in Fig. 2 and Fig. 3 we do not give names to the tasks like we did for the tasks in Fig. 1. Most of the times, we do not care about them anyway because the optimal schedule is

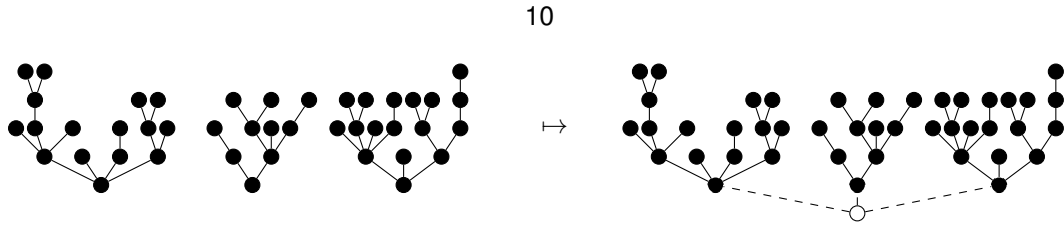


Fig. 3: An inforest and the corresponding intree with a supersink.

not affected by the naming scheme of the tasks. This is why we name the tasks in our examples in a way that we deem easiest or most appropriate. This means that isomorphic (with respect to naming the tasks) precedence graphs are considered identical. Actually, the names only become important when a final schedule is calculated and we want to visualize it in form of a Gantt chart or something similar, cp. Fig. 1.

3 Optimality Criteria

The last thing to be defined to characterize a scheduling problem is the optimality criterion that is used. This is done by an objective function that, most of the times, depends on the finishing times of the tasks. There is a whole variety of objective functions, some of which are explained below. We will focus on the very common *makespan*.

Denote the finishing time (or completion time) of task i by C_i , i.e., the time task i is fully processed on any processor. The two most common types of objective functions are either

$$\max\{f_i(C_i) : i = 1, \dots, n\}$$

or

$$\sum_{i=1}^n f_i(C_i)$$

where the f_i 's are special functions. Most of the times, the index is dropped from the notation as it is often obvious that i ranges over all tasks.

Definition 8 (Makespan). The *makespan*, denoted by C_{\max} , is the time at which the last task finishes, i.e.,

$$C_{\max} = \max\{C_i : i = 1, \dots, n\}.$$

Given a precedence relation by a graph G , we also call the makespan the *processing time of G* .

The optimality criterion which we use throughout this work is to minimize the makespan, i.e., to minimize the time at which the last task finishes. In other words, we want to find a scheduling strategy that results in the shortest (in terms of time) schedule that processes all tasks.

Other common objective functions (taken from [8]) are mentioned below, but will not be discussed any further in this work.

Consider a scenario with given deadlines (or due dates) d_i for each task i . Then the following values can be defined

$$\begin{array}{ll}
 L_i = C_i - d_i & \text{lateness,} \\
 E_i = \max\{0, d_i - C_i\} & \text{earliness,} \\
 T_i = \max\{0, C_i - d_i\} & \text{tardiness,} \\
 D_i = |C_i - d_i| & \text{absolute deviation,} \\
 S_i = (C_i - d_i)^2 & \text{squared deviation,} \\
 U_i = \begin{cases} 0 & C_i \leq d_i \\ 1 & \text{otherwise} \end{cases} & \text{unit penalty.}
 \end{array}$$

Many different objective functions can be generated from these values, e.g. the sum of the finishing times $\sum C_i$ or the *maximum lateness* $L_{\max} = \max\{L_i : i = 1, \dots, n\}$. Other common objective functions are $\sum T_i$, $\sum w_i T_i$ (with weights w_i), $\sum U_i$, $\sum w_i U_i$, and other combinations of the values above.

Definition 9 (Optimality). Given a scheduling problem, a scheduling strategy is *optimal* if, for any given set of tasks, the resulting schedule optimizes the objective function. Here, it means that there is no other scheduling strategy that results in a smaller makespan, i.e., a shorter schedule.

As we consider stochastic scheduling, and have to deal with probabilistic measures, the definition of our objective has to be changed a little bit.

Definition 10 (Expected Makespan). The *expected makespan*, denoted by $\mathbb{E}(C_{\max})$, is the expected time at which the last task finishes, i.e.,

$$\mathbb{E}(C_{\max}) = \mathbb{E}(\max\{C_i : i = 1, \dots, n\}),$$

where the expected value is taken over all possibilities that can occur according to the given distributions and scheduling strategy.

This is also called the *expected processing time of G* for a given precedence relation G .

In the same way, the definition of optimality must be adjusted for our purposes.

Definition 11 (Expected Makespan Optimality). Given a scheduling problem, a scheduling strategy is *optimal* if, for any given set of tasks, the resulting schedule minimizes the expected makespan.

Here, this means that there is no other scheduling strategy that results in smaller value for the expected makespan. Note that there may now be many different schedules as a result from only one scheduling strategy because of the stochastic nature of the problem. For two different scheduling strategies for the same problem, it may even be the case that one of those strategies is optimal, although the other one produces a shorter schedule in some specific instances. Of

course, the issue with a stochastic setting is that we can never be sure at which instances of a problem we are looking at, and what the processing times are. This is the reason why we have to optimize the expected values taken over the probability space of all possible resulting schedules.

4 Notation

A short notation to characterize a scheduling problem was introduced in [30]. The format is $\alpha|\beta|\gamma$ with the following.

- α : The *machine environment*. In our case, this is typically just two or three parallel identical (P) machines, but there are a lot of possibilities to characterize this, e.g. uniform machines (Q), unrelated machines (R), job shop (J), flow shop (F), and many more. Because we will always focus on identical machines, we will drop the letter P in the notation and only denote the number of processors.
- β : The *task characteristics* as described in Chapter 2. Here we describe up to six different properties of the tasks, separated by a semicolon:
 - If *preemption* is allowed, then we write *pmtn*.
 - If there are *precedence relations*, then we write e.g. *prec* (for general directed acyclic graphs). More specific cases like *intree*, *chains*, or *sp-graph* (for series-parallel graphs) are possible as well.
 - If *release dates* are specified, then we write r_i . A release date r_i denotes that task i is not available before time r_i , even if it has no predecessors.
 - The *processing requirements* are described, like $p_i = 1$ (unit processing requirements) or $p_i \in \{1, 2\}$, or stochastic processing requirements like e.g. $p_i \sim \exp(\lambda)$ (unless specified otherwise, the mentioned distributions are always independent and identical).
 - If *deadlines* are specified, then we write d_i .
 - If *batching* is required, then we write *p-batch* or *s-batch*. Batching describes grouping the tasks into several blocks/batches and then schedule these batches.
- γ : The *optimality criterion* as described above. Here we write e.g. L_{\max} , $\sum T_i$, or some stochastic criterion like $\mathbb{E}(C_{\max})$.

Many of the characterizations above will not show up in this work, but it should be clear that the tiniest change in one of those characteristics can make way for a whole new family of scheduling problems. We will only consider a glimpse of some of those problems, see [8] for a more elaborate description and some more examples.

The above notation has been widely accepted as standard notation and is also used in this work.

5 Reductions between Scheduling Problems

Because of the many different characteristics we defined above, there are hundreds of different scheduling problems. Some of those can be reduced to others, meaning some are just special cases of others. For example, consider a scheduling problem with unrelated parallel machines. Then any scheduling strategy for this problem will also work for identical machines, but not necessarily vice versa. An example which is more suited to the scheduling problems we deal with here in this work is that any strategy for a scheduling problem with intree precedence constraints will also work if the precedence constraints are just chains. Of course, this is because a graph of chains is always an inforest, and we have seen in Fig. 3 that an inforest can be reduced to an intree. These kinds of reductions do not necessarily show us the easiest scheduling strategy as that may simplify when the considered scheduling problem is simplified as well. But if the considered strategy is optimal for intrees for example, then it surely is optimal for a graph of chains, too.

6 The Highest-Level-First Scheduling Strategy

Definition 12 (Level). Given an intree $G = (V, E)$ and $v \in V$, define $\text{level}(v)$ as the number of edges on a path from v to the root of G , called the *level* of v .

Let L denote the maximum level of any node, i.e., the length of a longest path going to the root. Whenever it is obvious from the context which tasks are at level 0, then we can use the concept of a level even for inforests or *dags*. Usually, the level-0-tasks are the ones which have no successors.

Throughout a great part of this work, we will focus on a very famous scheduling strategy, namely the highest-level-first scheduling strategy.

Definition 13 (HLF). The *highest-level-first* (HLF) scheduling strategy assigns to idle processors the highest possible available tasks.

In general, there may exist several HLF scheduling strategies. If there are more available tasks on the highest level than we have idle processors, even an HLF scheduling strategy must make a choice. Thus, there are two different types of optimality regarding HLF scheduling strategies:

1. There exists an HLF scheduling strategy that is optimal for the given scheduling problem.
2. All HLF scheduling strategies are optimal for the given scheduling problem.

The first type is shown by proving that one specific HLF scheduling strategy is optimal. The second type can be proven in two ways:

- The claims and results are proven given arbitrary HLF scheduling strategies.
- The claims and results are proven given one specific HLF scheduling strategy, then the different strategies are proven to be equivalent.

In examples, we will use the level-oriented HLF scheduling strategy as defined in the following definition. In the theory section, we will prove the results using the first way and consider arbitrary HLF scheduling strategies. The first distinction between “existential optimality” and “universal optimality” is not particularly interesting in this chapter, but we will see more remarks about this in Part X.

Definition 14 (Level-oriented Topological Sorting). Given an intree G , we define a *level-oriented topological sorting* as a bijection from the set of tasks \mathcal{T} to $\{1, \dots, n\}$ with the following:

The root is assigned 1, its k predecessors are assigned the numbers $2, 3, \dots, k + 1$, their predecessors are assigned the next available numbers, and so on, assigning numbers to a task at level i only if all tasks at level $i - 1$ have already been assigned numbers.

Whenever we draw an intree or a precedence relation, respectively, we position the tasks with respect to the level-oriented topological sorting, i.e. within a level the numbering forms an increasing sequence when going from left to right. Since we interpret the level-oriented topological sorting with priorities for the tasks, for an HLF scheduling strategy this means that tasks on the same level are chosen to be processed from left to right if available.

Whenever we want to refer to the specific HLF scheduling strategy that breaks ties between equally high tasks by using the level-oriented topological sorting, we call this scheduling strategy “level-oriented *HLF*”.

In the following, we shorten the notation and use “*HLF*” (written in italics) as a placeholder for “HLF scheduling strategy” in general.

7 Configuration

Earlier, in Definition 2, we defined the concept of a decision point. Like the name suggests, a decision point is a point in time in the schedule when the scheduling strategy has to decide what to do next, i.e., which tasks to process on which processors, if any. And since we do not allow preemption in our schedules, a scheduling strategy cannot make any decisions between two decision points. This means that for the scheduling strategy, the state of the schedule during those intervals is not relevant, its choices only depend on the state of the schedule at the current decision point. To this end, we have the following definition to incorporate all relevant information at a decision point.

Definition 15 (Configuration). Let G be a precedence relation. A *configuration* of a scheduling strategy at a given decision point is defined as the tuple (G', L) , where $G' \subseteq G$ describes the remaining tasks, and $L \subseteq \mathcal{T} \times \mathbb{R}$ is a list of currently *active* tasks along with their respective already processed times, with $|L| \leq m - 1$ where m denotes the number of processors used by the scheduling strategy.

We observe that not all possible combinations of subgraphs G' of G and task lists L are configurations that can occur. G' has to be an appropriate subgraph of G , and in the case where G

is an intree, G' has to be an intree as well. In the following, we will use $m = 2$ processors, but the definition of a configuration allows us to make that distinction. We could also allow the case where all processors are not identical, then the list of active tasks L has to model a mapping to the processors as well. In this work, only identical processors are considered.

Usually, we depict configurations by drawing the underlying graph and marking the active tasks instead of giving the tuple as defined in Definition 15. In Fig. 4 we can see three examples of a configuration, all based on the same graph G . The configuration on the left has no active tasks, so the graph itself represents the configuration (G, \emptyset) – or just G . The middle configuration is $(G, (x_1 : i))$, depicted by the graph G with one marked (crossed) task, which has been processed for i time units so far. The configuration on the right has two active tasks, one processed for j , the other for k time units, respectively. The corresponding formal notation is $(G, (x_1 : j, x_4 : k))$. Observe that this is not an HLF configuration for an intree such as G , as the task x_4 was scheduled (and is being processed) before x_3 .

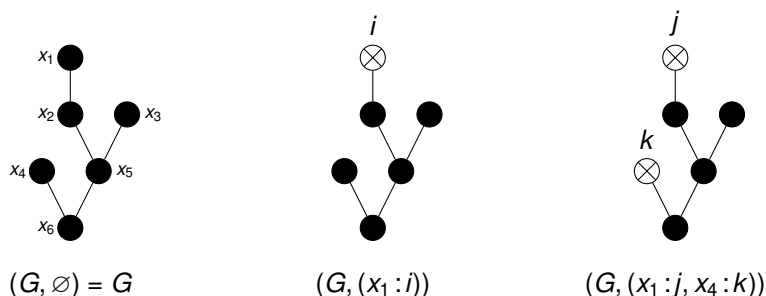


Fig. 4: Examples of configurations.

Definition 16. Let G be an intree, c a configuration of G , and S a scheduling strategy. Then $C_c^S(c)$ denotes the expected makespan of a schedule according to S of the configuration c . If S is any *HLF*, then we drop the superscript and just write $C_c(c)$.

We sometimes also write $C_c(G)$ for the configuration that represents the intree G without any active tasks. Alternatively, we sometimes provide G , or c , respectively, as a drawn image.

8 Profile

Definition 17 (Profile). Given a graph $G = (V, E)$ with well-defined levels, define the *profile* of G as the sequence $\text{profile}(G) = (p_1, p_2, \dots, p_L)$, with p_i the number of tasks on level i , $1 \leq i \leq L$.

Let L' denote the highest level with $p_{L'} > 1$, if available. Otherwise, i.e., if G is a chain, then L' is not defined. Note that if there is more than one task on the highest level L , then $L' = L$. Fig. 5 shows two different intrees, with profiles $(1, 2, 4, 3)$ and $(1, 1, 4, 3, 1)$, respectively.

Now, we have defined enough concepts to have a first small result.

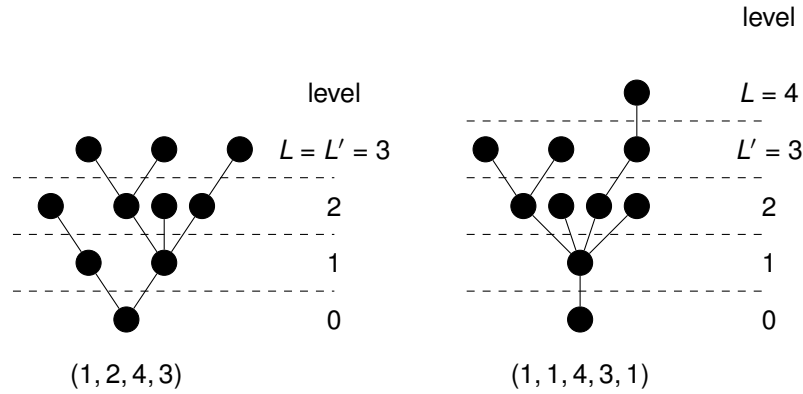


Fig. 5: Intrees with different profiles.

Lemma 18. *Let G be an intree with profile (p_1, \dots, p_L) . If $p_L \geq 2$, then the two highest sources are at level L . Otherwise, the highest source is at level L and the second highest source is at level L' .*

Proof. Tasks at level L are all sources, so if $p_L \geq 2$, the claim is obvious.

If $p_L = 1$, start searching for the second highest source at the level below. We know that one task at this level has a predecessor in level L , thus is not a source. So, if $p_{L-1} = 1$, continue searching at the level below, and so on. At the first level L' we encounter with $p_{L'} > 1$, we have one task with a predecessor at level $L' + 1$. Since G is an intree, all encountered tasks so far have exactly one successor each. Hence, at least one other task at level L' is without any predecessor, i.e. a source. \square

We define the notation $(p_1, \dots, p_L, 0, \dots, 0)$ to be equal to (p_1, \dots, p_L) , i.e., empty levels can always be added to the top of the intree. Then the highest level is defined as the highest level with at least one task on it. This way we can have the notation $(p_1, \dots, p_L - 1)$ even though level L might now be empty, i.e., $p_L - 1 = 0$.

Definition 19. Let G be an intree with profile p , and S a scheduling strategy. Then, $C_p^S(p)$ denotes the expected makespan of a schedule according to S of any configuration with profile p .

If S is any *HLF*, then we write $C_p(p)$.

We will show in Chapter 17 that this definition is actually well-defined and makes sense.



Mathematical Concepts

Table of Contents

9 Sums and Series	19
10 Power Series and Generating Functions	21
11 Hypergeometric Summation	23
12 Integrals	24
13 Probability Theory	24

In this subsection, we will define a few mathematical tools that we need for our analyses in the subsections afterwards. Most of the definitions and explanations come from [29, 17].

9 Sums and Series

We adopt the usual notation for sums, namely

$$a_1 + a_2 + \cdots + a_n \quad (9.1)$$

with the ellipsis mark (the three dots . . .), or, most of the times,

$$\sum_{i=1}^n a_i \quad (9.2)$$

with appropriate lower and upper (inclusive) limits, 1 and n here. This notation was introduced by Joseph Fourier in 1820, e.g. reprinted in [11], and has been used ever since. In general, the limits can be replaced by any conditions that describe the set of indices used for the summation. We have already seen this notation in Chapter 3 on page 10, but it should have been known to the reader even before that anyway.

A *series* describes an infinite sum, i.e., a sum like

$$\sum_{i=1}^{\infty} a_i = \sum_{i \geq 1} a_i,$$

where there is no upper limit for the summation index i .

Although a series is the sum over infinite terms, it may very well have a finite value. In fact, there are a lot of those *converging* series, e.g. the *geometric series*:

$$\sum_{i \geq 0} x^i = \frac{1}{1-x} \quad (9.3)$$

for any real number x with $|x| < 1$ (it should be obvious that this series does not converge if $|x| \geq 1$). There is also an analogous result about finite *geometric sums*, i.e.,

$$\sum_{i=0}^n x^i = \frac{1-x^{n+1}}{1-x}, \quad (9.4)$$

which holds for any $x \in \mathbb{R} \setminus \{1\}$. For more details, we again refer to [29].

We also use multi-dimensional sums and series, which use more than one summation index. To this end, let $(a_{ij} : i, j \in \mathbb{N})$ be a two-dimensional sequence with the indices i and j both ranging over all natural numbers \mathbb{N} . Then we either write a two-dimensional sum following the notation

of (9.1) as

$$\begin{aligned} & a_{11} + a_{12} + \cdots + a_{1m} \\ & + a_{21} + a_{22} + \cdots + a_{2m} \\ & \quad \vdots \\ & + a_{n1} + a_{n2} + \cdots + a_{nm} \end{aligned}$$

or following the notation of (9.2) in a shorter way as

$$\sum_{i=1}^n \left(\sum_{j=1}^m a_{ij} \right),$$

and in general, the parentheses are not used. Similarly, we can have the series

$$\sum_{i \geq 1} \sum_{j \geq 1} a_{ij} = \sum_{i,j \geq 1} a_{ij}, \quad (9.5)$$

ranging over all natural numbers again. However, the notation on the right in (9.5) is not preferred as it can easily be overlooked that this describes a two-dimensional series, because it only uses one summation sign. Similarly, we may use sums or series with higher dimensions than two.

The more interesting thing about more-dimensional sums and series is that sometimes we can change the order of summation (in particular, this can be done if the sums or series converge). For example, because addition is commutative, the following identity holds:

$$\sum_{i \geq 1} \sum_{j \geq 1} a_{ij} = \sum_{j \geq 1} \sum_{i \geq 1} a_{ij}.$$

We can make use of this property whenever summing over one index first would yield an easier intermediate results than when summing over the other first.

Binomial Coefficients

In all this work, combinatorics is omnipresent. It deals with counting certain objects or elements. A widely-used example in this work is the question “*How many possible pairs of available tasks can we choose?*”, or “*How many possible ways can a schedule go along to arrive at a certain configuration?*”. For both of these examples we use *binomial coefficients*, that count the number of k -element subsets from an n -element set.

Lemma 20 (Binomial Coefficients). *The binomial coefficient $\binom{n}{k}$, read “ n choose k ”, denotes the number of k -element subsets of an n -element set, and is equal to*

$$\frac{n(n-1) \cdots (n-k+1)}{k!} = \frac{n!}{k!(n-k)!},$$

where $n! = n \cdot (n-1) \cdots 2 \cdot 1$ denotes the factorial of n , i.e., the product of all natural numbers up to (and including) n .

A proof for Lemma 20 can be found in [15].

There are many different identities and inequalities using binomial coefficients, for example in [28, 27]. One we will use more often than others is

$$\binom{n}{k} = \binom{n}{n-k}.$$

Considering the interpretation of binomial coefficients as choosing k out of n elements, this should be obvious, as there is the same number of possibilities for choosing the remaining $n-k$ elements instead of choosing the k elements themselves. Another identity used in this work is the following.

Theorem 21 (Binomial Theorem). *Let $a, b \in \mathbb{R}$ and $n \in \mathbb{N}_0$. Then*

$$(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}.$$

The name “*Binomial Theorem*” refers to different versions of this identity (sometimes using real numbers n and so on) depending on the source material. Our version is the one seen in [3].¹ The proof is left as an easy exercise.

10 Power Series and Generating Functions

A *power series* in one (auxiliary) variable is a series of the form

$$\sum_{i \geq 0} a_i z^i \tag{10.1}$$

for a sequence $(a_i : i \geq 0)$ and a complex variable z . It can be seen as a generalization of a polynomial to infinite degree. We have already seen a power series before, the geometric series in (9.3) is one with $a_i = 1$ for all $i \geq 0$. A *generating function* for a given sequence $(a_i : i \geq 0)$ is exactly that power series in (10.1), i.e., the generating function of $(a_i : i \geq 0)$ is

$$A(z) = \sum_{i \geq 0} a_i z^i.$$

Again, there are countless examples of converging power series and generating functions that have a closed form. We want to give two short examples. First, consider the geometric series from (9.3) again, then we have that the generating function for the sequence $(1, 1, 1, \dots)$ is $A(z) = \frac{1}{1-z}$ for $|z| < 1$. As a second example we have seen in Theorem 21 and in its footnote, that $(1+z)^n$ is the generating function of the sequence $(\binom{n}{i} : i \geq 0)$. The most interesting thing

¹One can also show that $(a+b)^n = \sum_{k \geq 0} \binom{n}{k} a^k b^{n-k}$ (with an infinite summation) if the binomial coefficients are properly defined to be 0 for $k > n$. But the interpretation tells us that these should be equal to 0 anyway, so this identity holds as well.

about these examples is that some infinite sequences can be represented entirely by short formulas, i.e., their behaviors are encoded in the closed forms.

As with sums and series in the subsections before, we can generalize this concept to higher dimensions as well. For example, the generating function of a two-dimensional series ($a_{ij} : i, j \geq 0$) is given by

$$A(x, y) = \sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j,$$

which is now a function in two variables x and y .

A short introduction as well as many detailed tools for dealing with generating functions can be found in [54].

Convolution

Now, consider two sequences ($a_i : i \geq 0$) and ($b_i : i \geq 0$) with corresponding generating functions $A(z)$ and $B(z)$, respectively. Then the product of these two generating functions is

$$A(z) \cdot B(z) = \left(\sum_{i \geq 0} a_i z^i \right) \cdot \left(\sum_{i \geq 0} b_i z^i \right),$$

which itself is a power series, as seen in the following lemma.

Lemma 22 (Convolution). *Let ($a_i : i \geq 0$), ($b_i : i \geq 0$) be sequences with corresponding power series $\sum_{i \geq 0} a_i z^i$ and $\sum_{i \geq 0} b_i z^i$, respectively. Then,*

$$\left(\sum_{i \geq 0} a_i z^i \right) \cdot \left(\sum_{i \geq 0} b_i z^i \right) = \sum_{i \geq 0} \left(\sum_{j=0}^i a_j b_{i-j} \right) z^i.$$

This lemma shows that the product of two power series is again a power series with coefficients

$$c_i = \sum_{j=0}^i a_j b_{i-j}.$$

The sequence ($c_i : i \geq 0$) is called *convolution* of the sequences (a_i) and (b_i).

Generalizing this to two dimensions, we get the following.

Lemma 23 (Multivariate Convolution). *Let ($a_{ij} : i, j \geq 0$) and ($b_{ij} : ij \geq 0$) be sequences with corresponding generating functions $\sum_{i \geq 0} \sum_{j \geq 0} f_{ij} x^i y^j$ and $\sum_{i \geq 0} \sum_{j \geq 0} g_{ij} x^i y^j$, respectively. Then,*

$$\left(\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j \right) \cdot \left(\sum_{i \geq 0} \sum_{j \geq 0} b_{ij} x^i y^j \right) = \sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{k=0}^i \sum_{\ell=0}^j a_{k\ell} b_{i-k, j-\ell} \right) x^i y^j.$$

In particular, the product of a two-dimensional power series $\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j$ with a one-dimensional power series $\sum_{i \geq 0} b_i x^i$ can be calculated with

$$\left(\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j \right) \cdot \left(\sum_{i \geq 0} b_i x^i \right) = \sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{k=0}^i a_{kj} b_{i-k} \right) x^i y^j.$$

The proofs for Lemma 22 and Lemma 23 are easy and left out here.

11 Hypergeometric Summation

A good introduction (and much more) to this topic can be found in [46].

We have seen an example of geometric summation in (9.3) in Chapter 9. To put it in a more formal way, a geometric series has the property that the ratio of every two consecutive terms is constant, i.e., for a geometric series $\sum_{i \geq 0} a_i$ it must hold that a_{i+1}/a_i is a constant function of i . Consider the example in (9.3), then it is easy to see that $a_{i+1}/a_i = x$, which is constant in i . This means that all geometric series are of the form

$$\sum_{i \geq 0} cx^i.$$

A *hypergeometric series* does not have to have this property. However, the ratio of two consecutive terms must follow some constraints, namely that, for a series $\sum_{i \geq 0} a_i$,

$$a_0 = 1 \quad \text{and} \quad \frac{a_{i+1}}{a_i} = \frac{P(i)}{Q(i)}$$

for two polynomial functions P and Q . This means that the ratio between two consecutive terms is a rational function of the summation index i .

Now consider that P and Q are completely factored, which yields

$$\frac{a_{i+1}}{a_i} = \frac{P(i)}{Q(i)} = \frac{(i + a_1)(i + a_2) \dots (i + a_p)}{(i + b_1)(i + b_2) \dots (i + b_q)(i + 1)} \cdot x$$

with some constant x . Then, the notation for the hypergeometric series (or *hypergeometric function*) $\sum_{i \geq 0} a_i x^i$ is

$${}_pF_q \left[\begin{matrix} a_1 a_2 \dots a_p \\ b_1 b_2 \dots b_q \end{matrix}; x \right].$$

Maybe the best-known example of a hypergeometric series is the exponential series

$$\exp(x) = e^x = \sum_{i \geq 0} \frac{x^i}{i!}.$$

Its ratio of consecutive terms is $x/(i + 1)$, which gives

$$e^x = {}_0F_0 \left[\begin{matrix} - \\ - \end{matrix}; x \right].$$

Identifying a series as a hypergeometric series is not trivial at all, and for most examples that follow in this work, we used computer software to do that job for us, mostly **Mathematica**¹ and **Maple**².

12 Integrals

Isaac Newton and Gottfried Leibniz independently invented the principle of integrals in the late 17th century, before Bernhard Riemann properly defined it in the 19th century. Intuitively, integrals can be seen as a generalization of summations for an uncountable number of elements. At least we use integrals in the same places as sums and series whenever the need arises. We adopt the usual notation

$$\int_a^b f(x) dx$$

for an integral ranging over the interval $[a, b]$ of the function f , and use the integration variable x here. The choice of the integration variable depends on the context and on whichever variable is already in use. In general we may also write

$$\int_A f(x) dx$$

for an appropriate region of integration A , which will be an interval most of the times.

The *fundamental theorem of calculus* [25] (or a corollary thereof, respectively) gives us a tool to calculate an integral by

$$\int_a^b f(x) dx = [F(x)]_a^b = F(b) - F(a), \quad (12.1)$$

where F is the *antiderivative* of the function f , i.e., $\frac{d}{dx}F = f$, whenever F exists.

13 Probability Theory

The following concepts can be found (among others) in [17, 23].

Probability theory is an essential tool for this work as we want to model the tasks' processing times with randomly distributed values and not deterministic, fixed ones.

¹Version 10.1.0 for Linux x86 (64-bit)

²Version 2015 (X86 64 Linux)

First, we must have the *sample space*, a (possibly infinite) set of possible results of the experiment or observation that we want to focus on. Each of those possible results is called *elementary (or simple) event*. One of our running examples throughout this subsection is tossing one or multiple coins. For this experiment we have the sample space

$$\Omega = \{H, T\},$$

where H and T stand for head and tail, respectively, the two outcomes of a coin toss. Similarly, for tossing two coins, we get $\Omega = \{HH, HT, TH, TT\}$ (whenever the two coins are distinguishable). An *event* is a subset of the sample space, here for example, the event of tossing at least one head $E = \{HH, HT, TH\}$. And because events are sets, they can be treated with the same algebraic principles as all other sets, i.e., we can form the union of two events, intersections, complements, differences, and so on.

We denote the probability of an event E by

$$\mathcal{P}(E)$$

and define the probability measure \mathcal{P} for all (elementary) events in Ω such that the following axioms are fulfilled:

1. $\mathcal{P}(E) \geq 0$ for all events $E \subseteq \Omega$.
2. $\mathcal{P}(\Omega) = 1$.
3. $\mathcal{P}(E \cup F) = \mathcal{P}(E) + \mathcal{P}(F)$ for any two mutually exclusive events $E, F \subseteq \Omega$, i.e., $E \cap F = \emptyset$.

A *probability distribution* is a way to define the probability measure \mathcal{P} .

The probability that any (or both) of the two events E and F occur is given by

$$\mathcal{P}(E \cup F) = \mathcal{P}(E) + \mathcal{P}(F) - \mathcal{P}(E, F),$$

where $\mathcal{P}(E, F)$ denotes the probability of the two events E and F occurring simultaneously. If $\mathcal{P}(E, F) = 0$, then E and F are *mutually exclusive* or *independent* and

$$\mathcal{P}(E \cup F) = \mathcal{P}(E) + \mathcal{P}(F).$$

A meaningful¹ definition of the probability measure for the example with two coin tosses would be

$$\mathcal{P}(HH) = \mathcal{P}(HT) = \mathcal{P}(TH) = \mathcal{P}(TT) = \frac{1}{4},$$

describing that all possible outcomes are equally likely to occur. Now, the two events “head on the first toss” $\{HH, HT\}$ and “tail on the first toss” $\{TH, TT\}$ are independent, whereas the events “at least one head” $\{HH, HT, TH\}$ and “at least one tail” $\{HT, TH, TT\}$ are not, because they overlap when the outcome is one of HT or TH .

¹when using a fair coin

A *discrete* probability distribution is defined over a finite or countably finite sample space Ω . Then, for every event E , we have

$$\mathcal{P}(E) = \sum_{x \in E} \mathcal{P}(x),$$

where the x 's are elementary events. For example, for the two coin tosses, we get that

$$\mathcal{P}(\text{at least one head}) = \mathcal{P}(HH) + \mathcal{P}(HT) + \mathcal{P}(TH) = \frac{3}{4}.$$

In general, the axioms above yield that the sum of the probabilities of all elementary events is always 1, i.e.,

$$\sum_{x \in \Omega} \mathcal{P}(x) = 1, \quad (13.1)$$

because $\bigcup_{x \in \Omega} x = \Omega$ and because elementary events are defined to be pairwise mutually exclusive. Of course, this makes sense since the union of all elementary events amounts to all possible outcomes and no other possibility is left.

A *continuous* probability distribution is defined over a real interval $\Omega = [a, b]$ with $a < b$. Since there is now an uncountable number of elementary events, almost all of those have a probability of 0 so as to satisfy axioms 2 and 3. Of course, (13.1) would not work in this case, but still, we can satisfy all axioms if we consider integration instead of summation (which would not be properly defined for an uncountable set anyway). To this end, we define

$$\mathcal{P}(E) = \int_E pdf(x) dx,$$

where $E \subseteq [a, b]$ and *pdf* is the *probability density function* defining the probability distribution. Similar to (13.1), we get

$$\int_{\Omega} pdf(x) dx = \int_a^b pdf(x) dx = 1. \quad (13.2)$$

The *conditional probability* of an event E conditioned that another event F occurs is defined by

$$\mathcal{P}(E | F) = \frac{\mathcal{P}(E \cap F)}{\mathcal{P}(F)},$$

read “the probability of E given F ”, whenever $\mathcal{P}(F) \neq 0$. In particular, for independent events E, F , we have

$$\mathcal{P}(E | F) = \mathcal{P}(E),$$

because E and F being independent means $\mathcal{P}(E \cap F) = \mathcal{P}(E)\mathcal{P}(F)$.

A *random variable* $X : \Omega \rightarrow \mathbb{R}$ is a function from a sample space to the real numbers. Its purpose is to assign values to the outcomes of the experiment, and the probabilities of the different outcomes are given by the underlying probability distribution on Ω . Depending on the used distribution, a random variable can be *discrete* or *continuous*. We denote this by $X \sim \mathcal{D}$ where \mathcal{D} denotes the used distribution, see Part V for the first example. Usually, a continuous random

variable X is defined by giving the probability density function f_X (or *pdf* $_X$). The *support* of a random variable is the set $S \subseteq \Omega$ with non-zero values for $x \in S$, i.e., $S = \{x \in \Omega : f_X(x) \neq 0\}$. The *cumulative distribution function* $F_X(x) = \mathcal{P}(X \leq x)$ describes the probability that the random variable X takes a value less than or equal to x . In the case that X is discrete, we have

$$F_X(x) = \mathcal{P}(X \leq x) = \sum_{k \leq x} \mathcal{P}(X = k),$$

whereas, in the case that X is continuous, we have

$$F_X(x) = \int_{-\infty}^x f_X(z) dz.$$

From this identity, we see that the choice of small and capital letters f and F , respectively, is not arbitrary, it actually conforms to the notation in (12.1) as the cumulative distribution function is the antiderivative of the probability density function.

The *expected value* $\mathbb{E}(X)$ of a random variable X is defined by

$$\mathbb{E}(X) = \begin{cases} \sum_{x \in \Omega} x \cdot \mathcal{P}(x) & X \text{ is discrete} \\ \int_{\Omega} x \cdot f_X(x) dx & X \text{ is continuous.} \end{cases}$$

In some special cases, there are other ways to calculate the expected value, see the following lemma.

Lemma 24. *Let X be a nonnegative random variable, i.e., $X \geq 0$, then the expected value can be calculated by*

$$\mathbb{E}(X) = \int_0^{\infty} 1 - F_X(x) dx.$$

Proof. By partial integration [25] we have

$$\begin{aligned} \mathbb{E}(X) &= \int_0^{\infty} x \cdot f_X(x) dx \\ &= [-x(1 - F_X(x))]_0^{\infty} - \int_0^{\infty} -(1 - F_X(x)) dx \\ &= \int_0^{\infty} 1 - F_X(x) dx. \end{aligned}$$

□



The Expected Makespan

Table of Contents

14 Discrete Case	31
15 Continuous Case	32

Using the general nature of the definition of a configuration, we can propose a formula for calculating the expected makespan for a whole variety of scheduling problems. For a brief moment, we consider a very general scheduling problem with m processors, and assume that the current configuration is c , then the idea is that we are waiting until the next decision point, i.e., the next point in time when a task is finished, and then look at the possible configurations from then on:

$$C_c(c) = \text{expected time until next decision point} + \sum_{\text{possibility } p} \mathcal{P}(p) \cdot \text{expected remaining time},$$

where different possibilities describe different successor configurations.

We introduce the notation $X \wedge Y = \min\{X, Y\}$ for the minimum of two random variables. And because the minimum operator is commutative and associative, we also introduce $X_1 \wedge \dots \wedge X_n = \min\{X_1, \dots, X_n\}$ as the minimum of n random variables.

14 Discrete Case

The interpretation above put in a more formal expression then looks like

$$C_c(c) = \mathbb{E}(X_1 \wedge \dots \wedge X_k) + \sum_{i=1}^k \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i) \cdot C_c(c_i), \quad (14.1)$$

where the X_i 's denote the discrete distributions of the processing times of the $k \leq m$ processed tasks (we note that we still implicitly assume a certain scheduling strategy, otherwise finding these processed tasks is a non-trivial task itself), and c_i is the configuration that follows if the task with processing time distributed with respect to X_i is finished first. For the case where there is only one successor configuration (or none), the formula becomes much simpler, since the sum is not needed anymore.

Knowing the underlying scheduling strategy (and, of course, the structure of the precedence constraints) we can easily (not necessarily in the sense of efficiency) determine all possible configurations that might occur in an actual schedule, save for the already passed processing times of the active tasks.

However, there is an alternative way to interpret the expected makespan. Instead of considering the expected time until the next decision point, i.e., the expected value of the minimum, we can have a weighted sum of expected makespans corresponding to all possible successor configurations (as we have above) added to the passed time steps until that next configuration. Mathematically this means that

$$C_c(c) = \sum_{\text{possibility } p} \mathcal{P}(p) \cdot (\text{passed time} + \text{expected remaining time}).$$

Again, plugging in the actual definitions and values yields

$$C_c(c) = \sum_{i=1}^k \sum_{j=1}^{r_i} \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i) \mathcal{P}(a_j) \cdot (a_j + C_c(c_i)). \quad (14.2)$$

where the a_j 's are the r_j different values that the random variable X_j can take. Note that the configuration c_j may depend on the value of j . (14.1) can be obtained from (14.2) by considering $\sum \sum \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_j) \mathcal{P}(a_j) a_j$ as a separate sum whose value can be determined. The proof of the equivalence for different distributions can be seen in the respective parts, for example in Part V in (16.3) (page 37) for the exponential distribution.

15 Continuous Case

In the case where the processing times are continuously distributed, we need to introduce additional integrals, one for each possible successor configuration, and the formula looks like the following:

$$C_c(c) = \mathbb{E}(X_1 \wedge \dots \wedge X_k) + \sum_{i=1}^k \int_{I_i} \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i) pdf_{X_1 \wedge \dots \wedge X_k}(x) \cdot C_c(c_i) dx, \quad (15.1)$$

where an integral is over the interval I_i of all possible values for X_i , and $pdf_{X_1 \wedge \dots \wedge X_k}$ denotes the probability density function of the minimum $X_1 \wedge \dots \wedge X_k$. Observe, that in some cases, both $\mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i)$ and $C_c(c_i)$ may be functions of x , i.e., then they are not constant factors within the integral, but non-trivial functions for the integration. We refrain from using the notations $\mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i)(x)$ and $C_c(c_i, x)$ for the sake of readability.

There are even more details hidden in the notations, as the configurations incorporate active tasks and their already passed processing times, which means that c_i may depend on the X_j 's and $X_1 \wedge \dots \wedge X_k$. Also, the expected value, the probability, and the probability density function may actually be a conditional expected value, a conditional probability, or a function conditioned on some previous results as well, respectively. To this end, we define E as the event that $X_{j_1} \geq i_1, \dots, X_{j_\ell} \geq i_\ell$ for some indices $\{i_1, \dots, i_\ell\} \subseteq \{1, \dots, k\}$ and some appropriate values i_1, \dots, i_ℓ . Then this results in

$$C_c(c) = \mathbb{E}(X_1 \wedge \dots \wedge X_k | E) + \sum_{i=1}^k \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i | E) \int_{I_i} pdf_{X_1 \wedge \dots \wedge X_k | E}(x) \cdot C_c(c_i) dx,$$

meaning that we have to calculate the expected value and the probability under the condition that some tasks $X_{i_1}, \dots, X_{i_\ell}$ have already been processed for i_1, \dots, i_ℓ time units, respectively. The same holds for the probability density function, which can be conditioned on this event as well.

Similar to the discrete case, we have a formula for the other interpretation as well, i.e., an analogon to (14.2) for the continuous case:

$$C_c(c) = \sum_{i=1}^k \int_{I_i} \mathcal{P}(X_1 \wedge \dots \wedge X_k = X_i) pdf_{X_1 \wedge \dots \wedge X_k}(x) (x + C_c(c_i)) dx. \quad (15.2)$$

disregarding the notation with the conditional probability for easier readability.



The Exponential Distribution

Table of Contents

16 Calculating the Expected Makespan	36
17 The Optimal Scheduling Strategy	38
18 Formulas for the Expected Makespan	44
18.1 Naive Approaches	45
18.2 Using the Configuration Relation	48
18.3 Using the Profile Relation	51
18.4 Using the $\{h, b\}$ values	52
18.5 One Chain	57
18.6 Two Chains	57
18.7 k Chains	66
18.8 Y-graphs and Psi-graphs	80
18.9 Intree Decomposition	81

Most of the following concepts and results are taken from [50].

Definition 25 (Exponential Distribution). A continuous random variable X with support \mathbb{R}^+ is (negatively) exponentially distributed with parameter/rate $\lambda > 0$, denoted by $X \sim \exp(\lambda)$, if its probability density function f_X is

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

that is, its cumulative distribution function is

$$F_X(x) = \mathcal{P}(X \leq x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The expected value (or *mean*) of an exponentially distributed random variable X with parameter λ is

$$\mathbb{E}(X) = \int_{\mathbb{R}} x f_X(x) dx = \frac{1}{\lambda}.$$

Given a task with processing time distributed according to $X \sim \exp(\lambda)$, this means that the expected time until this task finishes is $\frac{1}{\lambda}$.

The next property of the exponential distribution is a crucial one when regarding the calculation of the makespan.

Theorem 26 (Memorylessness). *The exponential distribution is memoryless, i.e., for $X \sim \exp(\lambda)$ and $x \geq y$ we have*

$$\mathcal{P}(X > x | X > y) = \mathcal{P}(X > x - y).$$

Proof. Starting with the left hand side and the definition of the conditional probability, we have

$$\mathcal{P}(X > x | X > y) = \frac{\mathcal{P}(X > x, X > y)}{\mathcal{P}(X > y)} = \frac{\mathcal{P}(X > x)}{\mathcal{P}(X > y)},$$

where the second equality holds because $x \geq y$. Plugging in the definition for the cumulative distribution function yields

$$\begin{aligned} \frac{\mathcal{P}(X > x)}{\mathcal{P}(X > y)} &= \frac{e^{-\lambda x}}{e^{-\lambda y}} \\ &= e^{-\lambda(x-y)} \\ &= \mathcal{P}(X > x - y). \end{aligned}$$

□

Consider X to be the processing time of a task. Then, if we want to know how much longer this task is going to be processed, the memorylessness tells us that it does not matter how long

this task has been processed so far. The probability of the task to be processed for t more time units is the same whether it was already processed s time units or not:

$$\mathcal{P}(X > s + t | X > s) = \mathcal{P}(X > t).$$

In particular, the memorylessness shows that the expected value of an exponentially distributed random variable is $\frac{1}{\lambda}$ in any case, even if it has a conditional lower bound.

What we can actually show is that the property of memorylessness for a continuous random variable is exclusive for the exponential distribution, see the following result.

Theorem 27. *Every memoryless continuous random variable is exponentially distributed.*

The proof can be found in [24].

Next, we have the following result.

Lemma 28. *Let X_1, \dots, X_n be independent exponentially distributed random variables with parameters $\lambda_1, \dots, \lambda_n$ respectively. Then $X_1 \wedge \dots \wedge X_n$ is exponentially distributed with parameter $\lambda_1 + \dots + \lambda_n$.*

Proof. The proof is done for $n = 2$, the case for arbitrary n can be proven by induction.

$$\begin{aligned} \mathcal{P}(X_1 \wedge X_2 \leq x) &= \mathcal{P}(X_1 \leq x) + \mathcal{P}(X_2 \leq x) - \mathcal{P}(X_1 \leq x, X_2 \leq x) \\ &= 1 - e^{-\lambda_1 x} + 1 - e^{-\lambda_2 x} - (1 - e^{-\lambda_1 x})(1 - e^{-\lambda_2 x}) \\ &= 1 - e^{-\lambda_1 x} + 1 - e^{-\lambda_2 x} - 1 + e^{-\lambda_1 x} + e^{-\lambda_2 x} - e^{-\lambda_1 x} e^{-\lambda_2 x} \\ &= 1 - e^{-(\lambda_1 + \lambda_2)x}. \end{aligned}$$

□

Let $X \sim \exp(\lambda)$ and consider the processing times of the tasks independent and identically distributed according to X . Then, due to the memorylessness of the exponential distribution, the expected remaining processing time of a task that has already been processed for i time units is again $\frac{1}{\lambda}$.

Furthermore, when regarding a two machine problem, at any given time when both machines are busy processing two tasks, the expected time until one of them finishes is $\frac{1}{2\lambda}$ time units.

16 Calculating the Expected Makespan

Now consider a scheduling problem with two processors and independent, identically and exponentially distributed processing times, each with parameter λ , or in the notation introduced in Chapter 4 on page 12:

$$2 | p_i \sim \exp(\lambda) | \mathbb{E}(C_{\max}).$$

As an abbreviation, we also write “exponential processing times” for “independent, identically and exponentially distributed processing times”.

Because the minimum of two exponential distributions is again an exponential distribution, see Lemma 28, we can easily determine the value $\mathbb{E}(X \wedge Y) = \frac{1}{2\lambda}$. And due to the memorylessness of the exponential distribution, it does not matter how long an active task has already been processed, it is always distributed with mean λ . This means that both processed tasks are again identically distributed and are equally likely to finish first, i.e.,

$$\mathcal{P}(X \wedge Y = X) = \mathcal{P}(X \wedge Y = Y) = \frac{1}{2}.$$

In total, calculating the expected makespan of a configuration of such a scheduling problem can be done as in (15.1):

$$\begin{aligned} C_c(c) &= \frac{1}{2\lambda} + \frac{1}{2} \int_0^{\infty} \lambda e^{-\lambda x} C_c(c_x) dx + \frac{1}{2} \int_0^{\infty} \lambda e^{-\lambda x} C_c(c_y) dx \\ &= \frac{1}{2\lambda} + \frac{1}{2} C_c(c_x) \underbrace{\int_0^{\infty} \lambda e^{-\lambda x} dx}_{=1} + \frac{1}{2} C_c(c_y) \underbrace{\int_0^{\infty} \lambda e^{-\lambda x} dx}_{=1} \\ &= \frac{1}{2\lambda} + \frac{1}{2} C_c(c_x) + \frac{1}{2} C_c(c_y), \end{aligned} \quad (16.1)$$

with c_x and c_y denoting the configurations where the task x or y finishes first (and is then missing from the graph), respectively. The crucial part to lose the integrals in this calculation is between the first and second lines when we remove the values $C_c(c_x)$ and $C_c(c_y)$ from the integrands. Because of the memorylessness, these values do not depend on the integration variable x , and thus, can be removed from the integral. The equation in (15.2) amounts to

$$\begin{aligned} C_c(c) &= \frac{1}{2} \int_0^{\infty} f_{X \wedge Y}(x) (x + C_c(c_x)) dx + \frac{1}{2} \int_0^{\infty} f_{X \wedge Y}(x) (x + C_c(c_y)) dx \\ &= \frac{1}{2} \int_0^{\infty} 2\lambda e^{-2\lambda x} (x + C_c(c_x)) dx + \frac{1}{2} \int_0^{\infty} 2\lambda e^{-2\lambda x} (x + C_c(c_y)) dx. \end{aligned} \quad (16.2)$$

It is easy to see that both approaches (16.1) and (16.2) are identical:

$$\begin{aligned} (16.2) &= \frac{1}{2} \int_0^{\infty} 2\lambda e^{-2\lambda x} (x + C_c(c_x)) dx + \frac{1}{2} \int_0^{\infty} 2\lambda e^{-2\lambda x} (x + C_c(c_y)) dx \\ &= \int_0^{\infty} 2\lambda e^{-2\lambda x} \cdot x dx + \frac{1}{2} (C_c(c_x) + C_c(c_y)) \int_0^{\infty} 2\lambda e^{-2\lambda x} dx \\ &= [(-e^{-2\lambda x})]_0^{\infty} + \int_0^{\infty} e^{-2\lambda x} dx + \frac{1}{2} (C_c(c_x) + C_c(c_y)) \cdot 1 \\ &= \frac{1}{2\lambda} + \frac{1}{2} C_c(c_x) + \frac{1}{2} C_c(c_y) \\ &= (16.1) \end{aligned} \quad (16.3)$$

In the formulas above we drop the superscript for the scheduling strategy in the notation for the expected makespan, meaning that we explicitly use *HLF*. Here, it is used for easier readability, but later on, we show that this is actually justified, see Chapter 17.

Despite not having to consider active tasks thanks to the memorylessness, these formulas still do not cover all cases needed for calculating the expected makespan. They work only when there are at least two available tasks so that there are always two different successor configurations that are possible. However, the formulas are only becoming simpler in the case that there is at most one available task. If we only consider trees, then these cases occur exactly when the underlying graph is a chain. In particular, (16.1) then results in

$$C_c(c) = \frac{1}{\lambda} + C_c(c_x),$$

where c_x is the unique successor configuration of c obtained by removing task x . The expected time until the one processed task finishes is exactly the mean value of the exponential distribution, i.e., $\frac{1}{\lambda}$. The very last case that we need in order to fully describe the expected makespan via this recurrence is when there are no tasks left in the graph, i.e., when the underlying graph is empty. Obviously, the formula is then

$$C_c(\varepsilon) = 0,$$

where ε stands for the empty graph, or the empty configuration, respectively. In case the graph is a chain, we can even remove the recurrence, and have the formula

$$C_c(c) = \frac{1}{\lambda} \cdot n, \tag{16.4}$$

where n denotes the height of the chain. Note that for $n = 0$, we have exactly the case for the empty graph.

Another interesting property of exponential processing times is that both equations in (16.1) and (16.2) are scaled by the factor $\frac{1}{\lambda}$. Consider (16.1) as an example. For the first summand this is obvious. For the recursive calls to C_c this is also true, because every new C_c -call on another level of the recursion brings another summand $\frac{1}{2\lambda}$ and the base cases of the recursion are given by (16.4), which also scale with $\frac{1}{\lambda}$. This is why, from now on, w.l.o.g., we always use $\lambda = 1$ as the parameter for the exponential distribution of the processing times. Then, the expected time until a specific task finishes is 1. If there are two tasks being processed simultaneously, then the expected time until one of those two finishes is $\frac{1}{2}$.

17 The Optimal Scheduling Strategy

In this section, we prove that *HLF* is the optimal scheduling strategy when considering exponentially distributed processing times on two processors when the underlying precedence constraints are given by an intree, i.e.,

$$2|p_i \sim \exp(1); \text{intree} | \mathbb{E}(C_{\max}). \tag{17.1}$$

What we use in order to show this is that we can just as well calculate the expected makespan of a given graph or configuration using only its profile. For this definition, we explicitly use *HLF*, which makes it differ from what we have seen before in (16.1) although the recurrence itself is very similar to it.

Originally, Chandy and Reynolds [49] proved that *HLF* is optimal for (17.1) in 1979. However, as their proof is not always very detailed, in the following, we recreate their proof with some other methods as well and (hopefully) add more clarity to it.

Lemma 29. *Let G be an intree with $\text{profile}(G) = p$, then the expected makespan of G when applying any *HLF* can be calculated by:*

(i) *If $p = ()$, then $C_p(p) = 0$,*

(ii) *If $p = (1)$, then $C_p(p) = 1$,*

(iii) *If G is a chain, i.e. $p_L = 1$ and $n = L$, then*

$$C_p(p_1, \dots, p_L) = 1 + C_p(p_1, \dots, p_{L-1}),$$

(iv) *If the highest level has more than one task, i.e. $p_L > 1$ and $n > L$,*

$$C_p(p_1, \dots, p_L) = \frac{1}{2} + C_p(p_1, \dots, p_{L-1}),$$

(v) *If the highest level has only one task and G is not a chain, i.e. $p_L = 1$ and $n > L$, then*

$$C_p(p_1, \dots, p_{L'}, \dots, p_L) = \frac{1}{2} + \frac{1}{2} C_p(p_1, \dots, p_{L'}, \dots, p_{L-1}) \\ + \frac{1}{2} C_p(p_1, \dots, p_{L'} - 1, \dots, p_L).$$

Proof. Cases (i) to (iii) are obvious by definition of the exponential distribution as seen in (16.1) and (16.4).

In case (iv), we have at least two tasks on the highest level. Thus, any *HLF* will choose two of those. Hence, at the next decision point, the finished task has to be one of those on the highest level. Then the profile is (p_1, \dots, p_{L-1}) . And because the expected time until of these two tasks finishes is $\frac{1}{2}$, the above equality follows.

In case (v), we have that G is not a chain, but there is only one task on the top level L . By the pigeon hole principle, this means that there is a level with more than one task, denoted by L' . Then the two highest available tasks are on level L and on level L' , respectively. Because we use *HLF*, the one task at level L will be chosen, as well as one task at level L' . The expected time until the one of these tasks finishes is again $\frac{1}{2}$. Then, with identical probability of $\frac{1}{2}$, we either process the task at the top level or the task at level L' , resulting in

$$C_p(p_1, \dots, p_{L'}, \dots, p_L) = \frac{1}{2} + \frac{1}{2} C_p(p_1, \dots, p_{L'}, \dots, p_{L-1}) \\ + \frac{1}{2} C_p(p_1, \dots, p_{L'} - 1, \dots, p_L).$$

□

Remark: Note that this lemma only works for the two processor case. Here, a singular task on the highest level can only dominate one path (the blue nodes), i.e., one task per level below, meaning that only one path of tasks consists of descendants of that task on the highest level, see Fig. 6 on the left. This is why in case (v) we only have to find the level L' for another available task which can be scheduled. For the three processor case, consider a case where two tasks on the highest level are processed. These two tasks dominate two paths now, so if we find a level L^* with $p_{L^*} \geq 3$, then we can be sure to have another available task. But if there is no level with more than two tasks, then we cannot be sure if there is an available task or not. This is because the two active tasks on the highest level can dominate two tasks per level below, see Fig. 6 in the middle, or the two dominating paths may meet and then dominate only one task per level below that, see Fig. 6 on the right. This argument is not dependent on the profile itself, which means that the profile alone does not suffice to know whether three or only two tasks can be scheduled. The same is true if only one task is on the highest level, then there can be two or three available tasks for different intrees of the same profile, see Fig. 7. In other words: given a profile, we can easily deduce whether there are at least two available tasks or only one. In contrast, we cannot deduce whether there are at least three available tasks or less from the profile alone.

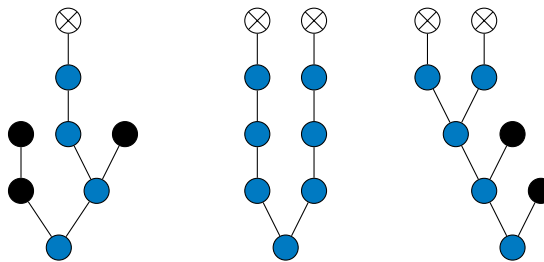


Fig. 6: Crucially different dominating paths.

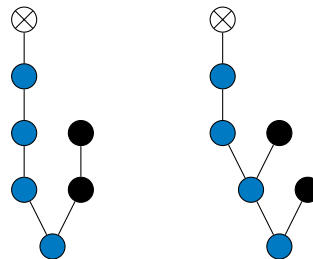


Fig. 7: The profile does not say anything about the number of available tasks.

For the following proofs, we use a concept called *flatness* of an intree, which was introduced in [12]. Here, we use a restricted version of it that is enough for our purposes.

Definition 30. Let G, H be two intrees with the same number of tasks and $\text{profile}(G) = (p(G)_1, \dots, p(G)_{L_G})$ and $\text{profile}(H) = (p(H)_1, \dots, p(H)_{L_H})$.

The flatness is a partial relation on intrees, defined as follows.

- G is as flat as H , denoted by $G \sim H$ if and only if $\text{profile}(G) = \text{profile}(H)$.
- G is as flat as or flatter than H , denoted by $G \preceq H$, if and only if there is a level i such that

$$\sum_{k=i+1}^{L_G} p(G)_k \leq \sum_{k=i+1}^{L_H} p(H)_k,$$

i.e., the number of tasks above level i in G is less than or equal to the number of tasks above level i in H .

- G is flatter than H , denoted by $G \prec H$, if and only if there is a level i such that

$$\sum_{k=i+1}^{L_G} p(G)_k < \sum_{k=i+1}^{L_H} p(H)_k,$$

i.e., the number of tasks above level i in G is **strictly** less than or equal to the number of tasks above level i in H .

Again, sometimes we use intrees and configurations interchangeably as well as the flatness relation on configurations. As the flatness does not consider active tasks, this is just as fine as comparing the respective intrees, at least for exponential processing times. Fig. 8 shows four different intrees with the following flatness relationships: $G_2 \preceq G_1$, $G_2 \preceq G_3$, $G_1 \sim G_3$, and G_4 cannot be compared to any of the other three, because it has one more task.

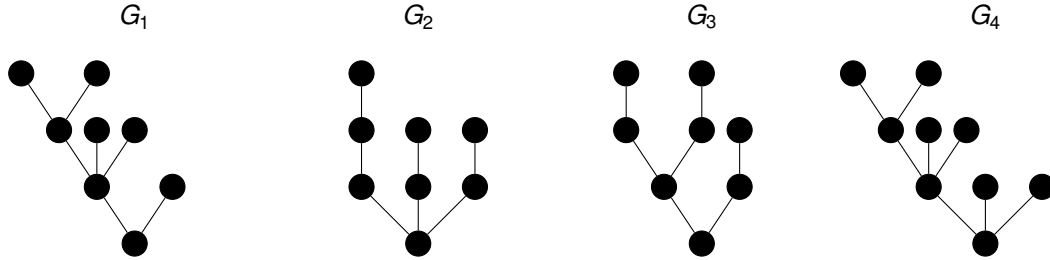


Fig. 8: Four intrees with $G_2 \preceq G_1$, $G_2 \preceq G_3$, $G_1 \sim G_3$, and G_4 is incomparable.

First, we observe that all *HLFs* will result in the same expected makespan. This was partly shown in Lemma 29. The following results show that an arbitrary *HLF* is optimal, and hence, all of them are.

Lemma 31. Let G be an intree and let G' be obtained by attaching a source of G on a lower level. Then $C_c(G) \geq C_c(G')$.

Proof. We prove this by induction on the number of tasks.

Induction Base: It holds that

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = 3.75 < 4 = C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right),$$

see Appendix C for the details.

Induction Step: Let n be the number of tasks in both G and G' . We have the scenario as seen in Fig. 9. We introduce the notation G_v for the graph/configuration $G \setminus \{v\}$, where v denotes a

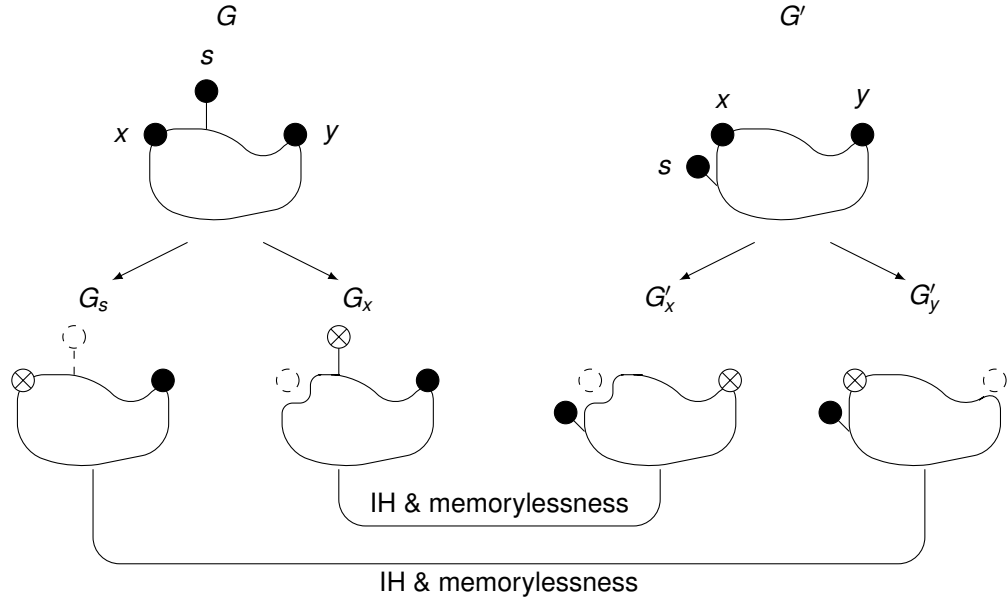


Fig. 9: G and G' and their respective successor configurations.

node corresponding to an available task, meaning that the node/task v as well as all its incident edges are removed from G . Then, the two outer configurations G_s and G'_y (i.e., the leftmost and the rightmost configurations in Fig. 9) both have underlying intrees with $n - 1$ tasks and the rightmost intree can be obtained from the leftmost intree by reattaching the source y on a lower level. Then, because of the memorylessness we do not have to specifically mark tasks as active as long as we know that they will be processed in the next time steps anyway. But as we are choosing tasks according to *HLF*, and every *HLF* leads to the same expected makespan, see Lemma 29, we just take the one with we had in the step first. In particular, this means that the task that is active is chosen to be processed next even if we do not mark it as active. The arguments for the case with the two inner configurations G_x and G'_x is the same.

□

Note that this proof does not necessarily hold for $G' \propto G$ in general, because we only reattached one task.

Lemma 32. *The expected makespan of a scheduling strategy S that uses exactly one step that is not an HLF step is worse than the expected makespan of HLF.*

Proof. Without loss of generality, let the first step of S be the non-*HLF* step and let G have more than two available tasks, where at least one of them is not on the highest level. Let the two successor configurations of G under *HLF* be denoted by c_1 and c_2 . Because S is not *HLF*, at least one of the two possible successor configurations of G under S is after a task finished which was not chosen by *HLF* in the step before. Let this successor configuration be denoted by c' . The other successor configuration c'' might be one of c_1 or c_2 or some other configuration. One obvious thing is that the intree in c'' can never be flatter than both the intrees in c_1 and c_2 , by the properties of *HLF*. Hence, we have that

$$c_1, c_2 \propto c' \text{ and } c_i \propto c''$$

for at least one $i \in \{1, 2\}$. Then with Lemma 31 we obtain

$$C_c(c'') \geq c_i \text{ and } C_c(c') > c_j$$

for at least one $i \in \{1, 2\}$ and $j \in \{1, 2\}, i \neq j$. □

With this result, it is shown that a single non-*HLF* step does not improve the expected makespan. Of course, now the question is whether **two or more** non-*HLF* steps can improve the makespan. Or in other words: can one non-*HLF* step counter the worsening effects done by other non-*HLF* steps before?

Lemma 33. *Let G and H be two intrees with the same number of tasks, and $G \propto H$. Consider an HLF being applied to G and a non-*HLF* S being applied to H . Then the expected makespan of G is at most the expected makespan of H .*

Proof. Without loss of generality, suppose that the first step applied to H is a non-*HLF* step. The claim is proven by induction on the number of tasks.

Induction Base: It holds that

$$C_c \left(\begin{array}{c} \circ \circ \bullet \\ \circ \bullet \bullet \end{array} \right) = 3.5 < 3.75 = C_c^{\text{non-}HLF} \left(\begin{array}{c} \circ \bullet \bullet \\ \circ \bullet \bullet \end{array} \right)$$

with G (on the left) being flatter than H , and

$$C_c \left(\begin{array}{c} \circ \circ \bullet \\ \circ \bullet \bullet \end{array} \right) = 3.625 < 3.75 = C_c^{\text{non-}HLF} \left(\begin{array}{c} \circ \bullet \bullet \\ \circ \bullet \bullet \end{array} \right),$$

where $C_c^{\text{non-}HLF}$ is introduced in order to denote the value of the processing time of a configuration which is not necessarily optimal (because it is not scheduled according to *HLF*) and because C_c was defined using *HLF* explicitly. The gray tasks mark tasks which will be chosen next by the strategies *HLF* or S , respectively.

Induction Step: Let n be the number of tasks in G and H , respectively. Furthermore, let G' and G'' be the successor configurations of G under HLF , and let H' and H'' be the successor configurations of H under \mathcal{S} . The underlying intrees all have $n - 1$ tasks. We show that $G', G'' \preceq H', H''$.

First, we assume that $G \preceq H$, because if G is as flat as H , then with the same arguments as in Lemma 32, we have that H' and H'' have flatter corresponding counterparts in G' and G'' . This assumption is justified anyway, because we want to use this lemma to compare an intree that stems from an HLF schedule and another from a non- HLF schedule. And the HLF intree will be flatter than the other.

So, if $G \preceq H$, then by definition, there exists a lowest level i with $p(G)_i > p(H)_i$ and $p(G)_j = p(H)_j$ for $j < i$ where $\text{profile}(G) = (p(G)_1, \dots, p(G)_{L_G})$ and $\text{profile}(H) = (p(H)_1, \dots, p(H)_{L_H})$.

This means that HLF on G schedules one task from level L_G and one task from level i , whereas a non- HLF on H schedules something which is not specified further (but we can w.l.o.g. assume that the scheduled tasks are not both the same as the ones from HLF). Now, assume for contradiction that one of H' or H'' (denoted by H^*) is flatter than both G' and G'' . Then there exists an i^* such that $p(H^*)_{i^*} > p(G^*)_{i^*}$ and $p(H^*)_j = p(G^*)_j$ for $j < i^*$, where $G^* \in \{G', G''\}$. Hence, HLF must have finished a task on level i in G and non- HLF must not have finished a task on level i or else we have $p(G^*)_i > p(H^*)_i$ in contradiction to $H^* \preceq G^*$. But this is a contradiction to the definition of HLF as it may schedule a task on level L_G which might finish before the other task on level i .

□

Combining the last results proves the optimality of all HLF scheduling strategies.

Theorem 34. Any HLF is optimal for the scheduling problem $2|p_i \sim \exp(1); \text{intree}| \mathbb{E}(C_{\max})$.

18 Formulas for the Expected Makespan

One of the goals of this work is to calculate the optimal expected makespan of a scheduling problem. In a deterministic setting, probably the easiest way to do this is to just run the optimal strategy and evaluate the resulting schedule. But there are some issues with this approach. Firstly, we must know the optimal strategy in order to use it, which is obviously the hardest part in most cases. Secondly, this is just not possible when we consider stochastic scheduling. As argued before, one schedule does not tell us much about optimality. Even if we know the optimal strategy in terms of optimizing the expected optimality criterion, then there is still some variation in its outcome. One possibility to counter this would be to create several schedules and calculate the arithmetic mean of all these sample schedules, but we would probably need a lot of samples to have some meaningful result in the end. Other than that, creating the schedule may take a long time. So the goal here is to calculate the makespan without actually creating a schedule.

Consider the $2|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max})$ scheduling problem, then we know that *HLF* is optimal, see Chapter 17. So, at least we know the optimal strategy. In this section, we elaborate several different approaches to obtain the expected makespan of an instance of this scheduling problem.

18.1 Naive Approaches

When we apply any *HLF* scheduling strategy to our scheduling problem, we observe that at some point, the remaining intree becomes a chain. From that point on, there is always at least one processor which is idle. Without loss of generality, let the first processor P_1 be the one which processes all the remaining tasks in the chain while P_2 is idle and does not do anything for the remainder of the schedule. Because the underlying precedence relation is an intree, it will never be the case again that two tasks will be available at the same time. We consider the first time when such a chain occurs, and call the topmost task in this chain the *head of the chain*. Note that this must be an available task at that time, but it may also be a task which is already being processed, i.e., active. In both cases, it is the first (in the sense of time in the schedule) task that is being processed alone on a processor, while the other processor is idle. Consider the intree in Fig. 2 from page 9. Then, Fig. 10 depicts two examples of possible chains as subtrees of that intree. Already processed tasks are drawn with dashed lines. Both of those configurations can occur in an *HLF* schedule, although with different probabilities.

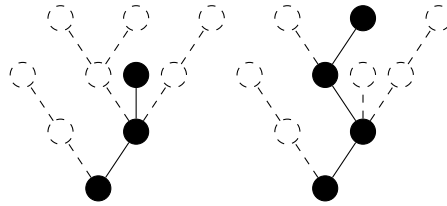


Fig. 10: Different heads of the chain for the same intree.

Using the fact that the processing times of the tasks are exponentially distributed, and thus, memoryless, we can give a short formula for calculating the expected makespan of any *HLF*.

Lemma 35. *The optimal expected makespan of the $2|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max})$ scheduling problem can be calculated as*

$$\mathbb{E}(C_{\max}) = \sum_{x \in T} \mathcal{P}(x \text{ is head of the chain}) \cdot \left(\frac{1}{2} \cdot (n - \text{level}(x)) + \text{level}(x) \right) \quad (18.1)$$

Proof. Because the processing times are exponentially distributed we have:

Case 1: Both processors are busy. Then let X and Y denote the two random variables that describe the processing times of the two processed tasks. By definition, X and Y are independent and identically, exponentially distributed with mean 1, even if one (or both) of the tasks has (have) already been processed for some time. Of course, this

is due to the memorylessness of the exponential distribution. Hence, one of the those processed tasks is expected to finish within $\mathbb{E}(X \wedge Y) = \frac{1}{2}$ time units.

Case 2: Only one processor is busy and the other is idle. Then one processed task is expected to finish within $\mathbb{E}(X) = 1$ time unit.

We know that at some decision point t in the schedule, a chain must occur, i.e., during the time interval $[0, t]$ both processors are busy, and during the interval $(t, \mathbb{E}(C_{\max})]$ only one processor is busy and the other one is idle. That means that at time t the head of the chain x is still being processed. The final chain is then of height $\text{level}(x)$, and the tasks being processed in the interval $[0, t]$ are exactly all the tasks which are not in the final chain, of which there are $n - \text{level}(x)$, i.e., all remaining tasks. Using the results from the two cases above, we get that the expected makespan for the case that x is the head of the chain is

$$\frac{1}{2} \cdot (n - \text{level}(x)) + \text{level}(x).$$

The result in the lemma is derived from summing over all possible tasks for head of the chain. \square

Note that (18.1) can be rewritten as

$$\mathbb{E}(C_{\max}) = \sum_{x \in \mathcal{T}} \mathcal{P}(x \text{ is head of the chain}) \cdot \left(\frac{1}{2} \cdot (n + \text{level}(x)) \right),$$

which is just a weighted sum $\sum_x w_x c_x$ with $w_x = \mathcal{P}(x \text{ is head of the chain})$ and $c_x = \frac{1}{2}(n + \text{level}(x))$.

Consider an example with the intree given in Fig. 11.

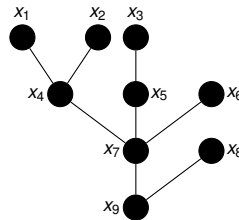


Fig. 11: An intree with profile $(1, 2, 3, 3)$.

The probabilities from (18.1) are given by

$$\begin{aligned}\mathcal{P}(x_1 \text{ is head of the chain}) &= \frac{1}{32}, \\ \mathcal{P}(x_2 \text{ is head of the chain}) &= \frac{1}{32}, \\ \mathcal{P}(x_3 \text{ is head of the chain}) &= \frac{1}{16}, \\ \mathcal{P}(x_4 \text{ is head of the chain}) &= \frac{1}{8}, \\ \mathcal{P}(x_5 \text{ is head of the chain}) &= \frac{1}{8}, \\ \mathcal{P}(x_6 \text{ is head of the chain}) &= \frac{3}{16}, \\ \mathcal{P}(x_7 \text{ is head of the chain}) &= \frac{15}{64}, \\ \mathcal{P}(x_8 \text{ is head of the chain}) &= \frac{15}{64}, \\ \mathcal{P}(x_9 \text{ is head of the chain}) &= 0.\end{aligned}$$

Given these probabilities, we can use (18.1) to calculate the makespan:

$$\begin{aligned}\mathbb{E}(C_{\max}) &= \frac{1}{32} \left(\frac{1}{2} \cdot 5 + 4 \right) + \frac{1}{32} \left(\frac{1}{2} \cdot 5 + 4 \right) + \frac{2}{32} \left(\frac{1}{2} \cdot 5 + 4 \right) \\ &\quad + \frac{4}{32} \left(\frac{1}{2} \cdot 6 + 3 \right) + \frac{4}{32} \left(\frac{1}{2} \cdot 6 + 3 \right) + \frac{6}{32} \left(\frac{1}{2} \cdot 6 + 3 \right) \\ &\quad + \frac{7}{32} \left(\frac{1}{2} \cdot 7 + 2 \right) + \frac{7}{32} \left(\frac{1}{2} \cdot 7 + 2 \right) \\ &= \frac{1}{32} (6.5 + 6.5 + 2 \cdot 6.5 + 4 \cdot 6 + 4 \cdot 6 + 6 \cdot 6 + 7 \cdot 5.5 + 7 \cdot 5.5) \\ &= \frac{187}{32} \\ &= 5 \frac{27}{32}.\end{aligned}$$

Right away, we can see the difference to the scheduling problem's deterministic counterpart $2 \lfloor p_i = 1; \text{intree} \rfloor C_{\max}$. Using Coffman and Graham's algorithm [14] for the deterministic problem, we obtain an optimal makespan of 5. This immediately shows that the makespan of an optimal deterministic schedule and the expected makespan of an optimal stochastic schedule may very well differ, and in general, do so. Again, there may be a very short stochastic schedule, even shorter than an optimal deterministic schedule, but of course, only with a certain probability. Generally, the optimal stochastic makespan is higher than the optimal deterministic makespan of the equivalent scheduling problem.

Another observation is that for tasks x_i and x_j that lie on the same level, the values $\frac{1}{2} (n + \text{level}(x_i))$ and $\frac{1}{2} (n + \text{level}(x_j))$ are the same. With this, we directly get another way to sum all tasks.

Lemma 36. *The optimal expected makespan can be calculated by*

$$\mathbb{E}(C_{\max}) = \sum_{i=1}^L \left(\frac{1}{2} (n+i) \cdot \sum_{\substack{x \in \mathcal{T}, \\ \text{level}(x)=i}} \mathcal{P}(x \text{ is head of the chain}) \right).$$

Obviously, the issue with these approaches so far is to calculate the probabilities that a certain task is the head of the chain. To the best of our knowledge, these approaches require a great amount of calculation, and the a priori knowledge of all possibilities. For larger inputs, this could be quite inefficient. This is why we present several other methods to calculate the expected makespan in the following.

18.2 Using the Configuration Relation

In the $2|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max})$ setting, a configuration simplifies to (G', x) , with x being the currently processed task. Because of the memorylessness of the exponential distribution, it does not matter how long the task has already been processed, its expected remaining processing time is always the expected value of the distribution. Note that x can also be missing in three different cases:

- at the start of the schedule,
- one task has been processed alone and just finishes, i.e., the configuration is a chain,
- two tasks have been processed and both finish at the same time.

But the last case occurs with a probability of 0, because we are dealing with continuous random variables here.

Now, we define a relation R on the set of configurations as follows. A pair of configurations (c_1, c_2) is contained in R if and only if, when applying a fixed scheduling strategy, there is a direct transition from c_1 to c_2 with some probability greater than zero.

As an example we consider level-oriented *HLF*. Furthermore, we define a cost function for a pair of configurations in R corresponding to the transition probabilities between the configurations. Fig. 12 shows the Hasse diagram of the relation R for the graph in Fig. 11. As defined before, a crossed node in a configuration corresponds to an active task. Whenever there are two arcs going out of a configuration, each of them has a value of $\frac{1}{2}$ assigned to it as this refers to the transition probability of going to either of those successor configurations (it is equally probable for either of the processed tasks to finish first). For only one possible successor configuration, that value is of course 1. In order not to overload the picture, the transition probabilities are not displayed. The number to the right of a configuration indicates the probability that it occurs during an execution of *HLF*. The probabilities in a given level sum up to 1, as every way a scheduling strategy takes to get to the last configuration on the bottom must cross this level (i.e., no scheduling strategy can skip a level), and as all configurations on the same level are mutually exclusive.

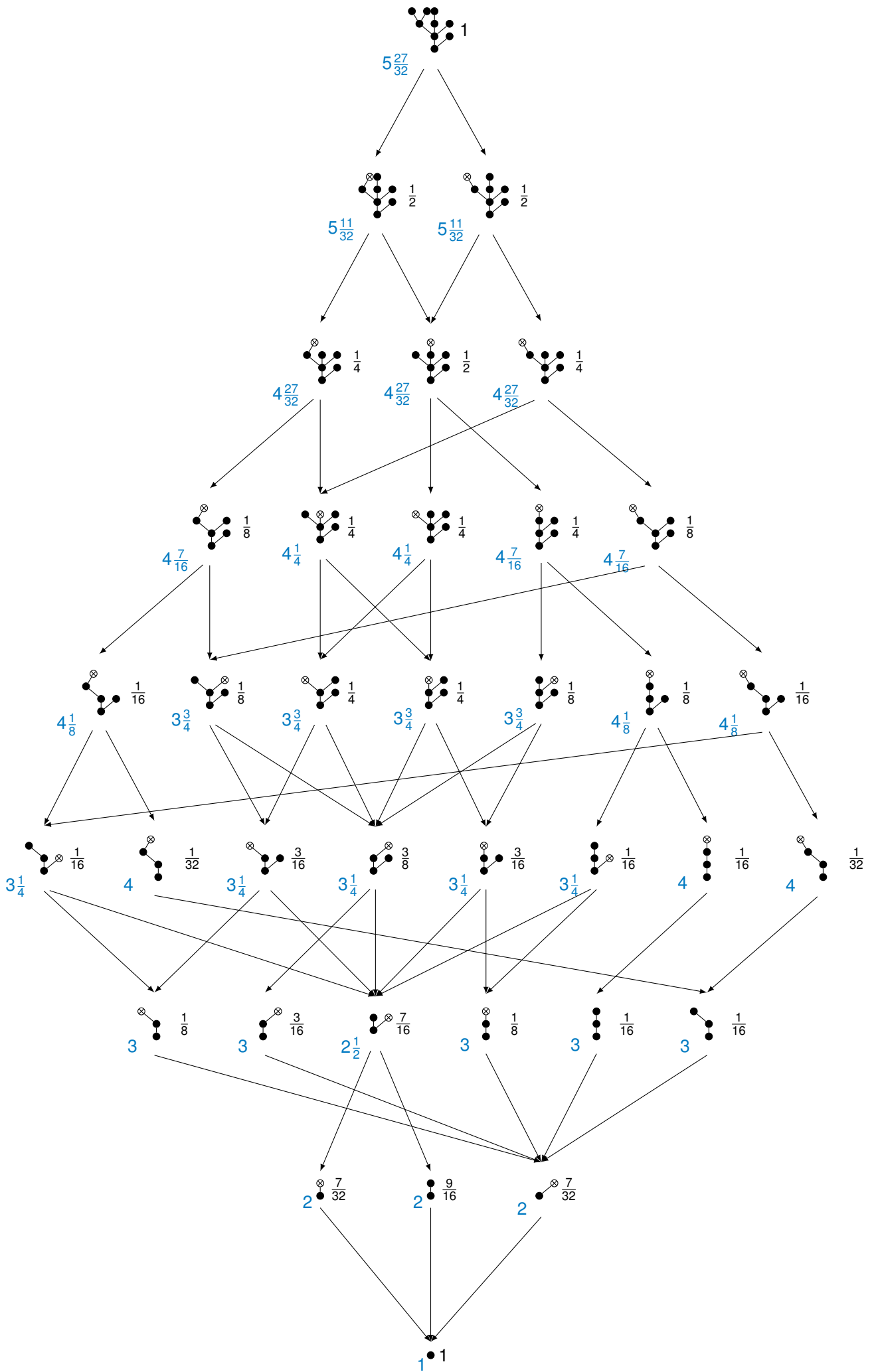


Fig. 12: Hasse diagram for the configuration relation of the intree in Fig. 2.

The graph in Fig. 12 can be condensed further if we allow isomorphic configurations to be merged. As these do not behave differently under any (reasonable)¹ strategy, they will be merged in future figures. In the very special case of two processors and exponential processing times, we can even disregard active tasks, and just focus on isomorphisms of the underlying intrees. In particular, this means that all the intrees in Fig. 13 are considered isomorphic (in the $2|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max})$ setting). Fortunately, the tree isomorphism problem can be handled very easily, see for example [7].

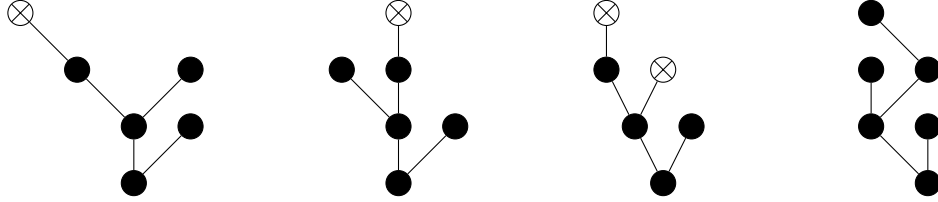


Fig. 13: Four isomorphic intrees.

The next lemma describes how the configuration (and the configuration relation) can be used to calculate the expected makespan of an *HLF* schedule.

Lemma 37. *Let c be a configuration with underlying intree precedence constraints and S a scheduling strategy. Then:*

- (i) *If $c = (\{\text{root}\}, \emptyset)$, i.e., the intree consists of a singular node called “root” without any arcs and without any active tasks, then $C_c^S(c) = 1$,*
- (ii) *If c has only one successor configuration c' , i.e., it corresponds to a chain, then*

$$C_c^S(c) = 1 + C_c^S(c'),$$

- (iii) *If c has two successor configurations c' and c'' , then*

$$C_c^S(c) = \frac{1}{2} + \frac{1}{2}C_c^S(c') + \frac{1}{2}C_c^S(c'').$$

Proof. The first case is obvious since it refers to a configuration with only one task left, e.g. at the very bottom of Fig. 12 or on the left of Fig. 14. The second case is as obvious: if c is succeeded by only one other configuration c' , then the subgraph in c is a chain. Because we have exponential processing times, and $\lambda = 1$, the claim follows. The only thing left to show is the third claim. As seen in Fig. 14 on the right, c has two successor configurations, each equally probable to be reached, with a probability of $\frac{1}{2}$. This is only the case when two tasks are processed simultaneously. The expected time until one of those tasks finishes is $\frac{1}{2}$. Hence,

¹We may as well define a strategy that chooses tasks according to their names or indices. But it should be obvious that these kinds of strategies are not the ones we want to focus on, and, in general, are not meaningful.

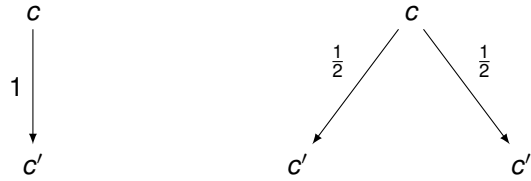


Fig. 14: A configuration and its successor configurations.

after half a time unit we arrive either at configuration c' or at c'' . Thus, the expected makespan from c on is given by

$$C_c^S(c) = \frac{1}{2} + \frac{1}{2} C_c^S(c') + \frac{1}{2} C_c^S(c'').$$

Note that all this works only with the exponential distribution as we can disregard active tasks because of the memorylessness. \square

In the following, we will use this result with *HLFs* only, i.e., only with the $C_c := C_c^{HLF}$ values. The C_c values of all feasible level-oriented *HLF* configurations can be seen in Fig. 12 on the lower left of each configuration node in blue. As can be seen in the figure as well, the expected makespan of configurations with identical profiles is the same.

18.3 Using the Profile Relation

We have already seen in Chapter 17 in Lemma 29 how to calculate the expected makespan using the concept of a profile. Of course, using the approach above with the concept of a configuration does not change the result. The following lemma proves that.

Lemma 38. *Let G be an intree, c a configuration of G , and p the corresponding profile of c . Then $C_c(c) = C_p(p)$.*

Proof. By induction on n , the number of tasks in c .

Induction Base: For $n = 1$ or $n = 2$, the claim holds by definition of $C_c(c)$ and $C_p(p)$.

Induction Step: Now let c be a configuration with $n + 1$ tasks, then there are two cases.

Case 1: c has only one successor configuration c' , see Fig. 14 on the left. Then the graph in c is a chain, and c' is a chain without the topmost task of the graph of c . But then, there is only one successor profile p' , for which the claim already holds by assumption. Hence,

$$\begin{aligned} C_c(c) &= 1 + C_c(c') \\ &\stackrel{\text{IH}}{=} 1 + C_p(p') \\ &= C_p(p). \end{aligned}$$

Case 2: c has two successor configurations c' and c'' , see Fig. 14 on the right. Then either the two corresponding successor profiles are different, see Fig. 15 on the left, or they are the same, see Fig. 15 on the right. In both cases, we can write that

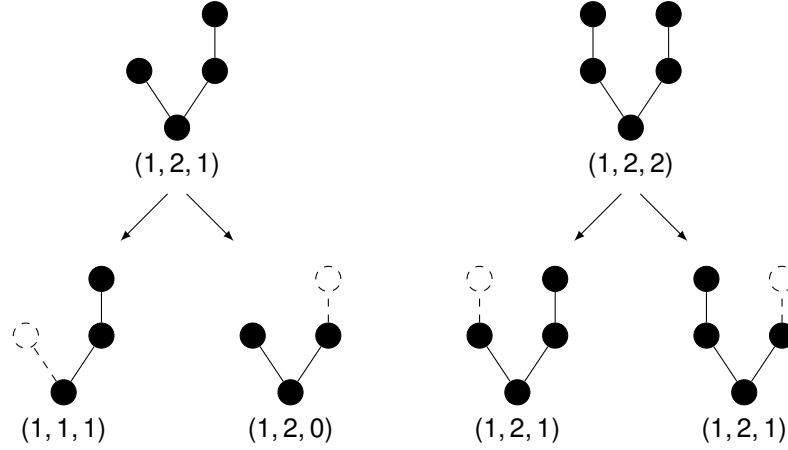


Fig. 15: Successor configurations can have different profiles.

$$C_c(c') + C_c(c'') = C_p(p') + C_p(p''),$$

where, in the second case, we have that $C_p(p') = C_p(p'')$. Hence,

$$\begin{aligned} C_c(c) &= \frac{1}{2} + \frac{1}{2}C_c(c') + \frac{1}{2}C_c(c'') \\ &\stackrel{\text{IH}}{=} \frac{1}{2} + \frac{1}{2}C_p(p') + \frac{1}{2}C_p(p'') \\ &= C_p(p). \end{aligned}$$

□

Again, the issue with this approach is that it is actually needed to know the configuration graph as seen in Fig. 12.

In Fig. 16, we can see the Hasse diagram of the feasible profile values of the intree in Fig. 11 and the profile relation. As for the configuration relation in Fig. 12, the values C_p are written at the lower left of the nodes in blue. We observe that these are the same values as in the corresponding configurations in Fig. 12. To the right of each node, there is the probability that this profile occurs during an *HLF* schedule.

18.4 Using the $\{h, b\}$ values

Recall the fact from (18.1) that, given a head of the chain, the expected remaining processing time is only dependent on its level. This is the idea behind the next approach that identifies an

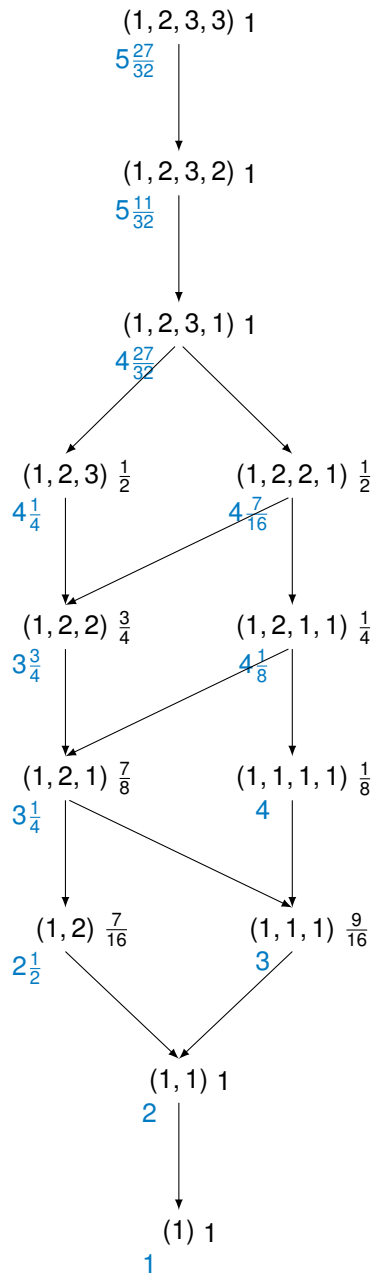


Fig. 16: Hasse diagram for the profile relation of the intree in Fig. 2.

intree G by only two numbers h and b , where h denotes the height of G , i.e., the length of a longest path from a node to the root, and b is the number of tasks in G which are not on this longest path. Using our usual notation, we have $h = L$ and $b = n - L$. Again, we can define a relation similar to those in the sections before describing the possible $\{h, b\}$ values any HLF encounters.

Lemma 39. *Let G be an intree with values $\{h, b\}$. Define $C_{h,b}(h, b)$ as the expected makespan of an HLF schedule of G . Then:*

(i) *If G is a chain, i.e., $b = 0$, then $C_{h,b}(h, 0) = h$,*

(ii) *If G has exactly one task on the highest level, i.e., $p_L = p_h = 1$, then*

$$C_{h,b}(h, b) = \frac{1}{2} + \frac{1}{2}C_{h,b}(h-1, b) + \frac{1}{2}C_{h,b}(h, b-1), \quad (18.2)$$

(iii) *If G has at least two tasks on the highest level, i.e., $p_L = p_h \geq 2$,*

$$C_{h,b}(h, b) = \frac{1}{2} + C_{h,b}(h, b-1). \quad (18.3)$$

Proof. The first case is obvious. For (18.2) there are two cases: either the one task on the top level finishes, then the successor values are $\{h-1, b\}$; or one other task finishes, which is not on the longest chain, then the successor values are $\{h, b-1\}$. Each of those possibilities has a probability of $\frac{1}{2}$ as argued before. For (18.3) there are no distinctions to make as there will be at least one task left on the top level h , which means that the successor values are $\{h, b-1\}$. \square

In contrast to the definitions for the expected makespan with the configuration, see Definition 16 on page 15, or with the profile, see Definition 19 on page 16, we do not define $C_{h,b}$ with general scheduling strategies \mathcal{S} . We could do this, but we will not use this any further.

Note that the values of h and b can be easily determined given the profile, namely by

$$(p_1, \dots, p_L) \text{ corresponds to } \left\{ L, \sum_{i=1}^L (p_i - 1) \right\}.$$

Also note that this correspondence is not bijective. For example, Fig. 17 shows that the values $\{3, 1\}$ might refer to an intree with profile $(1, 2, 1)$ (left) or to an intree with profile $(1, 1, 2)$ (right).

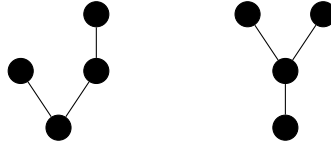


Fig. 17: Two different intrees with the same $\{h, b\}$ values.

If we are additionally given the original intree, then the profile is uniquely determined. This seems to make sense, as we need the profile of the intree in order to distinguish between those

cases defined above. This way, we can come up with a similar recursive relation to that of the profile relation.

The Hasse diagram that shows all feasible combinations of $\{h, b\}$ values of the intree in Fig. 2 on page 9 and their relation can be seen in Fig. 18. Again, at the lower left of each node in blue, the $C_{h,b}$ values are drawn that describe the expected makespan of the intree with a profile that corresponds to the one in this node. And again, these values are the same as in Fig. 16 and Fig. 12, respectively. The occurrence probabilities are given to the right of each node again, as well.

Lemma 40. *Let G be an intree, p the profile of G , and $\{h, b\}$ its corresponding values. Then $C_p(p) = C_{h,b}(h, b)$.*

Proof. By induction on n , the number of tasks in p . For $n = 1$ or $n = 2$, the claim holds by definition of C_p and $C_{h,b}$. Now, let p be a profile with $n + 1$ tasks. Then we have to distinguish between several cases:

Case 1: p describes a chain, i.e. $p_h = 1, h = n + 1, b = 0$. Then, p has only one possible successor profile $p' = (p_1, \dots, p_{h-1})$, thus

$$\begin{aligned} C_p(p) &= 1 + C_p(p') \\ &\stackrel{\text{IH}}{=} 1 + C_{h,b}(h-1, 0) \\ &= C_{h,b}(h, 0). \end{aligned}$$

Case 2: p is no chain and it holds that $p_h > 1$. Then, p has only one possible successor profile $p' = (p_1, \dots, p_{h-1})$, thus

$$\begin{aligned} C_p(p) &= \frac{1}{2} + C_p(p') \\ &\stackrel{\text{IH}}{=} \frac{1}{2} + C_{h,b}(h, b-1) \\ &= C_{h,b}(h, b). \end{aligned}$$

Case 3: p is no chain and it holds that $p_h = 1$. Then, p has two possible successor profiles $p' = (p_1, \dots, p_{h-1})$ and $p'' = (p_1, \dots, p_{L'} - 1, \dots, p_h)$ (with L' as before as the largest index with $p_{L'} > 1$), thus

$$\begin{aligned} C_p(p) &= \frac{1}{2} + \frac{1}{2} C_p(p') + \frac{1}{2} C_p(p'') \\ &\stackrel{\text{IH}}{=} \frac{1}{2} + \frac{1}{2} C_{h,b}(h-1, b) + \frac{1}{2} C_{h,b}(h, b-1) \\ &= C_{h,b}(h, b), \end{aligned}$$

where the last equalities each hold because of the definition of $C_{h,b}$. □

Part IV defined several equivalent approaches to calculate the expected makespan of an optimal *HLF* schedule. The issue with all these approaches is that more information about the structure

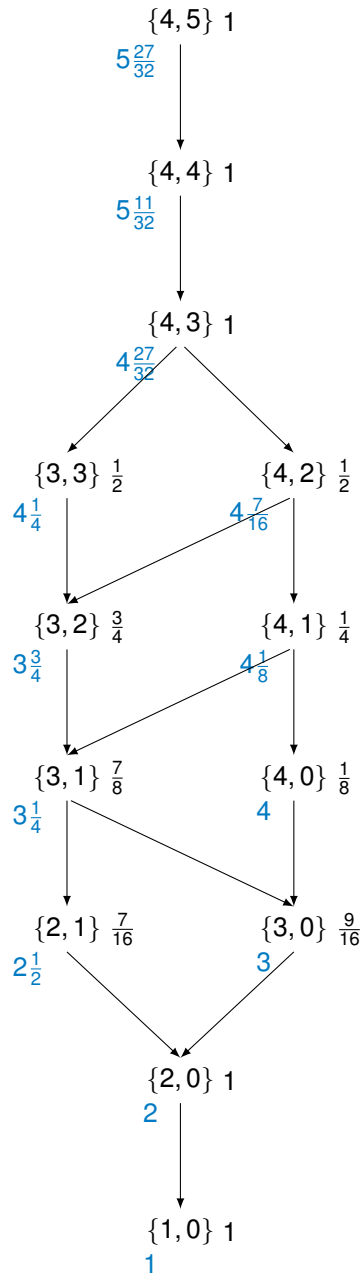


Fig. 18: Hasse diagram of the $\{h, b\}$ -relation for the intree in Fig. 11.

of the schedule in form of the configuration graph, or the profile graph, is needed. This is why in the following section, we consider a special case of our scheduling problem in order to derive some formulas for the expected makespan with only the information about the structure of the input graph, but without any information about the structure of the configuration graph. In particular, we consider the problem where the precedence constraints are restricted to only a sequence of chains. Then the scheduling problem we consider is denoted by

$$2|p_i \sim \exp(1); \text{chains} | \mathbb{E}(C_{\max}).$$

18.5 One Chain

We start with the easiest (and very trivial) example: only one chain. In this case, a second processor does not help in processing these tasks, as there is never a time when two tasks are available. The expected makespan for such a chain of length (or height) n is of course n . Before we head on, we need some notations for the appearing precedence constraints.

Definition 41. Given a set \mathcal{T} of n tasks, a *chain* of these tasks is a set of precedence constraints such that $j_1 \rightarrow j_2 \rightarrow \dots \rightarrow j_n$ with $j_i \in \mathcal{T}, i = 1, \dots, n$. We denote a chain of length n by $[n]$.

Using the notation from the definition above, we have

$$C_c([n]) = n, \tag{18.4}$$

when applying *HLF* (or any other scheduling strategy that does not leave processors unnecessarily idle for that matter).

18.6 Two Chains

The situation becomes a lot more interesting if we consider a second chain, i.e., some precedence constraints for the tasks which consist of two separate independent chains of generally different lengths. To this end, we define the following.

Definition 42. Given a set \mathcal{T} of n tasks, k chains correspond to precedence constraints for \mathcal{T} such that the connected components of the corresponding graph are chains where the combined number of tasks in all chains is exactly n . For chain lengths l_1, \dots, l_k with $l_1 + \dots + l_k = n$, we denote these precedence constraints by $[l_1, \dots, l_k]$.

For reasons of simplicity we might assume without loss of generality that $l_1 \geq l_2 \geq \dots \geq l_k$. First, we focus on the case with only two chains, i.e. $[s, t]$ with $s \geq t$.

18.6.1 Using the Combinatorial Approach to Find a Closed Form

The interesting part is to determine the expected makespan of the schedule without actually running the schedule. To this end, we try to reduce this case to one where there is only one of

the two chains left, because for only one chain we can determine the expected makespan right away with (18.4). So, eventually, we will end up in a configuration where one of the two chains is fully processed. Then we have three parameters to look after:

- How many tasks are left in the one chain that is not fully processed?
For this, we consider the different cases separately and use the law of total probability, i.e., we have to sum over all possibilities multiplied by their respective probabilities.
- How much time has passed since the beginning of the schedule up to the point where only one chain remains? And how much time will pass until the end of the schedule?
This is easily determined. Given two chains $[s, t]$, consider the configurations $([i, 0], \text{top of first chain})$ or $([0, i], \text{top of second chain})$ ², respectively, i.e., one chain is fully processed and the other has i tasks left.³ Then, the expected time passed until this configuration is exactly $\frac{1}{2}(n - i)$, and the expected remaining processing time until completion of the schedule is i .
- What is the probability to get to this configuration? The probability of arriving at a certain configuration is the hardest part of the calculation. We split this into two cases, one for either of the chains being the first chain that is completed.

Consider the configuration $[s', 0]$. Then, looking at the configuration graph, we know that the probability of arriving at a given configuration is of the form

$$p \cdot 2^{-\ell},$$

where ℓ denotes its distance from the starting configuration and p is the number of paths to reach it. The distance ℓ can be easily calculated, it is exactly the number of tasks that have been processed before, in this case $\ell = n - s'$. The number of paths leading to this configuration is determined by the binomial coefficient $\binom{n-s'}{t}$, because there are $n - s'$ tasks to be processed until then, and exactly t of them must be from the second chain with length t . Considering a timeline of finished tasks, we now have $\binom{n-s'}{t}$ possible ways to arrange jobs from the two chains onto this timeline. Unfortunately, this includes cases where the last task in this arrangement is from the chain $[s]$, which means that the preceding configuration was $[s' + 1, 0]$, which should be considered separately. To make sure that we are not counting configurations more than once, we want the last task finished in this arrangement to be the last one of chain $[t]$. This is why we consider the following:

1. Calculate the number of possible ways to arrive at the configuration $[s', 1]$ for some $s' \in \{1, \dots, s\}$. Then, the desired configuration $[s', 0]$ is only one step away, and is reached with a probability of $\frac{1}{2}$ from $[s', 1]$. Using this fact, and the arguments we stated above, we have

²Usually, these two configurations would not differ as we said that the chains are ordered according to their heights. But for considering only two chains, we will make that distinction.

³In the following, we will denote a configuration only by its corresponding precedence constraints. In this case, there is only one possible way to schedule the tasks according to *HLF*, so we do not declare the task which is already in process in that configuration. And because of the memorylessness of the exponential distribution it does not matter either, as we argued before.

that the number of possible ways to reach $[s', 1]$ are $\binom{n-s'-1}{t-1}$, and the probability of reaching it is $2^{-(n-s'-1)}$. Thus,

$$\mathcal{P}([s', 0] \text{ is the first configuration with only one chain left}) = \binom{n-s'-1}{t-1} \cdot 2^{-(n-s')}.$$

The expected processing time until arriving at that configuration and from there until the end of the schedule are then given by $\frac{1}{2}(n-s') + s'$.

2. Similarly, we do these calculations for the case where we reach the configuration $[0, t']$ for $t' \in \{1, \dots, t\}$.

In total, we have for the expected makespan of $[s, t]$:

$$C_c([s, t]) = \sum_{s'=1}^s 2^{-(n-s')} \binom{n-s'-1}{t-1} \left(\frac{1}{2}(n-s') + s' \right) + \sum_{t'=1}^t 2^{-(n-t')} \binom{n-t'-1}{s-1} \left(\frac{1}{2}(n-t') + t' \right).$$

Simplifying this expression, we obtain the following result.

Theorem 43. *Given a set \mathcal{T} of n tasks with exponential processing times with mean 1, and precedence constraints $[s, t]$ with $s + t = n$, then the optimal expected makespan is*

$$C_c([s, t]) = 2^{-(n+1)} \left(\sum_{s'=1}^s 2^{s'} \binom{n-s'-1}{t-1} (n+s') + \sum_{t'=1}^t 2^{t'} \binom{n-t'-1}{s-1} (n+t') \right). \quad (18.5)$$

Proof. We have already shown that *HLF* is the optimal scheduling strategy in Chapter 17. Adding a sink node to the two chains as seen in Fig. 3 on page 10, results in an intree that is optimally scheduled by *HLF* with an expected makespan equal to 1 plus the expected makespan of the two chains. Thus, *HLF* is optimal for inforests, and in particular, for chains, as well. By the arguments above, the equation in (18.5) correctly captures the essence of the configuration tree. The only thing left to discuss is the case $[1, 1]$, because this configuration is counted twice in the above approach, once for $[s', 1]$ with $s' = 1$, and once for $[1, t']$ with $t' = 1$. However, $[1, 1]$ has the two possible successor configurations $[1, 0]$ and $[0, 1]$, which are considered each with a probability of $\frac{1}{2}$. This means that because $[1, 1]$ is the only configuration where we can reach two of those desired configurations $[s', 0]$ or $[0, t']$, respectively, we have to count it twice. \square

The calculation for some example can be seen in Calculation 1 in Appendix A.

Using **Mathematica**, we can write this in an even more succinct way without the use of sums, i.e.,

$$C_c([s, t]) = 2^{-n}(n+1) \left(\binom{n+2}{t-1} {}_3F_2 \left[\begin{matrix} 1, 1-s, n+2 \\ 2-n, n+1 \end{matrix}; 2 \right] + \binom{n-2}{s-1} {}_3F_2 \left[\begin{matrix} 1, 1-t, n+2 \\ 2-n, n+1 \end{matrix}; 2 \right] \right), \quad (18.6)$$

where we used the code from Listing 18.1 to obtain this result.

```
1 2^(-s-t-1)*(Sum[2^i*Binomial[s+t-ss-1,t-1]*(s+t+ss), {ss,1,s}] +
Sum[2^j*Binomial[s+t-tt-1,s-1]*(s+t+tt), {tt,1,t}])
```

Listing 18.1: **Mathematica** code for (18.5). The execution yields the expression in (18.6).

However, (18.6) contains hypergeometric functions, see Chapter 11 on page 23, and thus it can be argued that this formula is not really more succinct than than one in (18.5), because hypergeometric functions are such powerful tools and contain so much information in just a few symbols. Nevertheless, for people who are more comfortable with the theory and application of hypergeometric functions, this may seem more elegant.

One interesting special case is if $s = t$, then the whole equation (18.5) can be simplified even further to obtain

$$C_c([s, s]) = \sum_{k=1}^s 2^{-(n-k)} \binom{n-k-1}{s-1} (n+k). \quad (18.7)$$

We can even have a closed form of (18.7) without a summation and without hypergeometric functions:

$$C_c([s, s]) = s + 2^{-(2s-1)} (2s-1) \binom{2s-2}{s-1}, \quad (18.8)$$

where we have used the code from Listing 18.2 to get this identity.

```
1 Simplify[Sum[2^(-2*s+k)*Binomial[2*s-k-1,s-1]*(2*s+k), {k,1,s}]]
```

Listing 18.2: **Mathematica** code for (18.7), resulting in (18.8).

18.6.2 Using Generating Functions to Find a Closed Form

A generating function is a power series of the form $\sum_{i \geq 0} a_i x^i$, see Chapter 10 on page 21, and for many specific functions, this series converges (under some specific circumstances). If that happens, the corresponding limit encodes the behavior of the sequence $(a_i : i \geq 0)$. We consider the multivariate power series

$$\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j,$$

with the sequence $(a_{ij} : i, j \geq 0)$ denoting the expected makespan of a given precedence graph $[i, j]$. If the limit has a closed form, we try to formulate it in such a way that it resembles an already known generating function, so that we can make some statements about the sequence (a_{ij}) . The expectation from this approach is that it might be generalized to three or more chains in a straightforward way.

The recursive formula of the sequence in our case is given as follows:

$$\begin{aligned} a_{00} &= 0, \quad a_{10} = a_{01} = 1, \quad a_{i0} = i, \quad a_{0j} = j, \\ a_{ij} &= \frac{1}{2} + \frac{1}{2} (a_{i-1,j} + a_{i,j-1}), \quad i, j \geq 1. \end{aligned} \quad (18.9)$$

Using the base cases, we can calculate the a_{ij} s using dynamic programming with the lattice seen in Fig. 19. Consider the value a_{ij} in the black square. Then, this can be calculated using only the two values in the gray squares.

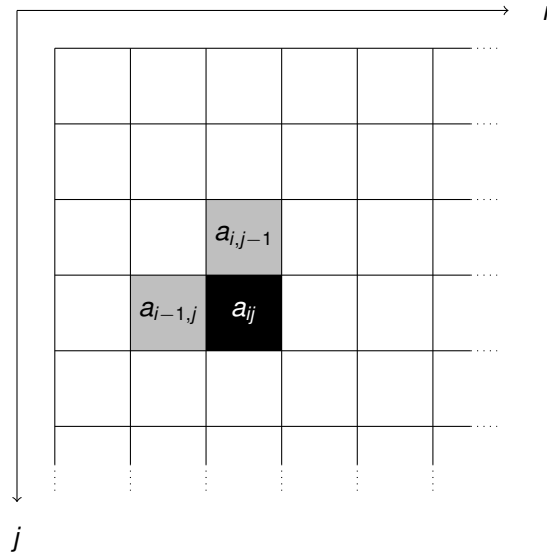


Fig. 19: The lattice for the computation of a_{ij} via dynamic programming.

Before we solve the recurrence to get a non-recursive expression for the a_{ij} s, we need a helpful result.

Lemma 44. *The following equality holds:*

$$\sum_{i \geq 0} \sum_{j \geq 0} \binom{i}{j} \mathbb{1}_{\{j \leq i\}} \left(\frac{1}{2}\right)^i x^{i-j} y^j = \sum_{i \geq 0} \sum_{j \geq 0} \left(\frac{1}{2}\right)^{i+j} \binom{i+j}{j} x^i y^j,$$

where $\mathbb{1}_{j \leq i}$ is the indicator function of the set $\{j \leq i\}$, i.e.,

$$\mathbb{1}_{j \leq i} = \begin{cases} 1 & j \leq i \\ 0 & j > i. \end{cases}$$

Proof. Consider the sum on the left and focus on all the terms with $x^{i^*} y^{j^*}$. These are exactly these with $j = j^*$ and $i - j^* = i^*$, i.e., $i = i^* + j^*$. Thus, we can rewrite the separate summands in this manner, and obtain the sum on the right. The indicator function vanishes because on the sum on the left, there can never be negative exponents. \square

As a notational shorthand, we define the values

$$F_{*0} = 0 + x + 2x^2 + 3x^3 + \dots = \sum_{i \geq 0} ix^i = \frac{x}{(1-x)^2},$$

$$F_{0*} = 0 + y + 2y^2 + 3y^3 + \dots = \sum_{j \geq 0} jy^j = \frac{y}{(1-y)^2},$$

where the difference in both lies in the choice of the variable. These identities are taken from [29]. F_{*0} is the one-dimensional power series over the first variable x whereas F_{0*} has the variable y . We note that the first summands are actually 0, which means that $F_{*0} = \sum_{i \geq 1} ix^i$ and $F_{0*} = \sum_{j \geq 1} jy^j$. With this, we can define our power series

$$F = F(x, y) = \sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j$$

and plug in the known relations from above to get

$$\begin{aligned} F &= \sum_{i \geq 1} \sum_{j \geq 1} a_{ij} x^i y^j + \sum_{j \geq 1} a_{0j} y^j + \sum_{i \geq 1} a_{i0} x^i + \underbrace{a_{00}}_{=0} \\ &= \sum_{i \geq 1} \sum_{j \geq 1} \left(\frac{1}{2} + \frac{1}{2} (a_{i-1,j} + a_{i,j-1}) \right) x^i y^j + F_{0*} + F_{*0} \\ &= \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} x^i y^j + \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} a_{i-1,j} x^i y^j + \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} a_{i,j-1} x^i y^j + F_{0*} + F_{*0} \\ &= \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} x^i y^j + \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 1} a_{i,j} x^{i+1} y^j + \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 0} a_{i,j} x^i y^{j+1} + F_{0*} + F_{*0} \\ &= \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} x^i y^j + \frac{1}{2} x \left(\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j - \sum_{i \geq 0} a_{i0} x^i \right) \\ &\quad + \frac{1}{2} y \left(\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j - \sum_{j \geq 0} a_{0j} y^j \right) + F_{0*} + F_{*0} \\ &= \frac{1}{2} \sum_{i \geq 1} \sum_{j \geq 1} x^i y^j + \frac{1}{2} x \cdot F - \frac{1}{2} x \cdot F_{*0} + \frac{1}{2} y \cdot F - \frac{1}{2} y \cdot F_{0*} + F_{0*} + F_{*0} \\ &= \frac{1}{2} \left(\sum_{i \geq 0} \sum_{j \geq 0} x^i y^j - \sum_{i \geq 0} x^i - \sum_{j \geq 0} y^j + 1 \right) + \frac{1}{2} x \cdot F + \left(1 - \frac{1}{2} x \right) F_{*0} + \frac{1}{2} y \cdot F - \left(1 - \frac{1}{2} y \right) F_{0*} \\ &= \left(\frac{1}{2} x + \frac{1}{2} y \right) F + \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j - \frac{1}{2} \sum_{i \geq 0} x^i - \frac{1}{2} \sum_{j \geq 0} y^j \\ &\quad + \left(1 - \frac{1}{2} x \right) F_{*0} + \left(1 - \frac{1}{2} y \right) F_{0*} + \frac{1}{2}. \end{aligned} \tag{18.10}$$

At this point, we have two directions to move in. The first is to use the formulas

$$\sum_{i \geq 0} x^i = \frac{1}{1-x} \quad \text{and} \quad \sum_{i \geq 1} \sum_{j \geq 1} x^i y^j = \frac{x}{1-x} \cdot \frac{y}{1-y},$$

cp. (9.3) from page 19, to obtain a more succinct and more readable expression for F as a rational function in the variables x and y . The second direction is about letting the sums be sums and using the convolution results from Lemma 22 and Lemma 23 (page 22) to obtain an expression for F as a sum where we can see the (non-recursive) description of the a_{ij} s. What we do in the following is to go in the second direction and come up with an expression for the a_{ij} s without any recursive parts. Then, we apply the methods of the second direction to the result of the first to obtain a succinct formula for the generating function of the sequence $(a_{ij} : i, j \geq 0)$. The second approach yields

$$F = \frac{1}{1 - \frac{1}{2}x - \frac{1}{2}y} \cdot \left(\frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j - \frac{1}{2} \sum_{i \geq 0} x^i - \frac{1}{2} \sum_{j \geq 0} y^j \right. \\ \left. + \left(1 - \frac{1}{2}x\right) \sum_{i \geq 0} ix^i + \left(1 - \frac{1}{2}y\right) \sum_{j \geq 0} jy^j + \frac{1}{2} \right).$$

We define the notations

$$A = \frac{1}{1 - \frac{1}{2}x - \frac{1}{2}y}, \\ B = \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j - \frac{1}{2} \sum_{i \geq 0} x^i - \frac{1}{2} \sum_{j \geq 0} y^j + \left(1 - \frac{1}{2}x\right) \sum_{i \geq 0} ix^i + \left(1 - \frac{1}{2}y\right) \sum_{j \geq 0} jy^j + \frac{1}{2}.$$

and have a look at the factors A and B separately. First, for B , we get

$$B = \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j - \frac{1}{2} \sum_{i \geq 0} x^i - \frac{1}{2} \sum_{j \geq 0} y^j + \left(1 - \frac{1}{2}x\right) \sum_{i \geq 0} ix^i + \left(1 - \frac{1}{2}y\right) \sum_{j \geq 0} jy^j + \frac{1}{2} \\ = \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j + \underbrace{\sum_{i \geq 0} \left(i - \frac{1}{2}\right) x^i - \frac{1}{2} \sum_{i \geq 0} ix^{i+1}}_{= \frac{1}{2} \sum_{i \geq 0} ix^i - \frac{1}{2}} + \underbrace{\sum_{j \geq 0} \left(j - \frac{1}{2}\right) y^j - \frac{1}{2} \sum_{j \geq 0} jy^{j+1}}_{= \frac{1}{2} \sum_{j \geq 0} jy^j - \frac{1}{2}} + \frac{1}{2} \\ = \frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j + \frac{1}{2} \sum_{i \geq 0} ix^i + \frac{1}{2} \sum_{j \geq 0} jy^j - \frac{1}{2}$$

As for A , we have

$$\begin{aligned}
A &= \frac{1}{1 - \frac{1}{2}x - \frac{1}{2}y} \\
&= \sum_{i \geq 0} \left(\frac{1}{2}x + \frac{1}{2}y \right)^i \\
&= \sum_{i \geq 0} \sum_{j=0}^i \binom{i}{j} \left(\frac{1}{2}x \right)^{i-j} \left(\frac{1}{2}y \right)^j \\
&= \sum_{i \geq 0} \sum_{j \geq 0} \mathbb{1}_{\{j \leq i\}} \binom{i}{j} \left(\frac{1}{2} \right)^i x^{i-j} y^j, \\
&= \sum_{i \geq 0} \sum_{j \geq 0} \left(\frac{1}{2} \right)^{i+j} \binom{i+j}{j} x^i y^j, \tag{18.11}
\end{aligned}$$

where, for the second-to-last equality, we use that $\binom{i}{j} = 0$ for $j > i$, and the corresponding indicator function $\mathbb{1}_{\{j \leq i\}}$, i.e., it is 1 if $j \leq i$, and 0 if $j > i$. For the last equality, we used Lemma 44 to get rid of the difference in the exponent, as well as the indicator function.

In total, this yields

$$\begin{aligned}
F &= A \cdot B \\
&= \left(\sum_{i \geq 0} \sum_{j \geq 0} \left(\frac{1}{2} \right)^{i+j} \binom{i+j}{j} x^i y^j \right) \cdot \left(\frac{1}{2} \sum_{i \geq 0} \sum_{j \geq 0} x^i y^j + \sum_{i \geq 0} i x^i + \sum_{j \geq 0} j y^j - \frac{1}{2} \right).
\end{aligned}$$

The separate products of the sums can be solved by convolution, see Lemma 23. Thus, with a little more simplifying, we obtain a formula

$$F = \sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j$$

and the following theorem.

Theorem 45. *The expected makespan of two chains of lengths i and j , respectively, i.e., a configuration $[i, j]$, is*

$$\begin{aligned}
a_{ij} &= \sum_{k=0}^i \sum_{\ell=0}^j \left(\frac{1}{2} \right)^{k+\ell+1} \binom{k+\ell}{\ell} \\
&\quad + \sum_{k=0}^i \left(\frac{1}{2} \right)^{k+j+1} \binom{k+j}{j} (i-k) \\
&\quad + \sum_{\ell=0}^j \left(\frac{1}{2} \right)^{i+\ell+1} \binom{i+\ell}{\ell} (j-\ell) \\
&\quad - \left(\frac{1}{2} \right)^{i+j+1} \binom{i+j}{j}. \tag{18.12}
\end{aligned}$$

(18.12) gives us a way to calculate the optimal expected makespan of a configuration $[i, j]$ without even simulating a schedule in the same way that (18.5) does this. Calculation 1 in Appendix A on page 181 shows the equivalence to the other approaches from (18.5) and Lemma 37 with a small example.

Now that (18.12) gives us a nicer expression for the sequence $(a_{ij} : i, j \geq 0)$, we can use these sums to find another succinct expression for the generating function of this sequence, i.e., the first direction which we described above. To this end, we have

$$\sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{k=0}^i \sum_{\ell=0}^j \left(\frac{1}{2} \right)^{k+\ell+1} \binom{k+\ell}{\ell} \right) x^i y^j = \left(\sum_{i \geq 0} \sum_{j \geq 0} b_{ij} x^i y^j \right) \left(\sum_{i \geq 0} \sum_{j \geq 0} c_{ij} x^i y^j \right)$$

by Lemma 23 with $b_{ij} = \left(\frac{1}{2} \right)^{i+j+1} \binom{i+j}{j}$ and $c_{ij} = 1$. Thus, this is equivalent to

$$\begin{aligned} \sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{k=0}^i \sum_{\ell=0}^j \left(\frac{1}{2} \right)^{k+\ell+1} \binom{k+\ell}{\ell} \right) x^i y^j &= \frac{1}{2} \left(\sum_{i \geq 0} \sum_{j \geq 0} \left(\frac{1}{2} \right)^{i+j} \binom{i+j}{j} x^i y^j \right) \left(\sum_{i \geq 0} \sum_{j \geq 0} x^i y^j \right) \\ &= \frac{1}{2} \left(\sum_{i \geq 0} \sum_{j \geq 0} \binom{i+j}{j} \left(\frac{x}{2} \right)^i \left(\frac{y}{2} \right)^j \right) \left(\sum_{i \geq 0} \sum_{j \geq 0} x^i y^j \right) \\ &= \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{2}x - \frac{1}{2}y} \cdot \frac{1}{(1-x)(1-y)} \\ &= \frac{1}{2-x-y} \cdot \frac{1}{(1-x)(1-y)}, \end{aligned}$$

parts of which we have already seen in (18.11). Similarly, we can find expressions for the generating functions of the other three lines of the right hand side of (18.12). For the second line, for example, we have, again with the result about convolution of Lemma 23,

$$\begin{aligned} \sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{k=0}^i \underbrace{\left(\frac{1}{2} \right)^{k+j+1} \binom{k+j}{j}}_{=b_{kj}} \underbrace{(j-k)}_{=c_{i-k}} \right) x^i y^j &= \left(\sum_{i \geq 0} \sum_{j \geq 0} b_{ij} x^i y^j \right) \left(\sum_{i \geq 0} c_i x^i \right) \\ &= \frac{1}{2} \left(\sum_{i \geq 0} \sum_{j \geq 0} \binom{i+j}{j} \left(\frac{x}{2} \right)^i \left(\frac{y}{2} \right)^j \right) \left(\sum_{i \geq 0} i x^i \right) \\ &= \frac{1}{2-x-y} \cdot \frac{x}{(1-x)^2}. \end{aligned}$$

For the remaining two lines, we have

$$\sum_{i \geq 0} \sum_{j \geq 0} \left(\sum_{\ell=0}^j \left(\frac{1}{2} \right)^{i+j+1} \binom{i+\ell}{\ell} (j-\ell) \right) x^i y^j = \frac{1}{2-x-y} \cdot \frac{y}{(1-y)^2}$$

and

$$\sum_{i \geq 0} \sum_{j \geq 0} \left(\left(\frac{1}{2} \right)^{i+j+1} \binom{i+j}{j} \right) x^i y^j = \frac{1}{2-x-y}.$$

Altogether, this amounts to

$$\sum_{i \geq 0} \sum_{j \geq 0} a_{ij} x^i y^j = \frac{1}{2 - x - y} \left(\frac{1}{(1 - x)(1 - y)} + \frac{x}{(1 - x)^2} + \frac{y}{(1 - y)^2} - 1 \right). \quad (18.13)$$

Our efforts to simplify (18.13) were not successful. Both **Mathematica** and **Maple**, as well as several results and tools from the theory of generating functions did not yield a simplified form of (18.13). We used methods from [54] or [34] for example, but unfortunately, no simpler equivalent formula was found.

18.7 *k* Chains

We generalize the situation from the previous section and consider more than two chains. This means that now we are given a set \mathcal{T} of n tasks corresponding to the precedence constraints $[l_1, \dots, l_k]$, i.e., k different chains with lengths l_1, \dots, l_k . Without loss of generality, we may assume that $l_1 \geq \dots \geq l_k$. And because *HLF* is the optimal scheduling strategy for inforests, this means that tasks are scheduled from left to right.

For the special case with $k = 2$ we derived a closed formula for the expected makespan that can be computed without running a schedule. So, the objective for this section is to derive such a formula for the general case as well.

18.7.1 Using the Combinatorial Approach

In Theorem 43, we used methods from combinatorics to determine the expected makespan of a two-chain-configuration $[s, t]$. The binomial coefficients essentially capture the number of different configuration paths a schedule may take to arrive at a certain configuration. When only two chains are present then exactly one task of each chain will be scheduled at any time (until one of the chains is processed completely). As every two-processor scheduling strategy on two chains is *HLF*, all possible orders in which the tasks are processed are possible without contradicting the “highest-level first” paradigm.

Now consider three chains. There is a generalization of the binomial coefficients, aptly called the *multinomial coefficients*. These can be defined by

$$\binom{n}{l_1, \dots, l_k} = \frac{n!}{l_1! \cdots l_k!}, \quad (18.14)$$

where $l_1 + \dots + l_k = n$, and can be interpreted as the number of possible combinations of how to put n distinct objects into k distinct bins (with l_1 objects into the first bin, l_2 objects into the second, and so on). The interpretation that we want to use here is to pick tasks (objects) from the k chains (bins) to be processed on the two processors. Now, the order of the tasks being picked from one particular chain does not matter, because we can only pick the topmost task anyway. However, the order of the tasks from different chains really is important.

Consider the graph $[2, 1, 1]$, and the desired configuration $[1, 0, 0]$. Then the multinomial coefficient

$$\binom{3}{1, 1, 1} = 6$$

gives the number of possibilities a schedule can process tasks from these three chains to arrive at $[1, 0, 0]$, i.e., one task from each of the three chains. However, two of those possibilities are the ones where the tasks from the second and third chains are processed before the topmost task from the first chain, see Fig. 20. But this is clearly against the paradigm of *HLF*, making the multinomial coefficient useless in this form.

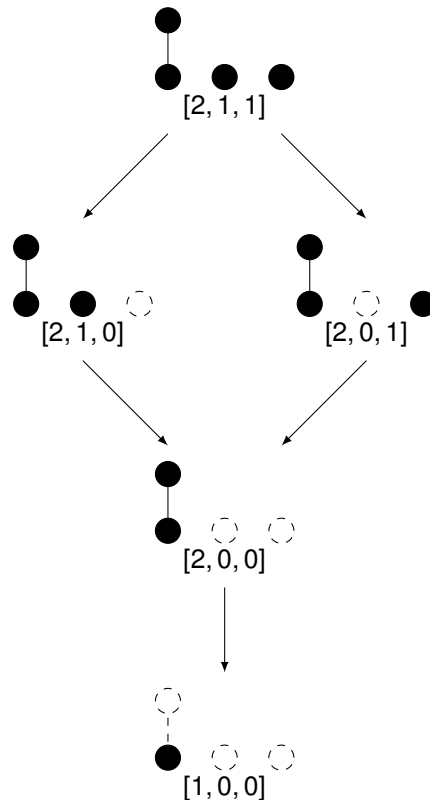


Fig. 20: Two configuration paths that lead from $[2, 1, 1]$ to $[1, 0, 0]$.

Exceptions from this are the cases where we have exactly as many chains as we have processors available. If we had three processors for our three chains as in the example, then again, every scheduling strategy would be *HLF*. The same holds for k chains and k processors. Thus, the number of paths that lead to a certain configuration can be calculated using the multinomial coefficients. Still, the formula does not look that easy, because we have to adjust the factors in it as soon as one chain is fully processed. As long as there are k chains to choose from, the expected time until the next tasks finishes is exactly $\frac{1}{k}$ and the probability that a task of a

particular chain will be finished next is again $\frac{1}{k}$. After one chain is completely processed, these values reduce to $\frac{1}{k-1}$. This means that we would have a formula for k chains, which recursively uses every possible/feasible formula for $k - 1$ chains.

For example, consider $[i, j, k]$. As in the case with two chains and two processors, see Theorem 43, we focus on the configurations $[r, s, 1]$, $[r, 1, t]$, and $[1, s, t]$, i.e., configurations where one chain is almost completely processed and the schedule only needs $\frac{1}{3}$ time units longer (with a probability of $\frac{1}{3}$) to arrive at a configuration with only two chains remaining. The expected makespan for $[i, j, k]$ is

$$\begin{aligned}
C_c([i, j, k]) &= \sum_{r=1}^i \sum_{s=1}^j \left(\underbrace{\left(\frac{1}{3} \cdot \left(\frac{1}{3} \right)^{i-r+j-s+k-1} \cdot \binom{i-r+j-s+k-1}{i-r, j-s, k-1} \right)}_{\text{prob. to arrive at } [r, s, 0] \text{ via the config. } [r, s, 1]} \cdot \left(\frac{1}{3}(i-r+j-s+k-1) + \frac{1}{3} + C_c([r, s]) \right) \right) \\
&\quad + \sum_{r=1}^i \sum_{t=1}^k \left(\underbrace{\left(\frac{1}{3} \cdot \left(\frac{1}{3} \right)^{i-r+j-1+k-t} \cdot \binom{i-r+j-1+k-t}{i-r, j-1, k-t} \right)}_{\text{prob. to arrive at } [r, 0, t] \text{ via the config. } [r, 1, t]} \cdot \left(\frac{1}{3}(i-r+j-1+k-t) + \frac{1}{3} + C_c([r, t]) \right) \right) \\
&\quad + \sum_{s=1}^j \sum_{t=1}^k \left(\underbrace{\left(\frac{1}{3} \cdot \left(\frac{1}{3} \right)^{i-1+j-s+k-t} \cdot \binom{i-1+j-s+k-t}{i-1, j-s, k-t} \right)}_{\text{prob. to arrive at } [0, s, t] \text{ via the config. } [1, s, t]} \cdot \left(\frac{1}{3}(i-1+j-s+k-t) + \frac{1}{3} + C_c([s, t]) \right) \right). \tag{18.15}
\end{aligned}$$

The case with the configuration $[1, 1, 1]$ is counted multiple times, namely three times in this case. But because its three successor configurations $[1, 1, 0]$, $[1, 0, 1]$, and $[0, 1, 1]$ are all configurations with only two chains (for the first time), we have to count it three times.

In order to generalize (18.15) to k chains and k processors, we have to define some abbreviations in our notation so that we can derive a rather succinct expression. To this end, let α be a *multiindex* with k entries, i.e., $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ (see [48] for example). We define the size of a multiindex by

$$|\alpha| = \alpha_1 + \dots + \alpha_k.$$

Furthermore, let A_i be a set of multiindices α with $1 \leq \alpha_i \leq l_i$ for all $1 \leq i \leq k$ and at least one entry equal to 1, i.e.,

$$A_i = \{\alpha = (\alpha_1, \dots, \alpha_k) : \forall i \in \{1, \dots, k\} : 1 \leq \alpha_i \leq l_i \text{ and } \exists i : \alpha_i = 1\},$$

and define the multinomial coefficient, see (18.14), by

$$\binom{n-|\alpha|}{\alpha} = \binom{n-|\alpha|}{\alpha_1, \dots, \alpha_k}.$$

Then, the expected makespan of a configuration $[l_1, \dots, l_k]$ for the k -processor problem is

$$C_c([l_1, \dots, l_k]) = \sum_{\alpha \in A_1} \left(\frac{1}{k}\right)^{n-|\alpha|} \cdot \binom{n-|\alpha|}{\alpha} \cdot \frac{1}{k} \cdot \left(\frac{n-|\alpha|}{k} + \frac{1}{k} + C_c([\alpha])\right). \quad (18.16)$$

Here, $[\alpha]$ just denotes the configuration $[\alpha_1, \dots, \alpha_k]$ (after appropriately sorting the chains). The interpretation for (18.16) is the same as in (18.5), or later in Theorem 49: the first two factors describe the probability of arriving at a given configuration, the third factor is the probability that one chain vanishes at the next decision point, and the summands in the last factor describe the expected processing time so far, the expected time until the next decision point, and the expected remaining processing time after the next decision point, respectively.

We end the brief excursion to k processors and focus on the case with two processors again. But still, the combinatorial approach is the one we want to use. We argued that the multinomial coefficients will produce wrong results in the general case when the number of chains is not equal to the number of available processors. However, with a recursive approach as in (16.1) on page 37 applied repeatedly, we can use the binomial coefficient very well. For a configuration $[l_1, l_2, l_3, \dots]$ with $l_1 \geq l_2 \geq l_3 \geq \dots$, we know that (in an optimal schedule) only tasks from the first two chains are processed until one of those chains has less than l_3 tasks. To this end, we consider the following two cases.

Case 1: The first (leftmost) chain is processed until it has exactly l_3 tasks left while the second chain has j tasks left, with $l_3 \leq j \leq l_2$. Then the number of possible configuration paths to get there is

$$\binom{l_1 - l_3 + l_2 - j}{l_1 - l_3}.$$

The probability for any one of these paths is

$$\left(\frac{1}{2}\right)^{l_1 - l_3 + l_2 - j}.$$

The expected time that has passed while processing any one of these paths is

$$\frac{l_1 - l_3 + l_2 - j}{2}.$$

Then, we need one more step, with a probability of $1/2$, to get to a configuration where the first chain is lower than the third. This whole case (or cases as j is a variable) amounts to the sum in (18.17).

Case 2: The second chain is processed until it has exactly l_3 tasks left while the first chain has i tasks left, with $l_3 \leq i \leq l_1$. Then, analogously to the first case, we have

$$\binom{l_1 - i + l_2 - l_3}{l_2 - l_3}$$

possible configurations paths, each with probability

$$\left(\frac{1}{2}\right)^{l_1 - i + l_2 - l_3}$$

and expected time

$$\frac{l_1 - i + l_2 - l_3}{2}.$$

Add one more step, then the second chain is lower than the third. This amounts to the sum in (18.18).

Hence, the expression for the expected makespan is

$$C_c([l_1, l_2, l_3, \dots, 1]) = \sum_{j=l_3}^{l_2} \binom{l_1 - l_2 + l_2 - j}{l_1 - l_3} \cdot \left(\frac{1}{2}\right)^{l_1 - l_3 + l_2 - j} \cdot \frac{1}{2} \cdot \left(\frac{l_1 - l_3 + l_2 - j}{2} + \frac{1}{2} + C_c([l_3 - 1, j, l_3, \dots, 1])\right) \quad (18.17)$$

$$+ \sum_{i=l_3}^{l_1} \binom{l_1 - i + l_2 - l_3}{l_2 - l_3} \cdot \left(\frac{1}{2}\right)^{l_1 - i + l_2 - l_3} \cdot \frac{1}{2} \cdot \left(\frac{l_1 - i + l_2 - l_3}{2} + \frac{1}{2} + C_c([i, l_3 - 1, l_3, \dots, 1])\right). \quad (18.18)$$

Note that the configurations in the recursion are not in order yet, e.g., in (18.17) the second chain with j tasks is now the highest chain, followed by the third chain and any other chain of height l_3 . Only then we have the first chain of height $l_3 - 1$ in the order. So, after rearranging the chains in the notation the recursion can go on.

Another approach which was quite fruitful in the special case with only two chains was using generating functions, see Section 18.6.2. The results of using these with k chains are presented in the next section.

18.7.2 Using Generating Functions for Specific Examples

For starters, we consider the easy example $[l_1 = i, 1, \dots, 1]$ with $j + 1$ chains in total, i.e., one high chain of height i and j small ones with only one task each. We use the notation

$$b_{ij} := C_c([i, \underbrace{1, \dots, 1}_j]).$$

This is denoted by ℓ_{ij} in [10]. Then, we can define a recurrence relation very similar to (18.9) to describe the recursive nature of b_{ij} :

$$b_{00} = 0, b_{10} = b_{01} = 1, b_{i0} = i, b_{0j} = \frac{j+1}{2},$$

$$b_{ij} = \frac{1}{2} + \frac{1}{2}(b_{i-1,j} + b_{i,j-1}), i, j \geq 1. \quad (18.19)$$

Because of the highly similar structure to (18.9) – actually, only the last base case is different – the end result is again highly similar to what we have calculated before. The intermediate

calculations are omitted here, and the result is

$$\begin{aligned}
b_{ij} &= \sum_{k=0}^i \sum_{\ell=0}^j \left(\frac{1}{2}\right)^{k+\ell+1} \binom{k+\ell}{\ell} \\
&\quad + \sum_{k=0}^i \left(\frac{1}{2}\right)^{k+j+1} \binom{k+j}{j} (i-k) \\
&\quad + \sum_{\ell=0}^j \left(\frac{1}{2}\right)^{i+\ell+2} \binom{i+\ell}{\ell} (j-\ell)
\end{aligned} \tag{18.20}$$

Notice that the exponent in the last sum has a “+2” instead of just a “+1”. This is because one of the base conditions is a ratio with denominator 2, cp. (18.19). Moreover, here we have no term without a summation, in contrast to the result with the a_{ij} s from (18.12).

A formula for a general case, however, would not look as nice and clean as (18.20), which is why we try another approach in the following section.

18.7.3 Using a Stair Function for the General Case

We want to reduce the general case to the special case with $k = 2$. This means that we have to calculate the probability that a configuration with only two remaining chains occurs, as well as the expected time until we reach such a configuration.

We derive expressions for the same approach we used in the case of two chains. There, for $[s, t]$, we considered configurations of the form $[s', 1]$, or $[1, t']$, resp., and calculated the corresponding probabilities as well as the expected makespans. Here, we consider configurations of the form $[l', 1, 1, 0, \dots, 0]$, i.e., configurations which are only one step away from having only two chains left. First, we show that these are exactly the kind of configurations we can focus on, and not more have to be considered. To this end we show the following lemma.

Lemma 46. *Given a sequence of chains $[l_1, \dots, l_k]$ with $k \geq 3$ and consider the first time a configuration with only two chains left occurs, i.e., a configuration of the form $[l', i', 0, \dots, 0]$. Then it holds that $i' = 1$.*

Proof. Assume that the first time only two chains are left is at $[l', i', 0, \dots, 0]$ with $l' \geq i' > 1$. Then the preceding configuration must have been $[l', i', 1, 0, \dots, 0]$. But as $l' > 1$ and $i' > 1$, the lone task from the third chain cannot be scheduled without contradicting *HLF*. \square

Again, the expected time that passes from the start of the schedule until a certain configuration is easy to determine, as long as there are always both processors busy during this time. For our cases this is true and we only have to take care of calculating the probability of arriving at a certain configuration. We do this by calculating the ratio of the number of possible paths to such a configuration and the number of all paths of that length. The length is easy to determine as it is just the number of tasks which have been finished processing until then, say ℓ . And the number of all paths of that length is $2^{-\ell}$ because we have two possible successor configurations for the two tasks that are being processed by *HLF*. In some cases, the two successor configurations

are identical as we do not distinguish between two tasks which are on the same level. We take this into account by counting these two cases separately.

Let us consider an example. Given a sequence of chains $[5, 5, 4, 2]$, we have the following grid of configurations, see Fig. 21.

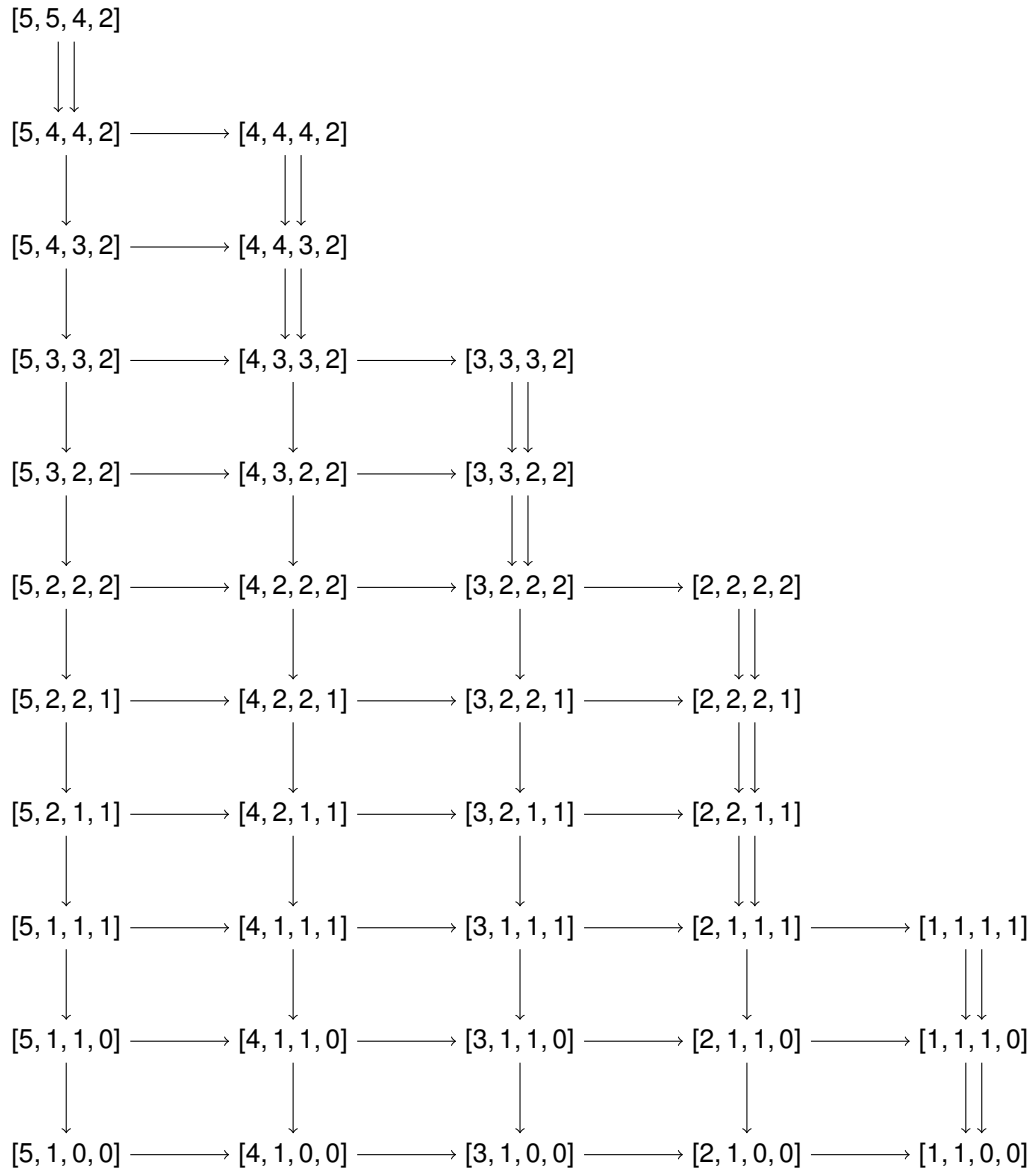


Fig. 21: The chain grid for $[5, 5, 4, 2]$.

The structure of the grid can tell us some information:

- The columns represent configurations with the same value for the highest chain. For example, the first column corresponds to configurations where the highest chain has

height 5, the second to height 4, and so on. In general, the first column would correspond to a highest chain of height l_1 , the second to height $l_1 - 1$, etc.

- The grid has steps in it. The height of the steps can be determined as well, they are exactly the number of tasks on the levels minus 1. This is because for more than one task on the highest level, HLF will have two identical configurations disregarding isomorphisms. The number of tasks on each level is known and can be determined with the lengths of the chains, namely these are the p_i values from the profile. For example, the height of the first step is the number of tasks on the highest level minus 1, i.e., $p_5 - 1 = 1$, the height of the second step is $p_4 - 1 = 2$, and so on.
- The width of each step is exactly 1.

For future figures we will not make the edges directed, we define edges to either go to the right or down. Also for simplicity, we will mostly consider configurations with $l_1 > l_2$ as starting configurations, because this is the first configuration with two different successor configurations. Now, the task is to find out how many paths there are reaching from the starting configuration $[5, 5, 4, 2]$ to a configuration in the second-to-last row, i.e., to a configuration that is only one step away from another with only two chains left. For this, we define the stair function.

Definition 47 (Stair function). The *stair function* $F_W^H(x_1, \dots, x_m)$ is defined as the number of possible paths in a grid with total width W , total height H , and x_1, \dots, x_m the number of double edges at each step of the grid, with $m \leq W$.

The total width W corresponds to the number of edges along the bottom row of the grid, the total height H corresponds to the number of edges along the leftmost column of the grid. The values x_1, \dots, x_{m-1} also denote the step heights, whereas x_m might be less than the height of the last step, i.e., the last step consists of double edges and single edges. See Fig. 22 for some examples. We omit the node labels, as we only care about the number of paths starting from the upper left corner and ending in the lower right corner of the grid. Nodes in the grids are drawn as gray rectangles, in order to avoid confusion with tasks in graphs (that are drawn as circles in black, gray, white, or other patterns).

Now, this number of paths can be described using a recurrence relation.

Lemma 48. *Given a grid with steps and double edges (e.g. as in Fig. 22) with total width W , total height H , and step sizes x_1, \dots, x_m , the number of paths from the upper left corner to the lower right corner is given by*

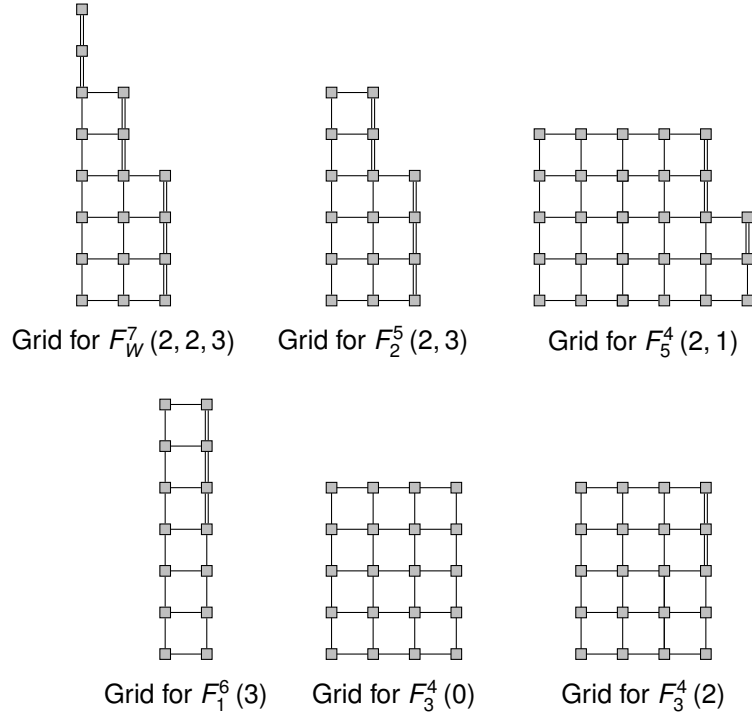


Fig. 22: Corresponding grids to some examples.

$$F_W^H(x_1, \dots, x_m) = F_W^{H-1}(x_1, \dots, x_m) + F_{W-1}^H(x_1, \dots, x_m) \text{ if } m = W, \quad (18.21)$$

$$F_W^H(x_1, \dots, x_m) = 2^{x_1} \cdot F_W^{H-x_1}(x_2, \dots, x_m) \text{ if } m = W + 1, \quad (18.22)$$

$$F_0^H(0) = 1, \quad (18.23)$$

$$F_1^H(0) = H + 1,$$

$$F_W^0(0) = 1,$$

$$F_W^H(0) = \binom{W+H}{H}. \quad (18.24)$$

Proof. We prove this by induction on n , the number of nodes in the grid. The trivial base cases, i.e., (18.23) to (18.24), are as follows: $F_0^H(0)$ corresponds to a grid which only consists of a single chain of height H (without any double edges), $F_1^H(0)$ corresponds to a $1 \times H$ -grid (without any double edges), $F_W^0(0)$ corresponds to a single row of width W , and $F_W^H(0)$ corresponds to a $W \times H$ -grid (without any double edges). See Fig. 23 for examples of the base cases.

Consider a grid with $n + 1$ nodes with width W , height H and steps x_1, \dots, x_m . The first line (18.21) is when there are the two possibilities to either go down or to the right. Then, the total number of ways to the bottom right node is the sum of the total number of ways from those two

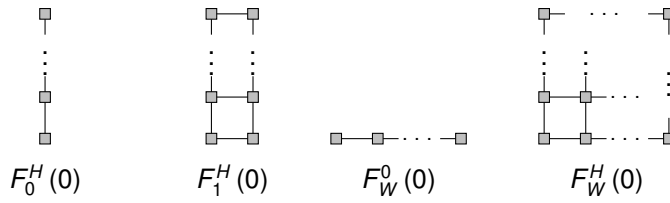


Fig. 23: The base cases of the recurrence relation

reachable nodes. The claim then follows by the induction hypothesis. The second line (18.22) captures a node where there are no possibilities to go to the right, only downwards. Then, there are no possibilities for choosing the way until a node is reached with a right neighbor. Such a node is reached after x_1 many steps, because this is exactly the step size. So, the number of possible ways contains a factor of 2^{x_1} , and the claim again follows by the induction hypothesis. Fig. 24 shows an example for each of those two equations. The thick red arrows indicate the two possibilities to continue the way. □

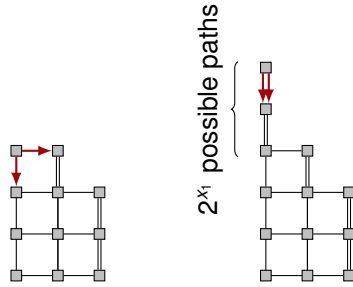


Fig. 24: Examples for (18.21) and (18.22) on the left and right, respectively.

Although there is a large amount of research already been done in the field of lattice (grid) paths [52, 53], like the discovery of Catalan numbers [1] and Delannoy numbers [2] for square-shaped lattices, or general formulas and number sequences for rectangular lattices [20], even with weighted edges [21], none of the works that we looked at relate to our cases with chain grids and double steps. In the special case that there are no double steps, we can use a slightly smaller formula than the one in Lemma 48, using only recursion and the binomial coefficient

$$\binom{W + H}{W},$$

which denotes the number of paths in an $W \times H$ -grid with only single steps. Getting back to the example [5, 5, 4, 2], we want to calculate the number of paths from the starting configuration to a configuration in the second-to-last row from Fig. 21. So we have to consider the number of paths from the configuration [5, 5, 4, 2] to the configurations [1, 1, 1, 0],

$[2, 1, 1, 0]$, $[3, 1, 1, 0]$, $[4, 1, 1, 0]$, and $[5, 1, 1, 0]$, which are

$$|\text{paths from } [5, 5, 4, 2] \text{ to } [1, 1, 1, 0]| = F_4^9(1, 2, 2, 3, 1) = 2^1 \cdot F_4^8(2, 2, 3, 1),$$

$$|\text{paths from } [5, 5, 4, 2] \text{ to } [2, 1, 1, 0]| = F_3^9(1, 2, 2, 3) = 2^1 \cdot F_3^8(2, 2, 3),$$

$$|\text{paths from } [5, 5, 4, 2] \text{ to } [3, 1, 1, 0]| = F_2^9(1, 2, 2) = 2^1 \cdot F_2^8(2, 2),$$

$$|\text{paths from } [5, 5, 4, 2] \text{ to } [4, 1, 1, 0]| = F_1^9(1, 2) = 2^1 \cdot F_1^8(2),$$

$$|\text{paths from } [5, 5, 4, 2] \text{ to } [5, 1, 1, 0]| = F_0^9(1) = 2^1 \cdot F_0^8(0).$$

Fig. 25 shows the grids to be considered (on the first level of the recursion at least) in the five equations above, i.e., the grids for $F_4^9(1, 2, 2, 3, 1)$ and $F_3^9(1, 2, 2, 3)$ (at the top, from left to right), and the grids for $F_2^9(1, 2, 2)$, $F_1^9(1, 2)$, and $F_0^9(1)$ (at the bottom, from left to right). The configurations which correspond to the start or the end of the paths are the top-left or the bottom-right nodes, respectively.

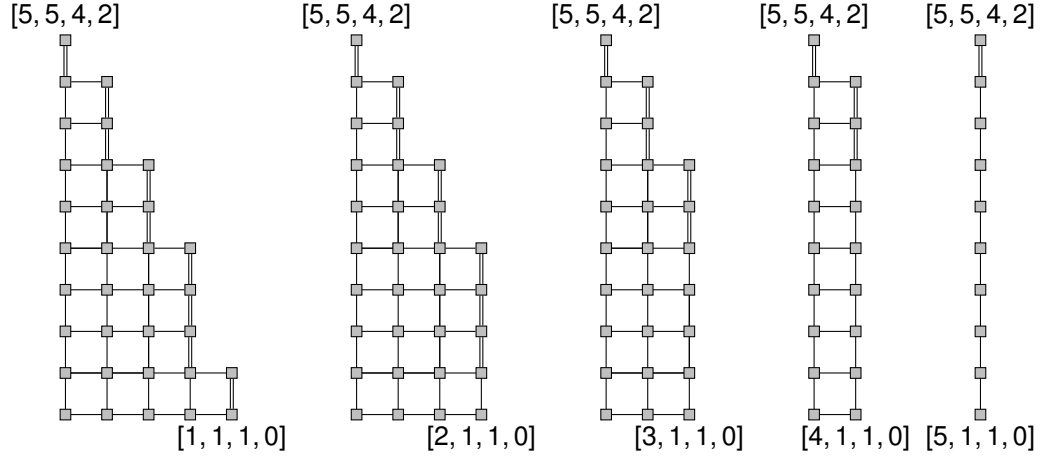


Fig. 25: The grids to be considered in the example from Fig. 21.

Using the above values we can calculate the expected makespan for $[5, 5, 4, 2]$.

$$C_c([5, 5, 4, 2]) = \sum_{i=1}^5 \frac{1}{2} \cdot |\text{paths from } [5, 5, 4, 2] \text{ to } [i, 1, 1, 0]| \cdot 2^{-((2-0)+(4-1)+(5-1)+(5-i))} \cdot \left(\frac{1}{2}((2-0) + (4-1) + (5-1) + (5-i) + 1) + C_c([i, 1]) \right),$$

where we can determine $C_c([i, 1])$ by using the formula developed for two chains, which simplifies to $C_c([i, 1]) = i + 2^{-i}$.

For the general case with k chains $[l_1, \dots, l_k]$, we use some definitions to shorten the formula and for a better understandability. First, we have $L = l_1$. Then, define

$$x_i = p_{L-i} - 1, \text{ for } i \in \{0, \dots, L-2\},$$

$$x_{L-1} = p_1 - 2,$$

while using the values from the profile $p = (p_1, \dots, p_L)$. These can be easily determined from the l_i values by

$$p_i = \max\{j : l_j \geq i\}, \quad i \in \{1, \dots, L = l_1\}.$$

In addition, define

$$\begin{aligned} P_i &= (l_1 - (i - 1)) + (l_2 - 1) + (l_3 - 1) + l_4 + \dots + l_k \\ &= l_1 + \dots + l_k - i - 1, \end{aligned}$$

i.e., the length of a path from the starting configuration to the configuration $[i, 1, 1, 0, \dots, 0]$, and

$$H = (l_2 - 1) + (l_3 - 1) + l_4 + \dots + l_k = \sum_{j=2}^k l_j - 2$$

for $k > 2$, and $H = l_2 - 1$ for $k = 2$. Then, the formula for the expected makespan is given by

$$\begin{aligned} C_c([l_1, \dots, l_k]) &= \sum_{i=1}^L \underbrace{2^{x_0} \cdot F_{L-i}^H(x_1, \dots, x_{L-i})}_{\text{no. of paths to } [i, 1, 1, 0, \dots, 0]} \cdot \underbrace{2^{-P_i}}_{\text{pr. of one path}} \\ &\quad \cdot \underbrace{\frac{1}{2}}_{\text{pr. to next conf.}} \cdot \underbrace{\left(\frac{1}{2}(P_i + 1) + C_c([i, 1])\right)}_{\text{expected processing time}}. \end{aligned}$$

When simplified, this leads to the following result.

Theorem 49. *Given a set \mathcal{T} of n tasks as k chains $[l_1, \dots, l_k]$ with exponential processing times, the optimal expected makespan is given by*

$$C_c([l_1, \dots, l_k]) = \sum_{i=1}^L 2^{x_0 - P_i - 1} \cdot F_{L-i}^H(x_1, \dots, x_{L-i}) \cdot \left(\frac{1}{2}(P_i + 1) + i + 2^{-i}\right),$$

with $L = l_1$,

$$\begin{aligned} x_i &= p_{L-i} - 1, \quad i \in \{0, \dots, L - 2\}, \\ x_{L-1} &= p_1 - 2, \\ P_i &= \sum_{j=1}^k l_j - (i - 1), \quad i \in \{1, \dots, L\}, \\ H &= \begin{cases} \sum_{j=2}^k l_j - 2 & k > 2 \\ l_2 - 1 & k = 2, \end{cases} \end{aligned}$$

and the recurrence relation F as defined in Lemma 48.

Alternatively, one could make a distinction between the different chains, i.e., not order them by decreasing height in every step. This way, it is easier to count the possibilities, because there

are no configurations which have to be counted twice or even more times. For example, with four chains, there are four possible configurations with only one task left if the chains are not ordered, namely $[1, 0, 0, 0]$, $[0, 1, 0, 0]$, $[0, 0, 1, 0]$, and $[0, 0, 0, 1]$. When we order the chains by decreasing height, then only $[1, 0, 0, 0]$ remains, but may have to be counted several times now. See Fig. 26 for the configuration graph of the example configuration $[4, 4, 3, 2]$. The arcs relate to the (up to two) successor configurations, the labels on the arcs indicate the index of the chain of the last task that was finished. They are not important for the calculation, just for a better understanding. In the end, we want to calculate the number of paths from the starting configuration (the topmost one) to one of the blue configurations. These are all of the form that one chain has only one task left, another chain has at least one task left, and all the other chains do not have any tasks left.

The upside of this approach is that it is easy to calculate because all it takes is to build the graph of the transitions between the configurations. This can be done efficiently, because considering some configuration, and implicitly using the level-oriented *HLF*, we just have to find the two highest (and leftmost) of the k chains and we get the two successor configurations by decreasing the height of those chains by 1, respectively. For example, consider the configuration $[1, 2, 3, 2]$ in Fig. 26, marked with a dashed border. Our *HLF* schedules the two tasks from the second and third chains. Then, the two corresponding successor configurations are: $[1, 1, 3, 2]$ if the task from the second chain finishes first, or $[1, 2, 2, 2]$ if the task from the third chain finishes first. After creating this graph, we only need to run a graph traversal algorithm like Breadth-First-Search or Depth-First-Search to find the number of possible ways from the start configuration to the desired end configuration, with can be done very efficiently, i.e., in linear time in the size of the graph.

Observe, that not all possible configurations for exactly two chains with exactly one task each occur in this graph. This is due to the chosen scheduling strategy. When ties are to be broken, tasks on the left are prioritized over tasks on the right. Hence, for example, the configuration $[1, 1, 0, 0]$ can never occur. It may occur when using another scheduling strategy.

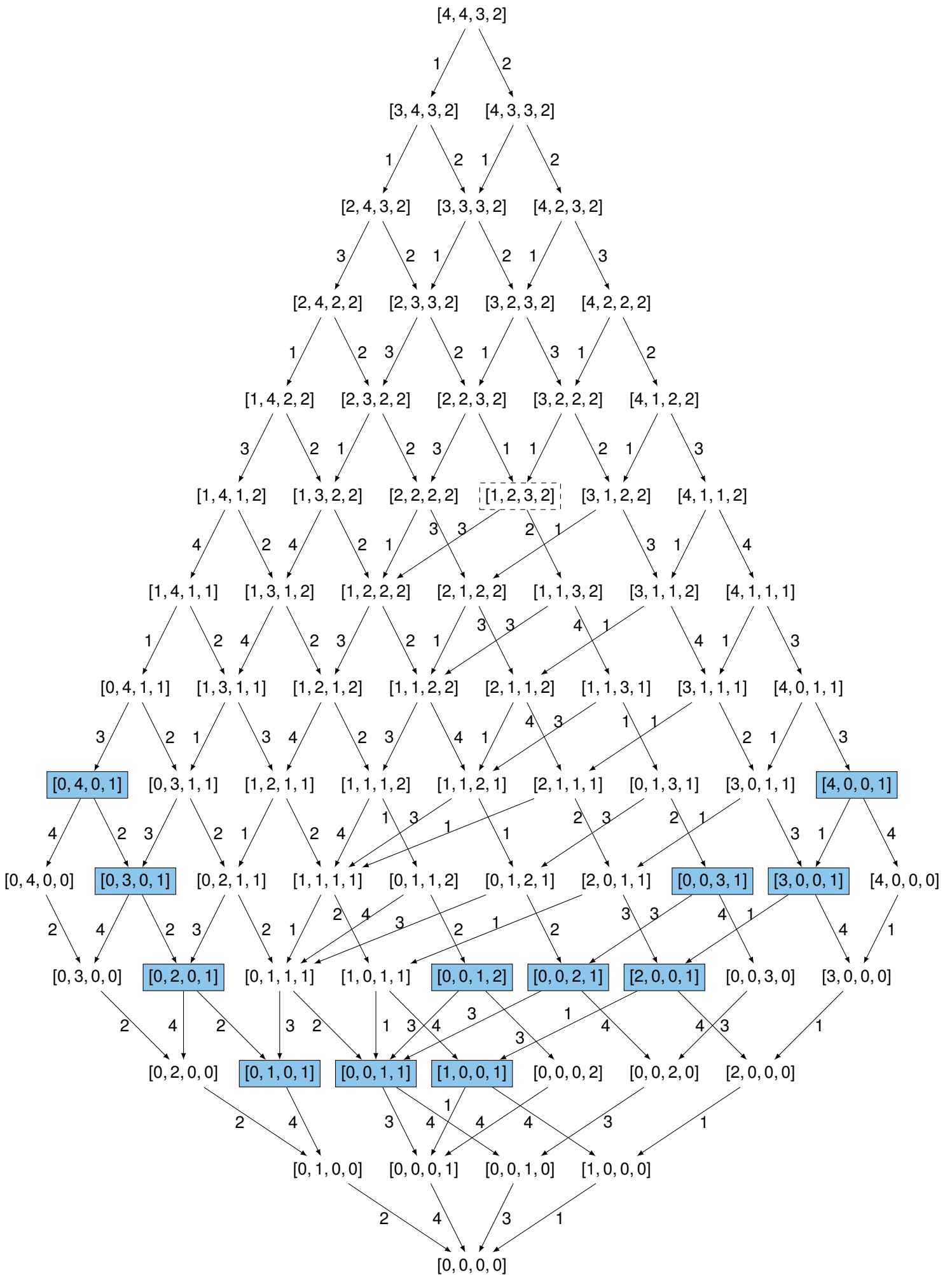


Fig. 26: The configuration graph for $[4, 4, 3, 2]$.

18.8 Y-graphs and Psi-graphs

In the last section, we focused on very simple precedence constraints, namely only sequences of chains. One decisive property of a sequence of chains is that every node has indegree at most 1. In the following, we will consider precedence constraints (and corresponding graphs) where this assumption is loosened and there can be nodes with an indegree of more than 1. At first, we study the case where at most one node is allowed to have two predecessors.

Definition 50 (Y-graph). A *Y-graph* is an acyclic, connected graph with the following properties:

- every node has outdegree at most 1,
- every node has indegree at most 2,
- at most one node has indegree 2.

A Y-graph can be decomposed into three chains as seen in Fig. 27. Using this fact, we denote a Y-graph with chain lengths r , s and t by $Y_{r,s,t}$.

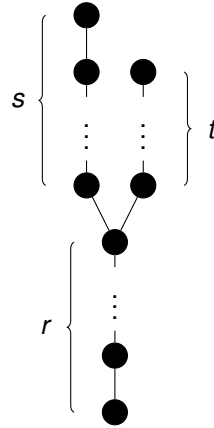


Fig. 27: A Y-graph $Y_{r,s,t}$.

As for all intrees, a Y-graph is optimally scheduled using *HLF* when considering exponential processing times and two processors, i.e., it optimally solves the scheduling problem

$$2|p_i \sim \exp(1); Y|\mathbb{E}(C_{\max}).$$

And because a Y-graph consists of three linked chains, we can reduce the problem of finding the expected makespan for such a schedule to the problem of calculating this value for a graph of two chains and a single chain. In particular, we get that

$$C_c(Y_{r,s,t}) = C_c([s, t]) + r.$$

This follows from the fact that the last r tasks in the chain $[r]$ can only be processed when both the chains from $[s, t]$ are completely processed, because every task from $[s, t]$ is an ancestor of every task in $[r]$, and of the highest task from $[r]$ in particular.

We can generalize the idea of a Y-graph to a graph where one node can have an outdegree 3 or more, see Fig. 28. We will call these graphs Psi-graphs and, using the same decomposition approach as for the Y-graphs, denote them by Ψ_{r,s_1,\dots,s_k} . Similarly to Y-graphs, the calculation of the expected makespan of an HLF schedule can be reduced to a case with a sequence of chains. In particular, it holds that

$$C_c(\Psi_{r,s_1,\dots,s_k}) = C_c([s_1, \dots, s_k]) + r.$$

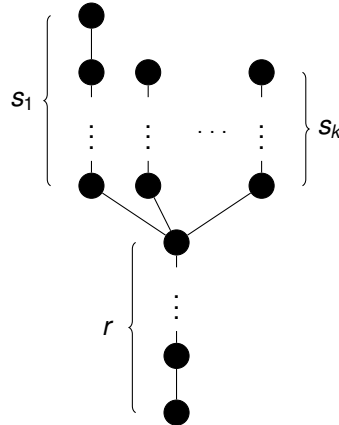


Fig. 28: A Psi-graph Ψ_{r,s_1,\dots,s_k} .

As we can see, the case where only one node is allowed to have a large indegree is easily handled, we can just use the formulas we derived for the special case with a sequence of chains to calculate the makespan of such a graph. Now, the question is whether we can derive a formula for a generalized case which relies only on the formulas for a sequence of chains. The following section covers that.

18.9 Intree Decomposition

In Section 18.8, we considered intrees where at most one node has more than one predecessor. These kinds of graphs are very restricted, which is why we want to generalize our approach to intrees which can have more than one node with more than one predecessor each. In this section, we decompose such an intree into several parts and consider the parts separately to find a formula for the expected makespan of an optimal *HLF* schedule. First, we introduce a notion to simplify the notations and improve the understanding of the decomposition.

Definition 51 (Join Node). A *join node* in G , or *join* for short, is a node with indegree more than 1. A *join level* denotes a level which contains at least one join node.

See Fig. 29 for an example. The join nodes are the blue ones.

At first, we consider a special class of intrees with a *width* of at most 2, i.e., all entries in the corresponding profiles are at most 2. The intree in Fig. 29 is an example for such an intree.

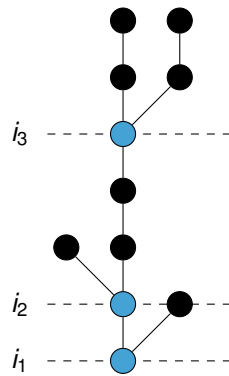


Fig. 29: An intree with width 2 and three join nodes on join levels i_1 , i_2 , and i_3 .

Because *HLF* is the optimal strategy in this case, it is clear to see that the lower sources below the join level i_3 can only be scheduled when at least one of the “arms” protruding from the join node on that level is completely processed, thus letting one processor be idle, and ready for processing other, lower sources. After this has happened, the situation is the same as before. Now, sources below join level i_2 must wait until one arm above that level is completely processed. And because the width of the intree is at most 2, we only have to calculate expected makespans of sequences of two chains for all cases that appear. This behavior is depicted in Fig. 30. Applying *HLF* to the intree in Fig. 29 first processes only the tasks above level i_3 , marked by the gray rectangle on the left. These tasks form two chains. After one of these chains is completely processed, the schedule arrives at the scenario on the right (or a similar one), and now processes the tasks in the gray rectangle on the right, also forming two chains. And so on.

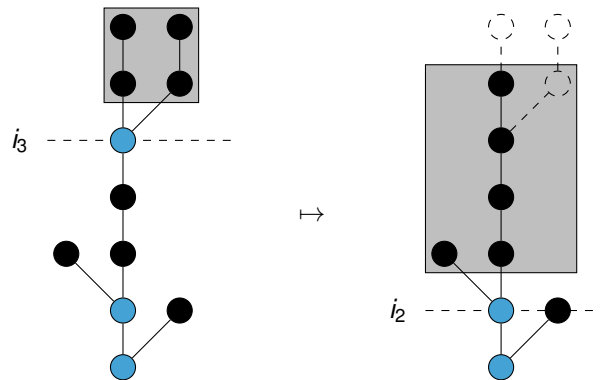


Fig. 30: *HLF* applied to the intree in Fig. 29.

Before we get into more detail in the corresponding calculations and expressions here, we generalize this approach to arbitrary wide intrees as this approach does not change that much by doing that. This means, that now, we consider general intrees and try to decompose them into several disjoint parts, see Fig. 31 for an example.

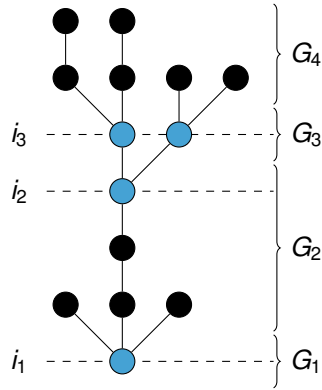


Fig. 31: An intree with decomposition (G_1, \dots, G_4) .

Let (i_1, i_2, \dots, i_m) be the increasing sequence of join levels for some m , i.e., $i_1 < i_2 < \dots < i_m$. Then the decomposition is as follows. G_1 contains all levels from 1 to i_1 , G_2 contains all levels from $i_1 + 1$ to i_2 , and so on, with G_{m+1} containing all levels from $i_m + 1$ to L . The decomposition of G is given by the ordered sequence $\mathcal{G} = (G_1, \dots, G_{m+1})$, see Fig. 31.

With this decomposition we can address our previous problem of finding a formula for the expected makespan in parts.

Let G be an intree with decomposition $\mathcal{G} = (G_1, \dots, G_{m+1})$. Note that each of the G_j 's is a sequence of chains, cp. Section 18.7. In particular, we have that G_1 contains only one chain (of length i_1), and G_j contains $p_{i_{j-1}+1}$ chains. First, *HLF* only schedules tasks from G_{m+1} , up until the decision point when there is only one chain of G_{m+1} left. Then, *HLF* only schedules available tasks from G_m plus the ones that are left from G_{m+1} . After G_m has only one task left, then *HLF* only schedules tasks from G_{m-1} plus the ones left from G_m , and so on, as seen in Fig. 30.

Given a part of the decomposition G_j , define

$$G_j \cup [s]$$

to be the graph that is obtained by adding a chain of length s to the longest chain in G_j (recall that G_j is a sequence of chains). Furthermore, define the following:

- $G_j \mapsto [s]$ = configuration where G_j is processed until only one chain of height s remains,
- $\mathcal{P}(G_j \mapsto [s])$ = probability that G_j is processed until only one chain of height s remains,
- $C_c(G_j \mapsto [s])$ = expected processing time *HLF* takes to schedule G_j until only one chain of height s remains.

The above definitions can be made similarly when G_j is substituted by $G_j \cup [s]$, because the structure of the underlying graph does not change. In addition, let $\text{height}(G_j)$ and $\text{height}(G_j \cup [s])$ denote the height of the highest chain in the sequence of chains of G_j or $G_j \cup [s]$, respectively. Obviously, $\text{height}(G_j \cup [s]) = \text{height}(G_j) + s$.

Then, for an intree G and its decomposition $\mathcal{G} = (G_1, \dots, G_{m+1})$ we get

$$C_c(G) = \sum_{s=1}^{\text{height}(G_{m+1})} \mathcal{P}(G_{m+1} \mapsto [s]) \cdot \left(C_c(G_{m+1} \mapsto [s]) + C_c(G_m \cup [s]) \right),$$

and $C(G_m \cup [s])$ is defined recursively by

$$C_c(G_j \cup [s]) = \sum_{t=1}^{\text{height}(G_j \cup [s])} \mathcal{P}(G_j \cup [s] \mapsto [t]) \cdot \left(C_c(G_j \cup [s] \mapsto [t]) + C_c(G_{j-1} \cup [t]) \right)$$

for $j \in \{2, \dots, m\}$, and a base case (see below).

From Section 18.7 on page 66 and due to the G_j 's being sequences of chains, we can plug in the formula for $C_c(G_j \cup [s] \mapsto [t])$. Suppose that G_j is of the form $[l_1^j, \dots, l_{k_j}^j]^4$, then $G_j \cup [s]$ is of the form $[l_1^j + s, l_2^j, \dots, l_{k_j}^j]$. Hence,

$$C_c(G_j \cup [s] \mapsto [t]) = \frac{1}{2} (l_1^j + s + l_2^j + \dots + l_{k_j}^j - t) = \frac{1}{2} \left(s + \sum_{c=1}^{k_j} l_c^j - t \right).$$

Putting these expressions together, we obtain the following lemma.

Lemma 52. *Given an intree G with its decomposition $\mathcal{G} = (G_1, \dots, G_{m+1})$ for some m , the expected makespan of an HLF schedule for G can be calculated by the recurrence relation*

$$C_c(G_j \cup [s]) = \sum_{t=1}^{\text{height}(G_j \cup [s])} \mathcal{P}(G_j \cup [s] \mapsto [t]) \cdot \left(\frac{1}{2} \left(s + \sum_{c=1}^{k_j} l_c^j - t \right) + C_c(G_{j-1} \cup [t]) \right)$$

for any s and $j = 2, \dots, m+1$, and the base case

$$C_c(G_1 \cup [s]) = \text{height}(G_1) + s.$$

Moreover,

$$C_c(G) = C_c(G_{m+1}).$$

So, again, we focus on calculating the probabilities that a certain configuration is reached. When we have a look at a sequence of chains and consider the possibilities and paths to a configuration with only one remaining chain, we observe that the situation is similar to when we wanted to know the number of possible paths in the configuration graph to a configuration with only two chains remaining, cp. Section 18.7 and Fig. 21. As an example, consider $G_j = [4, 3, 3, 2]$, then the chain grid can be seen in Fig. 32.

We are interested in the number of paths through the grid starting in the upper left corner, i.e., at node $[4, 3, 3, 2]$, and reaching a node in the lowest row, i.e., nodes $[4, 0, 0, 0]$ to $[1, 0, 0, 0]$. Identically to the approach before, we choose the nodes directly above as endpoints of the

⁴Here, the superscript j is another index, and not to be confused with an exponent. (The same holds for this footnote.)

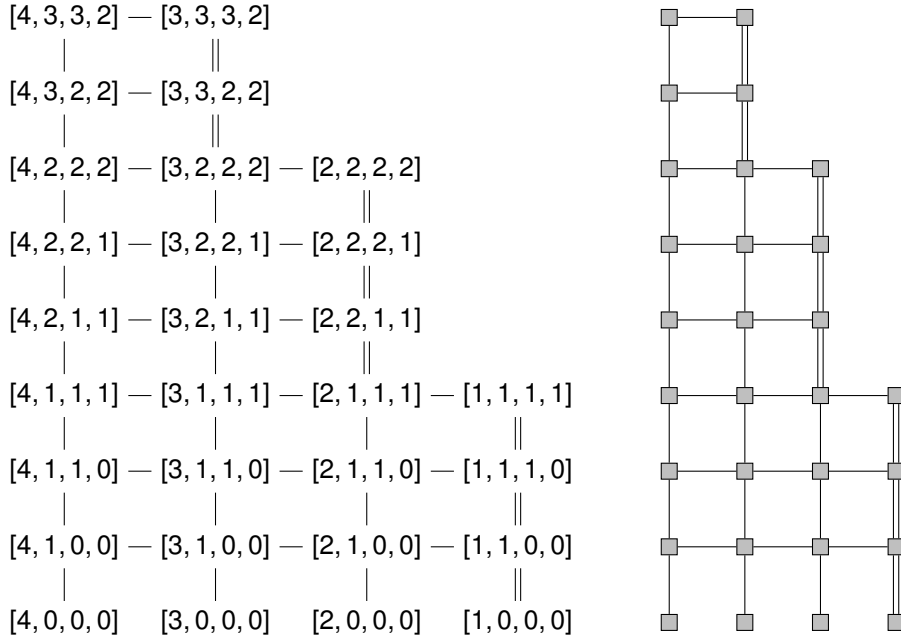


Fig. 32: The chain grid for $[4, 3, 3, 2]$ and its simplified version without labels.

paths. Note that for the special case to $[1, 1, 0, 0]$ we then have to multiply that number by 2 to get the correct number of paths to $[1, 0, 0, 0]$. Also note that $[4, 0, 0, 0] = [4]$. Hence, using the notation we derived in Section 18.7, we get

$$\begin{aligned}
 |\text{paths from } [4, 3, 3, 2] \text{ to } [4]| &= F_0^7(0) \cdot 2^{-7}, \\
 |\text{paths from } [4, 3, 3, 2] \text{ to } [3]| &= F_1^7(2) \cdot 2^{-8}, \\
 |\text{paths from } [4, 3, 3, 2] \text{ to } [2]| &= F_2^7(3) \cdot 2^{-9}, \\
 |\text{paths from } [4, 3, 3, 2] \text{ to } [1]| &= 2 \cdot F_3^7(2, 3, 2) \cdot 2^{-10}.
 \end{aligned}$$

Similar to what we used in Section 18.7, for every $j \in \{1, \dots, m+1\}$, we use the notations

$$\begin{aligned}
 h^j &= l_1^j, \\
 x_i^j &= p_{h^j-i}^j - 1, \quad i \in \{1, \dots, h^j - 2\}, \\
 x_{h^j-1}^j &= p_1^j - 2, \\
 H^j &= \sum_{c=2}^{h^j} l_c^j - 1, \\
 L_t^j &= \sum_{c=1}^{h^j} l_c^j - t - 1 = H^j + h^j - t, \quad \text{for some } t,
 \end{aligned}$$

and, additionally,

$$\delta_{ab} = \begin{cases} 1 & a = b, \\ 0 & a \neq b. \end{cases}$$

Because adding a chain $[s]$ to the top of the highest chain of G_j does not create additional paths, we can look at paths in the chain grid starting at level h^j , and have the following.

Lemma 53. *Let $G_j = [l_1^j, \dots, l_{k_j}^j]$, then*

$$\mathcal{P}(G_j \cup [s] \mapsto [t]) = (1 + \delta_{t1}) \cdot F_{h^j-t}^{H^j}(x_1^j, \dots, x_{h^j-1}^j) \cdot 2^{-L_t^j}.$$

Using this result, we can simplify the formulas we derived in Lemma 52 to obtain the following result.

Corollary 54. *Given an intree G and its decomposition (G_1, \dots, G_{m+1}) for some m , we have that $G_j = [l_1^j, \dots, l_{k_j}^j]$ for $j = 1, \dots, m+1$. Define*

$$h^j = \text{height}(G_j) = l_1^j, \quad j \in \{1, \dots, m+1\}$$

to be the height of the highest chain of G_j . Let $(p_1^j, \dots, p_{L^j}^j)$ denote the profile of G_j , and define

$$\begin{aligned} x_i^j &= p_{h^j-i}^j - 1, \quad i \in \{1, \dots, h^j - 2\}, \\ x_{h^j-1}^j &= p_1^j - 2, \\ H^j &= \sum_{c=2}^{h^j} l_c^j - 1, \\ L_t &= \sum_{c=1}^{h^j} l_c^j + s - t - 1 = H^j + h^j - t. \end{aligned}$$

Then the expected makespan of an HLF schedule for G can be calculated by the recurrence relation

$$C_c(G_j \cup [s]) = \sum_{t=1}^{H^j+s} 2^{-L_t^j+\delta_{t1}} \cdot F_{h^j-t}^{H^j}(x_1^j, \dots, x_{h^j-1}^j) \cdot \left(\frac{1}{2} \left(s + \sum_{c=1}^{k^j} l_c^j - t \right) + C_c(G_{j-1} \cup [t]) \right)$$

for any s and $j = 2, \dots, m+1$, and the base case

$$C_c(G_1 \cup [s]) = l_1^1 + s.$$

Moreover,

$$C_c(G) = C_c(G_{m+1}).$$



The Geometric Distribution

Table of Contents

19 Calculating the Expected Makespan	90
20 The Optimal Scheduling Strategy	93

Part V has shown us that the exponential distribution is relatively easy to handle because of the memorylessness. The obvious idea would be to consider other distributions with the same property. Unfortunately, as we have seen in Theorem 27, memorylessness is an exclusive property of the exponential distribution when considering continuous distributions. But luckily, there is another memoryless distribution, this time a discrete one.

Definition 55 (Geometric Distribution). A discrete random variable X with support \mathbb{N} is *geometrically distributed* with parameter/success probability p , denoted by $X \sim \text{geom}(p)$, if its probability mass function is

$$\mathcal{P}(X = k) = \begin{cases} p(1-p)^{k-1} & k \in \mathbb{N} \\ 0 & \text{otherwise.} \end{cases}$$

The interpretation is to repeatedly perform a random experiment with only two possible outcomes, i.e., a *Bernoulli experiment*, where the parameter p indicates the probability that an experiment is successful. This is why p is also called success probability. The geometrically distributed random variable X tells us the number of performed experiments before there is a first success. A well-known example is tossing a coin until it comes up heads. In this case, $p = \frac{1}{2}$.

Similar to an exponentially distributed random variable with parameter λ , which has an expected value of $\frac{1}{\lambda}$, a geometrically distributed random variable with parameter p has an expected value of $\frac{1}{p}$.

What we show with the next theorem is that the geometric distribution is the discrete analogon of the exponential distribution.

Theorem 56 (Memorylessness). *The geometric distribution is memoryless, i.e., for $X \sim \text{geom}(p)$ and $x \geq y$ we have*

$$\mathcal{P}(X > x \mid X > y) = \mathcal{P}(X > x - y).$$

Proof. As in Theorem 26, we prove this using only basic calculations. Then,

$$\begin{aligned} \mathcal{P}(X > x \mid X > y) &= \frac{\mathcal{P}(X > x)}{\mathcal{P}(X > y)} \\ &= \frac{1 - \mathcal{P}(X \leq x)}{1 - \mathcal{P}(X \leq y)} \\ &= \frac{1 - \sum_{k=1}^x p(1-p)^{k-1}}{1 - \sum_{\ell=1}^y p(1-p)^{\ell-1}} \\ &\stackrel{(9.4)}{=} 1 - (1 - (1-p)^{x-y}) \\ &= 1 - \sum_{k=1}^{x-y} p(1-p)^{k-1} \\ &= 1 - \mathcal{P}(X \leq x - y) \\ &= \mathcal{P}(X > x - y). \end{aligned}$$

□

The intuition behind the memorylessness of the geometric distribution can be explained using the interpretation. The coin tosses are independent, and one toss does not affect a later one in any way. This means that no matter how often a coin was tossed, the probability of success for the next coin toss is always the same. In other words: the coin cannot remember previous results.

But not only is the geometric distribution memoryless like the exponential distribution, it is the only discrete memoryless distribution, cp. Theorem 27.

Theorem 57. *Every memoryless discrete random variable is geometrically distributed.*

The proof can be found in [23].

As before, we have to consider the minimum of several independent random variables. In our cases, they will also be identically distributed, but it is just as easy to give the more general result.

Lemma 58. *Let X_1, \dots, X_n be independent, geometrically distributed random variables with parameters p_1, \dots, p_n , respectively. Then $X_1 \wedge \dots \wedge X_n$ is geometrically distributed with parameter $1 - \prod_{k=1}^n (1 - p_k)$.*

Proof. We show the proof for $n = 2$, the general case is done similarly with induction. First we have, for $i \in \{1, 2\}$,

$$\mathcal{P}(X_i \geq k) = \mathcal{P}(X_i > k - 1) = (1 - p_i)^{k-1},$$

because there must have been at least $k - 1$ failures before the first success. Using this, we obtain

$$\begin{aligned} \mathcal{P}(X_1 \wedge X_2 = k) &= \mathcal{P}(X_1 = k) \mathcal{P}(X_2 \geq k) + \mathcal{P}(X_2 = k) \mathcal{P}(X_1 \geq k) - \mathcal{P}(X_1 = k, X_2 = k) \\ &= p_1(1 - p_1)^{k-1}(1 - p_2)^{k-1} + p_2(1 - p_2)^{k-1}(1 - p_1)^{k-1} \\ &\quad - p_1(1 - p_1)^{k-1}p_2(1 - p_2)^{k-1} \\ &= (1 - p_1)^{k-1}(1 - p_2)^{k-1}(p_1 + p_2 - p_1p_2) \\ &= (1 - p_1)^{k-1}(1 - p_2)^{k-1}(1 - (1 - p_1)(1 - p_2)) \\ &= (1 - (1 - p_1)(1 - p_2))((1 - p_1)(1 - p_2))^{k-1}, \end{aligned}$$

and this is the probability for the value k of a geometrically distributed random variable with parameter $1 - (1 - p_1)(1 - p_2)$. \square

Let $X \sim \text{geom}(p)$ and consider the processing times of the tasks independent and identically distributed according to X . Then, the expected remaining processing time of an active task that has already been processed for i time units is $\frac{1}{p}$, again due to memorylessness, as in Part V on page 34 with the exponential distribution.

Whenever two machines are busy processing two different tasks, then the expected time until one of them finishes is $\frac{1}{1 - (1 - p)^2}$ time units.

19 Calculating the Expected Makespan

Now we consider the scheduling problem

$$2|p_i \sim \text{geom}(p); \text{intree} | \mathbb{E}(C_{\max}),$$

i.e., we have two processors at our disposal, intree precedence constraints between the tasks, and “geometric processing times” (an abbreviation for “independent, identically and geometrically distributed processing times, each with parameter p ”). Then, because of the property of memorylessness – and because the processing times are identically distributed –, whenever two tasks are being processed simultaneously, both tasks are equally likely to finish first. Of course, this is greatly similar to the case with exponential processing times. However, there is one big and important difference. Since the exponential distribution is continuous, two tasks being processed in parallel will never finish at the same time, i.e., the probability of that ever happening is zero. In the discrete case with the geometric distribution, this probability is strictly greater than zero, and depends on the value p . We will calculate these probabilities and adjust the formulas for the makespan accordingly. Again, we denote the random variables of the processing times of the two processed tasks by X and Y , respectively. First, consider the case that the task associated with the random variable X finishes first, and at time k :

$$\mathcal{P}(X = k, Y > k) = \mathcal{P}(X = k) \mathcal{P}(Y > k) = p(1 - p)^{k-1}(1 - p)^k.$$

Hence, we have that

$$\begin{aligned} \mathcal{P}(X < Y) &= \sum_{k=1}^{\infty} \mathcal{P}(X = k, Y > k) \\ &= \sum_{k=1}^{\infty} p(1 - p)^{k-1}(1 - p)^k \\ &= p(1 - p) \sum_{k=0}^{\infty} (1 - p)^{2k} \\ &= p(1 - p) \cdot \frac{1}{1 - (1 - p)^2} \\ &= \frac{1 - p}{2 - p}. \end{aligned}$$

This formula also has a nice interpretation. Consider the coin tosses for X and Y done simultaneously. Because of memorylessness, we ignore all coin tosses where both coins come up tails, and only focus on the first time that at least one of the coins comes up heads. This is handled as only one experiment (with two coins tossed simultaneously). In order to calculate the probability of that, we define E_i as the event that coin i comes up heads. Then, the probability

that at least one coin comes up heads is $2p - p^2$, which can be calculated by

$$\begin{aligned}
 \mathcal{P}(\text{at least one coin comes up heads}) &= \mathcal{P}(E_1 \cup E_2) \\
 &= \mathcal{P}(E_1) + \mathcal{P}(E_2) - \mathcal{P}(E_1)\mathcal{P}(E_2) \\
 &= p(1 - p) + (1 - p)p - p^2 \\
 &= 2p - p^2.
 \end{aligned}$$

Under the condition that at least one coin comes up heads, we now have three distinct cases: only the first coin comes up heads, only the second does, or both. Calculating these conditional probabilities gives us exactly the probabilities to the three events $X < Y$, $X > Y$, and $X = Y$:

$$\begin{aligned}
 \mathcal{P}(X < Y) &= \mathcal{P}(X = \text{heads}, Y = \text{tails} \mid \text{at least one coin comes up heads}) \\
 &= \frac{\mathcal{P}(X = \text{heads}, Y = \text{tails})}{\mathcal{P}(\text{at least one coin comes up heads})} \\
 &= \frac{p(1 - p)}{2p - p^2} \\
 &= \frac{1 - p}{2 - p}, \\
 \mathcal{P}(X > Y) &= \frac{\mathcal{P}(X = \text{tails}, Y = \text{heads})}{\mathcal{P}(\text{at least one coin comes up heads})} \\
 &= \frac{1 - p}{2 - p} \\
 \mathcal{P}(X = Y) &= \frac{\mathcal{P}(X = \text{heads}, Y = \text{heads})}{\mathcal{P}(\text{at least one coin comes up heads})} \\
 &= \frac{p^2}{2p - p^2} \\
 &= \frac{p}{2 - p}.
 \end{aligned}$$

Now, we want to have a formula for the expected makespan in the same way we do for the exponential distribution. We formulate two different approaches, cp. (14.1) and (14.2) from page 31.

The first one is the approach from (14.1):

$$C_c(G) = \frac{1}{1 - (1 - p)^2} + \frac{1 - p}{2 - p} C_c(G_x) + \frac{1 - p}{2 - p} C_c(G_y) + \frac{p}{2 - p} C_c(G_{x,y}). \quad (19.1)$$

The first term is the expected number of time steps until at least one of the two tasks finishes. The next three expressions correspond to the three cases $\mathcal{P}(X < Y)$, $\mathcal{P}(X > Y)$, and $\mathcal{P}(X = Y)$, in this order. The successor configurations G_x , G_y , and $G_{x,y}$ are defined as $G \setminus \{x\}$, $G \setminus \{y\}$, and $G \setminus \{x, y\}$, respectively, see Lemma 31 on page 41.¹ Now, the event that a coin comes up heads corresponds to a task being finished.

¹Not to be confused with the parts G_j of the decomposition seen in Section 18.9. There we have G_j with a numeric index j , whereas here we have G_x indexed by a task x .

The other approach is just a summation over all possible values that can occur. Given an intree G , we make a case distinction of the three different cases that may occur: one task finishes first, the other task finishes first, or both tasks finish at the same time. Furthermore we take the sum over all possible values k , which range from 1 to ∞ . Again, for simplicity, we also denote the configuration corresponding to the intree G by G . Note that we do not need to mark any active tasks because the underlying distribution for the processing times is memoryless, we just need to agree on a certain scheduling strategy before. Then, the formula for the expected makespan of a configuration is

$$C_c(G) = \sum_{k=1}^{\infty} \sum_{\ell=1}^{\infty} p(1-p)^{k-1} p(1-p)^{\ell-1} (\min(k, \ell) + C_c(G^\circ)), \quad (19.2)$$

with

$$G^\circ = \begin{cases} G_x & k < \ell \\ G_y & k > \ell \\ G_{x,y} & k = \ell. \end{cases}$$

This is just the formula in (14.2) on page 31 applied to the geometric distribution. To get rid of the different cases, we can rewrite this as three summations

$$\begin{aligned} C_c(G) &= \sum_{k=1}^{\infty} \sum_{\ell=k+1}^{\infty} p^2(1-p)^{k+\ell-2} (k + C_c(G_x)) \\ &\quad + \sum_{k=1}^{\infty} \sum_{\ell=1}^{k-1} p^2(1-p)^{k+\ell-2} (\ell + C_c(G_y)) \\ &\quad + \sum_{k=1}^{\infty} p^2(1-p)^{2k-2} (k + C_c(G_{x,y})). \end{aligned}$$

The missing cases, i.e., the base cases of the recurrence, are chains, just like we had before. A singular task is expected to finish in $\frac{1}{p}$ time steps. And as the expected value is additive, a chain of height n has an expected makespan of $\frac{n}{p}$.

There is one other important difference between using the geometric distribution like here and using the exponential distribution. As can be seen easily, (19.1) is not scalable by some factor dependent on p . In Part V, we argued that the calculation for the expected makespan scales with the factor $\frac{1}{\lambda}$. In (19.1), there is no such factor. This is due to Lemma 58 which shows that the parameter for the minimum of two geometric distributions is proportional to the product of the parameters of those distributions, and not the sum. This yields the squared expression for the expected value in (19.1), whereas the base case still has the linear expression of $\frac{n}{p}$. This is why, in contrast to the continuous case, we cannot just assume $p = \frac{1}{2}$ or some other easy value for p . We have to let the parameter p be arbitrary.

It should be clear that the two approaches from (19.1) and (19.2) are in fact identical, and are just covering two different interpretations. To be convinced, we consider a small example and calculate the expected makespan using the two different ways. This can be seen in Appendix B in Calculation 2.

20 The Optimal Scheduling Strategy

Although we have already argued about the equivalence of certain ways of calculating the optimal expected makespan in Chapter 19, we have not yet proven what the optimal strategy is. In the example in Appendix B on page 181 we used *HLF* without proper justification. What we want to show is that *HLF* is the optimal strategy for this intree and for arbitrary intrees as well, i.e., we show optimality of *HLF* for the scheduling problem

$$2|p_i \sim \text{geom}(p); \text{intree} | \mathbb{E}(C_{\max}).$$

What we actually do is a repetition of the results from Chapter 17. Some lines are obviously different from what we have done before, but the crucial argument we need to prove these results is the same, namely the memorylessness.

First, we give a recurrence relation for the makespan using only the profile of an intree, cp. Lemma 29 on page 39.

Lemma 59. *Let G be an intree with $\text{profile}(G) = p$, then the expected makespan of G when applying any *HLF* can be calculated by:*

(i): *If $p = ()$, then $C_p(p) = 0$,*

(ii): *If $p = (1)$, then $C_p(p) = \frac{1}{p}$,*

(iii): *If the highest level has only one task and G is a chain, i.e. $p_L = 1$ and $n = L$, then*

$$C_p(p_1, \dots, p_L) = \frac{1}{p} + C_p(p_1, \dots, p_{L-1}),$$

(iv): *If the highest level has more than one task and G is not a chain, i.e. $p_L > 1$ and $n > L$,*

$$C_p(p_1, \dots, p_L) = \frac{1}{1 - (1 - p)^2} + 2 \cdot \frac{1 - p}{2 - p} C_p(p_1, \dots, p_{L-1}) + \frac{p}{2 - p} C_p(p_1, \dots, p_{L-2}),$$

(v): *If the highest level has only one task and G is not a chain, i.e. $p_L = 1$ and $n > L$, then*

$$\begin{aligned} C_p(p_1, \dots, p_{L'}, \dots, p_L) &= \frac{1}{1 - (1 - p)^2} + \frac{1 - p}{2 - p} C_p(p_1, \dots, p_{L'}, \dots, p_{L-1}) \\ &\quad + \frac{1 - p}{2 - p} C_p(p_1, \dots, p_{L'} - 1, \dots, p_L) \\ &\quad + \frac{p}{2 - p} C_p(p_1, \dots, p_{L'} - 1, \dots, p_{L-1}). \end{aligned}$$

Proof. The proof uses the same arguments as the proof of Lemma 29, just using the equations from (19.1) and (19.2). □

We use the concept of flatness defined in Definition 30 on page 41 for the following result.

Lemma 60. Let G be an intree and let G' be obtained by attaching a source of G on a lower level. Then $C_c(G) \geq C_c(G')$.

Proof. We prove this by induction on the number of tasks.

Induction Base: It holds that

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{2p^2 - 11p + 13}{p(2-p)^2} < \frac{5p^2 - 17p + 16}{p(2-p)^2} = C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right), \quad (20.1)$$

due to some more elaborate calculations, which can be seen in detail in Appendix D on page 186.

Induction Step: Let n be the number of tasks in both G and G' . We have the scenario as seen in Fig. 33. There we have configurations G and G' with their successor configurations. Note that we may even draw the successor configurations without the tasks marked as active because of the memorylessness. The (now three) cases for the successor configurations can

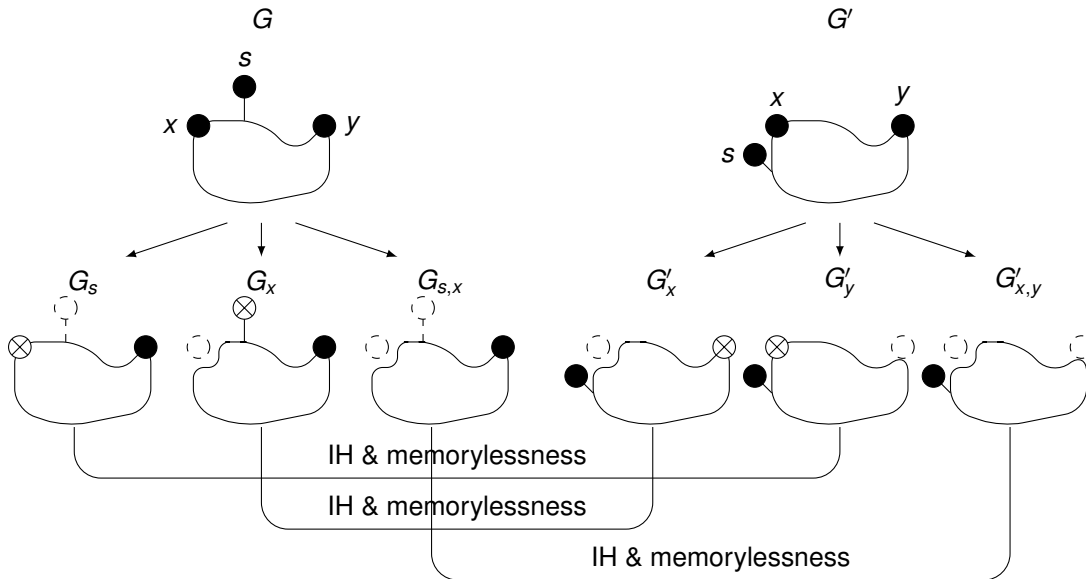


Fig. 33: G and G' and their respective successor configurations.

be handled in the same way as with the exponential distribution by the induction hypothesis (IH) and the memorylessness.

□

Lemma 32 can be used identically in the case of geometric processing times as well. This is because the flatness relation does not change with the other distribution, and we can use Lemma 60.

Hence, we get the following.

Lemma 61. *Let G and H be two intrees with the same number of tasks, and $G \preceq H$. Consider an HLF being applied to G and a non-HLF S being applied to H . Then the expected makespan of G is at most the expected makespan of H .*

Proof. Again, by induction on the number of tasks.

Induction Base: For the two base cases, it holds that

$$C_c \left(\begin{array}{c} \circ \quad \circ \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) < C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right)$$

and

$$C_c \left(\begin{array}{c} \circ \quad \circ \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) < C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right),$$

whose proofs can be seen in Appendix D.

Induction Step: The remaining parts of the proof now work exactly as in the proof of Lemma 33.

□

With these results we get the optimality of HLF for geometric processing times as well.

Theorem 62. *Any HLF is optimal for the scheduling problem $2|p_i \sim \text{geom}(p); \text{intree}| \mathbb{E}(C_{\max})$.*



The Uniform Distribution

Table of Contents

21 Discrete Case	99
22 Continuous Case	102
23 Calculating the Expected Makespan	104
23.1 Discrete Case	105
23.2 Continuous Case	108
24 <i>HLF</i> and Uniform Processing Times	110

Apart from the memoryless distributions, there are many other distributions that would make sense to be considered. For example the Erlang distribution as the sum of exponential distributions (see Part VIII on page 114) or the negative binomial distribution as the sum of geometric distributions. But at first, the distributions which we deemed the most interesting are uniform distributions. The applicability of uniformly distributed processing times seems more prominent than other distributions, as well as easier to calculate with.

21 Discrete Case

Definition 63 (Discrete uniform distribution). A discrete random variable X with support $A = \{a_1, \dots, a_n\}$ for some $n \in \mathbb{N}$ is (*discretely*) *uniformly distributed* over the set A , denoted by $X \sim \mathcal{U}\{a_1, \dots, a_n\}$, if it holds that

$$\mathcal{P}(X = a_i) = \frac{1}{n} \text{ for all } i \in \{1, \dots, n\}$$

and $\mathcal{P}(X = b) = 0$ for all $b \notin A$.

Whenever it is obvious from the context, we drop the word “discrete” (or “continuous”), and just say that a random variable is uniformly distributed.

The discrete uniform distribution is symmetric in the sense that each possible outcome, i.e., one of the values $\{a_1, \dots, a_n\}$, is equally likely to be the result. A standard example would be a fair six-sided die, where the values $\{1, 2, 3, 4, 5, 6\}$ are all equally likely to show up. In particular, the probability of a specific side to be on top is $\frac{1}{6}$.

The expected value of a discrete uniformly distributed random variable $X \sim \mathcal{U}\{a_1, \dots, a_n\}$ is

$$\mathbb{E}(X) = \frac{1}{n} \sum_{i=1}^n a_i,$$

i.e., the arithmetic mean of the support values. Usually, we consider only two different values a and b for the processing time of a task, which means that the expected time until a task finishes is $\frac{a+b}{2}$.

Moreover, we want to consider the minimum of several uniformly distributed random variables as we may have several tasks being processed simultaneously. First, consider the simple case with $X, Y \sim \mathcal{U}\{a, b\}$, i.e., X and Y are identically distributed. Then the minimum $X \wedge Y$ can also only take the two values a and b , and we have

$$\mathcal{P}(X \wedge Y = a) = \mathcal{P}(X = a) + \mathcal{P}(Y = a) - \mathcal{P}(X = a, Y = a) = \frac{3}{4}, \quad (21.1)$$

$$\mathcal{P}(X \wedge Y = b) = \mathcal{P}(X = b, Y = b) = \frac{1}{4}, \quad (21.2)$$

as well as

$$\mathbb{E}(X \wedge Y) = a \cdot \mathcal{P}(X \wedge Y = a) + b \cdot \mathcal{P}(X \wedge Y = b) = \frac{3a + b}{4}. \quad (21.3)$$

In contrast to the other distributions which we considered so far (Parts V and VI on pages 34 and 88), the discrete uniform distribution is not memoryless as we have seen in Theorem 57. The big difference is when considering a decision point, and only one task has finished being processed. Then, the other is still being processed, but apart from what we have done so far, we cannot just assume that both the newly scheduled task **and** the active task are identically distributed. This was a property that was exclusive to memoryless distributions. So, unfortunately, we only rarely use the results from (21.1) to (21.3). In Parts V and VI, we have shown how the minimum of several exponentially (or geometrically) distributed random variables is exponentially (or geometrically) distributed, which was an easy calculation even though the random variables were not necessarily identically distributed. The interesting part is that we did not need the random variables to be **not** identically distributed as in later applications – because of memorylessness – they always were. Unfortunately, now that we need the random variables **not** to be identically distributed, these calculations become more tedious. A general formula can get quite messy, so we restrict ourselves to only a few simple cases, namely with random variables

$$X \sim \mathcal{U}\{a, b\},$$

$$Y \sim \begin{cases} \mathcal{U}\{a-i, b-i\} & i < a \\ \mathcal{U}\{0, b-i\} & i \geq a. \end{cases}$$

The interpretation is that X corresponds to the processing time of the task which is scheduled next. The random variable Y corresponds to the remaining processing time of the active task. The reason why Y is distributed like that is because it is actually distributed in the same way as the total processing time \bar{Y} conditioned on $\bar{Y} > i$:

$$\begin{aligned} \mathcal{P}(Y \leq j) &= \mathcal{P}(\bar{Y} \leq i+j \mid \bar{Y} > i) \\ &= 1 - \mathcal{P}(\bar{Y} > i+j \mid \bar{Y} > i) \\ &= 1 - \frac{\mathcal{P}(\bar{Y} > i+j, \bar{Y} > i)}{\mathcal{P}(\bar{Y} > i)} \\ &= \begin{cases} 0 & i+j < a \\ \frac{1}{2} & i+j \in [a, b) \\ 1 & i+j > b, \end{cases} \end{aligned}$$

which is exactly the probability density function of a uniform distribution over $\{a-i, b-i\}$. So, even though the resulting conditioned distribution is not identical (because of the lack of memorylessness), it is at least again a uniform distribution. In other words: instead of writing this as a conditional probability (or a conditional distribution), we consider the equivalent distribution over all remaining values. In our case, with only two possible values for each random variable, we have the following six different cases to consider, which are denoted by the letters **A-F** from now on.

Case A: Let $i < a$ and $i > b - a$. There are only two different values for the next decision point, namely the two values that Y can be mapped to, see Fig. 34. The rectangular

markings on the time line indicate the two possible values for X , the circular markings indicate the values for Y . As $i > b - a \iff b - i < a$, we have that $\mathcal{P}(X \wedge Y = Y) = 1$ in this case. This means that the two different values $a - i$ and $b - i$ are both equally likely to occur, because Y is uniformly distributed.

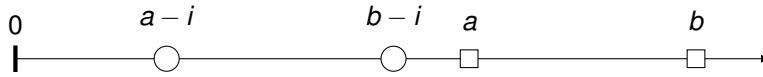


Fig. 34: Case **A**: $i < a$ and $i > b - a$.

Case B: Let $i < a$ and $i < b - a$. There are three different values for the next decision point, see Fig. 35. The value $a - i$ is the minimum with a probability of $\frac{1}{2}$ as $\mathcal{P}(Y = a - i) \mathcal{P}(X > a - i) = \frac{1}{2} \cdot 1$. For the value a we have $\mathcal{P}(Y > a) \mathcal{P}(X = a) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, and for $b - i$ we have $\mathcal{P}(Y = b - i) \mathcal{P}(X > b - i) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

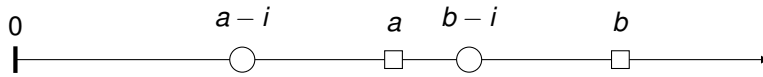


Fig. 35: Case **B**: $i < a$ and $i < b - a$.

Case C: Let $i \geq a$ and $i > b - a$. Then, $a - i \leq 0$ and as we have the convention that decision points are considered after a task is finished, a remaining processing time of 0 is not possible here (except for $a = 0$). But this means that there is only one possible value for Y left, i.e., $\mathcal{P}(Y = b - i) = 1$, see Fig. 36. Moreover, as in case **A**, we have $\mathcal{P}(X \wedge Y = Y) = 1$, so the only possible value for the next decision point is $b - i$.



Fig. 36: Case **C**: $i \geq a$ and $i > b - a$.

Case D: Let $i \geq a$ and $i < b - a$. Again, $a - i \leq 0$, but this time there are two possible values to be considered, both with probability $\frac{1}{2}$, see Fig. 37. This can be checked easily with elementary probability calculations like in case **B** for example.

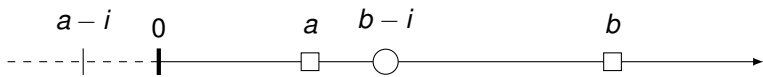


Fig. 37: Case **D**: $i \geq a$ and $i < b - a$.

Case E: Let $i < a$ and $i = b - a$. There are three different possibilities. Y may have the smallest value $a - i$ alone, i.e., $\mathcal{P}(Y = a - i) \mathcal{P}(X > a - i) = \frac{1}{2}$, or Y may have the smallest value

$a = b - i$ alone, i.e., $\mathcal{P}(Y = a) \mathcal{P}(X > a) = \frac{1}{4}$, or X and Y both share the value a , i.e., $\mathcal{P}(Y = a) \mathcal{P}(X = a) = \frac{1}{4}$, see Fig. 38.

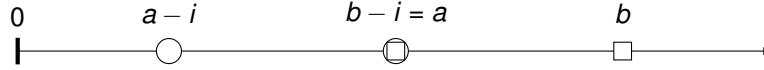


Fig. 38: Case **E**: $i < a$ and $i = b - a$.

Case F: Let $i \geq a$ and $i = b - a$. Because $a - i \leq 0$ we know that $\mathcal{P}(Y = a) = 1$. So then, we have $\mathcal{P}(Y = a) \mathcal{P}(X > a) = \frac{1}{2}$ and $\mathcal{P}(Y = a) \mathcal{P}(X = a) = \frac{1}{2}$, see Fig. 39.

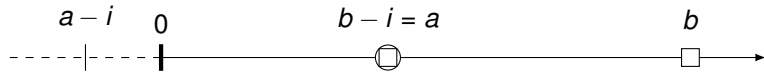


Fig. 39: Case **F**: $i \geq a$ and $i = b - a$.

22 Continuous Case

Definition 64 (Continuous uniform distribution). A continuous random variable X with support $[a, b]$ is (*continuously*) *uniformly distributed* over $[a, b]$, denoted by $X \sim \mathcal{U}[a, b]$, if its probability density function f_X is

$$f_X(x) = \begin{cases} \frac{1}{b-a} & x \in [a, b] \\ 0 & \text{otherwise,} \end{cases}$$

that is, its cumulative distribution function is

$$F_X(x) = \mathcal{P}(X \leq x) = \begin{cases} 0 & x < a \\ \frac{b-x}{b-a} & x \in [a, b] \\ 1 & x > b. \end{cases}$$

The expected value of a (continuous) uniformly distributed random variable $X \sim \mathcal{U}[a, b]$ is

$$\mathbb{E}(X) = \frac{a+b}{2},$$

i.e., the same as for a discrete uniform random variable over the values $\{a, b\}$.

As no other continuous distribution than the exponential distribution can be memoryless, see Theorem 27 on page 36, the uniform distribution is not either.

In general, the minimum of several uniform distributions can be handled as easily as in the case with exponential (or even geometric) distributions. This is why we focus only on the special cases that we really need for our calculations. Consider two independent and identically

distributed, (continuous) uniform random variables over $[a, b]$, i.e., $X, Y \sim \mathcal{U}[a, b]$. Then, the minimum $X \wedge Y$ has support $[a, b]$ as well, and we have

$$\begin{aligned} F_{X \wedge Y}(z) &= \mathcal{P}(X \wedge Y \leq z) \\ &= 1 - \mathcal{P}(X > z) \mathcal{P}(Y > z) \\ &= \begin{cases} 0 & z < a \\ 1 - \left(\frac{b-z}{b-a}\right)^2 & z \in [a, b] \\ 1 & z > b. \end{cases} \end{aligned}$$

From this we can deduce the probability density function

$$f_{X \wedge Y}(z) = \frac{d}{dx} (F_{X \wedge Y})(z) = \begin{cases} \frac{2(b-z)}{(b-a)^2} & z \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

and the expected value

$$\mathbb{E}(X \wedge Y) = \int_{\mathbb{R}} z f_{X \wedge Y}(z) dz = \frac{2a+b}{3}. \quad (22.1)$$

For the cases with two continuous uniform distributions which are not identical, we consider, similarly to the discrete case, the two random variables

$$\begin{aligned} X &\sim \mathcal{U}[a, b], \\ Y &\sim \begin{cases} \mathcal{U}[a-i, b-i] & i < a \\ \mathcal{U}[0, b-i] & i \geq a. \end{cases} \end{aligned}$$

Then again, we have to consider several different cases. But in the continuous case, we have $\mathcal{P}(X = Y) = 0$, i.e., cases **E** and **F** from the discrete case must not be considered as they never occur.

Case A: Let $i < a$ and $i > b - a$. Then, $\mathcal{P}(X \wedge Y = Y) = 1$ and the possible values to be considered are in the interval $[a-i, b-i]$, see Fig. 40. The red diagonally striped interval is the support of Y whereas the blue vertically striped interval depicts the support of X . For the possible values we have to calculate an integral and use the probability density function of Y .

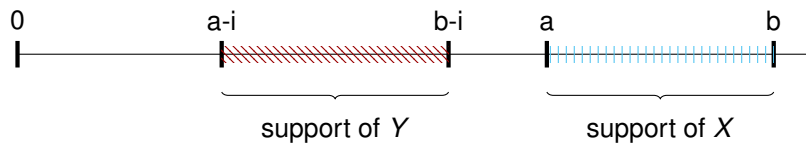


Fig. 40: Case **A**: $i < a$ and $i > b - a$.

Case B: Let $i < a$ and $i \leq b - a$. We have overlapping values in the interval $[a, b - i]$, i.e., in this interval both random variables have the chance to be the minimum. Now the values for the probabilities are no longer trivial. Consider the probability that the minimum $X \wedge Y$ is given by X , or in other words $X = t_x, Y = t_y$, and $t_x < t_y$ for some values t_x, t_y . Then we have

$$\begin{aligned} \mathcal{P}(X \wedge Y = X) &= \mathcal{P}(t_x < t_y) \quad \forall t_y \\ &= \frac{t_y - a}{b - a} \quad \forall t_y, \end{aligned}$$

where “ $\forall t_y$ ” means that we have to consider all possible values for t_y when calculating this probability, which we do by letting t_y be the integration variable. Similarly, we get

$$\mathcal{P}(X \wedge Y = Y) = \frac{t_x - (a - i)}{b - a} \quad \forall t_x$$

with t_x being the integration variable here. For the use of these parameters, see Chapter 23.

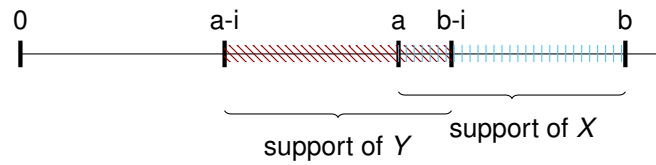


Fig. 41: Case **B**: $i < a$ and $i \leq b - a$.

Case C: Let $i \geq a$ and $i > b - a$. The only interval to be considered is $[0, b - i]$, see Fig. 42.



Fig. 42: Case **C**: $i \geq a$ and $i > b - a$.

Case D: Let $i \geq a$ and $i \leq b - a$. Then again, we have to make the adjustments seen in case **B** for the interval $[a, b - i]$, see Fig. 43. We obtain

$$\begin{aligned} \mathcal{P}(X \wedge Y = X) &= \frac{t_y - a}{b - a} \quad \forall t_y, \\ \mathcal{P}(X \wedge Y = Y) &= \frac{t_x - (a - i)}{b - a} \quad \forall t_x, \end{aligned}$$

with the integration variables t_y and t_x , respectively.

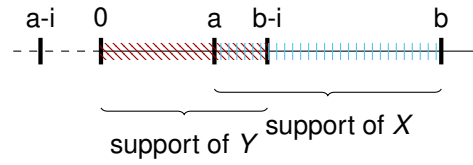


Fig. 43: Case **D**: $i \geq a$ and $i \leq b - a$.

23 Calculating the Expected Makespan

23.1 Discrete Case

Now we want to use these results to calculate the expected makespan of the scheduling problem

$$2|p_i \sim \mathcal{U}\{a, b\}; \text{intree} | \mathbb{E}(C_{\max}),$$

i.e., two processors, “(discrete) uniform processing times” (as an abbreviation for “independent, identically and discretely uniformly distributed processing times over $\{a, b\}$ ”), intree precedence constraints, and subject to expected makespan minimization.

In contrast to the memoryless distributions we considered before, we now need to characterize a configuration including the active tasks. By convention, in the two processor case, a configuration at a decision point can have at most one active task, cp. Chapter 7. The notation for such a configuration is $(G, (y : i))$, where G is the underlying intree and y has been active for i time units. We write the expected makespan in the form of (14.2) from page 31:

$$C_c(G) = \sum_{\text{possibility } p} \mathcal{P}(p) \cdot (\text{passed time} + \text{expected remaining processing time}),$$

where “expected remaining processing time” is one of $C_c(G_x, (x : j))$ for some j , $C_c(G_y, (x : j))$ for some j , or $C_c(G_{x,y})$. Depending on whether the current configuration has two or fewer successor configurations, the formula for the expected makespan is as follows.

Case 1: The current configuration has only one successor configuration, i.e., it is a chain. Then the expected makespan can be calculated by

$$C_c(G) = \frac{a+b}{2} \cdot n,$$

$$C_c(G, (y : i)) = \begin{cases} \frac{a+b}{2} - i + \frac{a+b}{2}(n-1) & i < a \\ b - i + \frac{a+b}{2}(n-1) & i \geq a, \end{cases}$$

where n denotes the number of tasks in G . The explanation should be obvious as we can only process the tasks in the chain sequentially, one after another. For each task, we have an expected processing time of $\frac{a+b}{2}$. This value has to be multiplied by n in the case that there are no active tasks. If the topmost task in the chain y has been

active for i time units, then depending on whether i is smaller or greater than a , the expected remaining processing time of this task is $\frac{a+b}{2} - i$ or $b - i$, respectively.

Case 2: The current configuration has at least two successor configurations. Then there is the simple case where there is no active task, thus,

$$C_c(G) = \frac{1}{4} (a + C_c(G_{x,y})) + \frac{1}{4} (a + C_c(G_y, (x:a))) \\ + \frac{1}{4} (a + C_c(G_x, (y:a))) + \frac{1}{4} (b + C_c(G_{x,y})).$$

With the help of the equations in (21.1) to (21.3) on page 99 it is easy to see that in this case the two approaches – one with the expected value written outside the sum, one with the “passed time” values inside the sum – are equivalent.

If there is an active task y , active for i time units, then we have to consider the six cases **A-F** separately, which results in

$$C_c(G, (y:i)) = \begin{cases} \frac{1}{2} (a - i + C_c(G_y, (x:a - i))) \\ \quad + \frac{1}{2} (b - i + C_c(G_y, (x:b - i))) & \text{Case A: } i < a, i > b - a \\ \frac{1}{2} (a - i + C_c(G_y, (x:a - i))) \\ \quad + \frac{1}{4} (a + C_c(G_x, (y:a + i))) \\ \quad + \frac{1}{4} (b - i + C_c(G_y, (x:b - i))) & \text{Case B: } i < a, i < b - a \\ b - i + C_c(G_y, (x:b - i)) & \text{Case C: } i \geq a, i > b - a \\ \frac{1}{2} (a + C_c(G_x, (y:a + i))) \\ \quad + \frac{1}{2} (b - i + C_c(G_y, (x:b - i))) & \text{Case D: } i \geq a, i < b - a \\ \frac{1}{2} (a - i + C_c(G_y, (x:a - i))) \\ \quad + \frac{1}{4} (a + C_c(G_y, (x:a))) \\ \quad + \frac{1}{4} (a + C_c(G_{x,y})) & \text{Case E: } i < a, i = b - a \\ \frac{1}{2} (a + C_c(G_y, (x:a))) \\ \quad + \frac{1}{2} (a + C_c(G_{x,y})) & \text{Case F: } i \geq a, i = b - a. \end{cases}$$

The values used in this formula are from Chapter 21.

We can actually rewrite this as in (14.1) to have

$$C_c(G, (y:i)) = \begin{cases} \frac{1}{2}(a+b-2i) + \frac{1}{2}C_c(G_y, (x:a-i)) \\ \quad + \frac{1}{2}C_c(G_y, (x:b-i)) & \mathbf{A}: i < a, i > b-a \\ \frac{1}{4}(3a+b-3i) + \frac{1}{2}C_c(G_y, (x:a-i)) \\ \quad + \frac{1}{4}C_c(G_x, (x:a+i)) \\ \quad + \frac{1}{4}C_c(G_y, (x:b-i)) & \mathbf{B}: i < a, i < b-a \\ b-i + C_c(G_y, (x:b-i)) & \mathbf{C}: i \geq a, i > b-a \\ \frac{1}{2}(a+b-i) + \frac{1}{2}C_c(G_x, (y:a+i)) \\ \quad + \frac{1}{2}C_c(G_y, (x:b-i)) & \mathbf{D}: i \geq a, i < b-a \\ \frac{1}{2}(2a-i) + \frac{1}{2}C_c(G_y, (x:a-i)) \\ \quad + \frac{1}{4}C_c(G_y, (x:a)) + \frac{1}{4}C_c(G_{x,y}) & \mathbf{E}: i < a, i = b-a \\ a + \frac{1}{2}C_c(G_y, (x:a)) + \frac{1}{2}C_c(G_{x,y}) & \mathbf{F}: i \geq a, i = b-a. \end{cases}$$

The equivalence of these six cases can be easily proven separately, all that needs to be done is to calculate the expected value of the minimum of the two distributions.

What we can see from the formulas is that, broken down to the smallest pieces, all appearing functions are either polynomials or rational functions in the variables a , b , and i , each with degree at most n , where n is the number of tasks. Each level of the recursion references the function C_c in six different ways, meaning that a fully executed case distinction on each level has almost 6^n different calls to the function C_c . Because of this property, this approach is not very efficient when regarding larger graphs, i.e., with more than 15 tasks.

Not only does the calculation of the expected makespan become more complicated than with exponential processing times, cp. Chapter 16 on page 36, but the resulting formulas are not scaled by the parameters a and/or b . With exponential processing times, we showed that the parameter λ does not matter for the calculation because it acts as a scaling factor. As a result, we can set it to 1 and simplify our efforts. For uniform processing times, the parameters a and b of the distribution really do matter and cannot be set arbitrarily to make life easier. Of course, this is not expected since $\mathcal{U}\{1, 3\}$ differs vastly from $\mathcal{U}\{7, 11\}$, for example. The same then holds for the following continuous case as well.

23.2 Continuous Case

We transfer this approach onto a continuous uniform distribution over an interval $[a, b]$, i.e., we consider the scheduling problem

$$2|p_i \sim \mathcal{U}[a, b]; \text{intree} | \mathbb{E}(C_{\max}).$$

Now that the distribution is continuous we have to deal with integrals instead of finite sums. In addition, we have that the probability of two tasks finishing at the same time — no matter what the distributions are — is zero. This case was captured in the cases **E** and **F** before. So, after starting processing in the first step, no schedule ever creates a time where both processors are idle until all tasks are finished.

Again, we distinguish between two different cases.

Case 1: The current configuration has only one successor configuration, i.e., it is a chain. Then, the expected makespan can be calculated by

$$C_c(G) = \frac{a+b}{2} \cdot n,$$

$$C_c(G, (y:i)) = \begin{cases} \frac{a+b}{2} - i + \frac{a+b}{2}(n-1) & i < a \\ \frac{b-i}{2} + \frac{a+b}{2}(n-1) & i \geq a. \end{cases}$$

Case 2: The current configuration has at least two successor configurations. At first, we consider the case without any active tasks, i.e., X and Y are identically distributed. Then, the expected value of the minimum, cp. (22.1), is

$$\mathbb{E}(X \wedge Y) = \frac{2a+b}{3}.$$

The values $\mathcal{P}(X \wedge Y = X)$ and $\mathcal{P}(X \wedge Y = Y)$ are easily calculated, as both cases are equally likely to occur. Altogether we get

$$C_c(G) = \frac{2a+b}{3} + \frac{1}{2} \int_a^b \frac{1}{b-a} C_c(G_x, (y:z)) dz + \frac{1}{2} \int_a^b \frac{1}{b-a} C_c(G_y, (x:z)) dz,$$

or, equivalently,

$$C_c(G) = \frac{1}{2} \int_a^b \frac{1}{b-a} (z + C_c(G_x, (y:z))) dz + \frac{1}{2} \int_a^b \frac{1}{b-a} (z + C_c(G_y, (x:z))) dz.$$

That these two formulas are in fact equivalent can be seen below in (23.1). For the more difficult case with an active task, we use the values from Chapter 22 and obtain

$$C_c(G, (y:i)) = \begin{cases} \int_{a-i}^{b-i} \frac{1}{b-a} (z + C_c(G_y, (x:z))) dz & \mathbf{A}: i < a, i > b-a \\ \int_{a-i}^a \frac{1}{b-a} (z + C_c(G_y, (x:z))) dz \\ \quad + \int_a^{b-i} \frac{1}{b-a} \cdot \frac{z-a+i}{b-a} (z + C_c(G_y, (x:z))) dz \\ \quad + \int_a^{b-i} \frac{1}{b-a} \cdot \frac{z-a}{b-a} (z + C_c(G_x, (y:i+z))) dz & \mathbf{B}: i < a, i \leq b-a \\ \int_0^{b-i} \frac{1}{b-i} (z + C_c(G_y, (x:z))) dz & \mathbf{C}: i \geq a, i > b-a \\ \int_0^a \frac{1}{b-i} (z + C_c(G_y, (x:z))) dz \\ \quad + \int_a^{b-i} \frac{1}{b-i} \cdot \frac{z-a+i}{b-a} (z + C_c(G_y, (x:z))) dz \\ \quad + \int_a^{b-i} \frac{1}{b-i} \cdot \frac{z-a}{b-a} (z + C_c(G_x, (y:i+z))) dz & \mathbf{D}: i \geq a, i \leq b-a \end{cases}$$

Like in the discrete case, the appearing functions inside the integrals are all either polynomial or rational, with degrees not higher than $2n$. The 2 in here is because of cases **B** and **D** where there is a quadratic function (in the variables a and b) in the denominator. The number of integrals grows by a constant number with each level, i.e., the number of nested integrals is linear (in the size of the graph). However, the number of cases or possibilities to be considered grows exponentially in the continuous case, too. In the case distinction, there are 8 recursive references to the function C_c with 2 different sets of values. This makes the base of the exponent for the number of possibilities smaller, but it is exponential nonetheless, implying that a computational approach using this formula is not really efficient.

For the exponential distribution we have already argued that the two different approaches from (15.1) and (15.2) on page 32 are the same, see (16.3) on page 37. We can show this for the

continuous uniform distribution, too:

$$\begin{aligned}
(15.2) &= \frac{1}{2} \int_a^b f_{X \wedge Y}(x) (x + C_c(c_x)) dx + \frac{1}{2} \int_a^b f_{X \wedge Y}(x) (x + C_c(c_y)) dx \\
&= \int_a^b \frac{2(b-x)}{(b-a)^2} x dx + \frac{1}{2} (C_c(c_x) + C_c(c_y)) \int_a^b \frac{2(b-x)}{(b-a)^2} dx \\
&= \frac{2a+b}{3} + \frac{1}{2} (C_c(c_x) + C_c(c_y)) \cdot 1 \\
&= (15.1) \tag{23.1}
\end{aligned}$$

Of course, this covers only the simple case, where there are no active tasks and at least two tasks are available. The other cases can be proven similarly, although with more complexity. The interesting part about calculating the expected makespan is the way how the different distributions are handled. In our approach, we adjusted the range of the active task's distribution to start at $\min(0, a - i)$. However, we could have defined and calculated the expected makespan with the same distributions all along, and adjust the limits of the integrals to contain the expected value of the minimum. Both approaches will actually lead to the same result, but we will use only the one approach presented above for the sake of consistency. Some exemplary identities showing that the different approaches are in fact equivalent can be seen in [22].

24 HLF and Uniform Processing Times

In this section, we consider the scheduling problem

$$2|p_i \sim \mathcal{U}\{a, b\};intree|\mathbb{E}(C_{\max}) \cdot, \tag{24.1}$$

In order to understand how to deal with uniform processing times, we want to have a look at the **deterministic** scheduling problem

$$2|p_i \in \{1, 2\};intree|C_{\max}. \tag{24.2}$$

The processing time of each task is known before and is not subject to stochastic fluctuation, it is either one or two. We find this helpful in understanding the uniform processing times because these are not memoryless and the tasks' expected remaining processing times are not identical anymore as they were in the case with exponential processing times, see (16.1) and (16.2). The idea is that if this deterministic problem is not optimally solved by *HLF*, then the stochastic uniform problem (24.1) for $a = 1$ and $b = 2$ probably is not either.

Nakajima, Leung, and Hakimi [44] showed that the problem (24.2) is polynomially solvable by a method which they called *Largest Subtree First (LSF)*. The method itself is similar to *HLF* as it is a greedy method, too. The problem is that *LSF* alone does not result in an optimal schedule, that is why there is another step in the algorithm that repairs a possible suboptimal choice made

by *LSF* before. Although the difference is small, the optimal strategy proposed in [44] is no longer greedy and thus, vastly different from *HLF*. Another issue with the strategy is that it is originally designed to solve (24.2) but with outtrees¹ instead of intrees. This poses no problem in the deterministic case because the optimal schedule for an outtree can just be reversed to provide an optimal schedule for the corresponding (reversed) intree. But it seems that there is no immediate analogous method for stochastic schedules that translates an optimal schedule for an outtree into one for the corresponding intree. To sum up, although we cannot rule out *HLF* or something similar, we suspect that at least the *LSF* method does not help in the stochastic case.

On the other hand, Du and Leung [18] showed that the version of (24.2) with the processing times being integer powers of an integer is **NP**-hard. This may point in the direction that for arbitrary parameters a and b for the uniform distribution the stochastic scheduling problem (24.1) is **NP**-hard as well.

Moreover, there are proofs that (24.2) but with an arbitrary number of processors is **NP**-hard [18], as well as a variant with *dag* precedences [55, 56]. This may imply that the corresponding variants of the stochastic problem are **NP**-hard too, but this is a topic for future work and is not discussed further here. The analysis of (24.1) is elaborated enough.

What follows are some observations that may indicate that *HLF* is not the optimal strategy for this problem. If it were, we would suspect the results below to hold.

Observation 65. Let G be an intree and also represent a configuration with an active task s . G' is obtained by setting another source on a lower level to active and making s non-active. Then no statements can be made about the comparability of the expected makespans of G and G' .

Proof. Consider the case with $a = 3$ and $b = 7$, then it holds that

$$C_c \left(\begin{array}{c} 2 \\ \otimes \\ \bullet \\ \bullet \end{array} \right) = \frac{154}{8} > \frac{120}{8} = C_c \left(\begin{array}{c} 2 \\ \bullet \\ \bullet \\ \bullet \end{array} \right),$$

but

$$C_c \left(\begin{array}{c} 1 \\ \otimes \\ \bullet \\ \bullet \end{array} \right) = \frac{66}{4} < \frac{90}{4} = C_c \left(\begin{array}{c} 1 \\ \bullet \\ \bullet \\ \bullet \end{array} \right),$$

where the crossed nodes in the intrees correspond to active tasks. The number next to an active task describes the time this task has already been processed. \square

In the cases of exponential or geometric processing times, the above result would actually hold because we can disregard active tasks. Thus, choosing a source on a lower level would always be worse than choosing the higher source.

Note that this does not rule out *HLF* as the optimal strategy. The provided counterexamples above might actually never occur during an *HLF* schedule.

Similarly to Observation 65, we can show the following.

¹An acyclic, connected graph where every node has indegree at most 1 (compared to outdegree at most 1 for an outtree).

Observation 66. Let G be an intree and also represent a configuration with an active task s . G' is obtained by removing s and attaching it to a lower level, but still in HLF range, and letting it stay active. Then no statements can be made about the comparability of the expected makespans of G and G' .

Proof. Consider the case with $a = 3$ and $b = 8$. It holds that

$$C_c \left(\begin{array}{c} 1 \\ \otimes \\ \bullet \\ \bullet \end{array} \right) = \frac{70}{4} < \frac{101}{4} = C_c \left(\begin{array}{c} 1 \\ \bullet \\ \otimes \\ \bullet \end{array} \right).$$

But with $a = 7$ and $b = 16$, it holds that

$$C_c \left(\begin{array}{c} 3 \\ \otimes \\ \bullet \\ \bullet \end{array} \right) = \frac{145}{4} > \frac{138}{4} = C_c \left(\begin{array}{c} 3 \\ \bullet \\ \otimes \\ \bullet \end{array} \right).$$

□

Take note, that the two configurations G and G' are not to be considered as two subtrees that occur in the same schedule. For Observation 65 for example, this can only be the case when preemption is allowed anyway. The idea behind these observations is to consider G and G' as two configurations from two different schedules, but at comparable decision points, i.e., they have the same number of tasks. What we want to achieve with these observations is to either construct a proof that HLF is an optimal strategy for this scheduling problem, or determine a counterexample that HLF is not optimal. To this end, we consider the approach from Chapters 17 and 20 on pages 38 and 94 which we used to show that HLF is optimal. The first step in our proofs was showing that “a flatter intree has a smaller makespan” by attaching a source on a lower level and comparing makespans. In the case of uniform processing time where active tasks have to be considered, this has to be true for several cases with and without active tasks. Observations 65 and 66 show counterexamples for two of those cases. Though this is not enough to dismiss HLF as the possible optimal strategy, at least we can say that the approach to show this has to be different from the one we used for exponential and geometric processing times.



The Erlang Distribution

Table of Contents

25 Calculating the Expected Makespan	116
26 The Generalized Erlang Distribution	120
27 <i>HLF</i> and Erlang processing times	121

The Erlang distribution, named after Danish mathematician Anger Krarup Erlang (1878-1929), is another distribution that we want to consider as the distribution for the tasks' processing times. The Erlang distribution can be modeled as a sum of exponential distributions, giving it a reasonable interpretation to be considered in the first place.

Let the tasks be divided into several smaller parts/(sub-)tasks that all have stochastic processing times, but have to be processed in succession in order to completely finish the larger task. If we consider the processing times of the smaller parts of the tasks to be exponentially distributed, then the large tasks themselves are Erlang distributed. Formally, we have the following definition, taken from [32].

Definition 67. A continuous random variable X with support \mathbb{R}^+ is *Erlang distributed* with shape $k \in \mathbb{N}$ and rate $\lambda > 0$, denoted by $X \sim E_k(\lambda)$, if its probability density function f_X is

$$f_X(z) = \begin{cases} \frac{\lambda^k z^{k-1}}{(k-1)!} e^{-\lambda z} & z \geq 0 \\ 0 & z < 0, \end{cases}$$

that is, its cumulative distribution function is

$$F_X(z) = \begin{cases} 1 - \sum_{j=0}^{k-1} \frac{(\lambda z)^j}{j!} e^{-\lambda z} & z \geq 0 \\ 0 & z < 0. \end{cases}$$

Sometimes in literature, the equivalent *scale* parameter $\mu = \frac{1}{\lambda}$ is used in the definition of the Erlang distribution. The expected value of a Erlang distributed random variable $X \sim E_k(\lambda)$ is

$$\mathbb{E}(X) = \frac{k}{\lambda} = k\mu.$$

As we can easily see, the special case $k = 1$ yields the exponential distribution with parameter λ . But, as mentioned before, we can actually show that each Erlang distribution can be seen as the sum of several identically exponentially distributed random variables. Formally, we have the following result.

Lemma 68. Let X_1, \dots, X_k be independent, identically exponentially distributed random variables with parameter λ . Then their sum is Erlang distributed with shape k and rate λ , i.e.,

$$X_i \sim \exp(\lambda) \Rightarrow \sum_{i=1}^k X_i \sim E_k(\lambda).$$

The proof uses the fact that the density (probability density function) of the sum of two or more independent random variables is exactly the convolution of their densities [32]. Here, this means that we have to calculate the convolution of the k exponential densities. This only involves some basic calculations and induction, and is left out here.

Using either this result or the definition directly, we can show that the Erlang distribution scales in the same way that the exponential distribution does, see Part V:

$$X \sim E_k(\lambda) \Rightarrow \alpha \cdot X \sim E_k\left(\frac{\lambda}{\alpha}\right) \text{ for all } \alpha > 0. \quad (24.3)$$

This means that we may restrict ourselves to the simple case with $\lambda = 1$ again, and simplify our calculations that way.

Using the same calculations as in the proof of Lemma 68 we can show that the sum of two Erlang distributions with the same rate yields an Erlang distribution where its shape is the sum of the shapes before, i.e.,

$$X \sim E_k(1), Y \sim E_\ell(1) \Rightarrow X + Y \sim E_{k+\ell}(1).$$

In other words: Erlang densities are *closed under convolutions*, meaning that the convolution of two Erlang densities again results in an Erlang density.

The Erlang distribution can also be generalized to non-integral shape parameters. The resulting distribution is called *Gamma distribution*, see for example [24]. We do not use this distribution here, the interpretation seems too far-fetched to consider it. What we will use, however, is the *incomplete Gamma function* as a notational shorthand in our calculations later on. For this we have the following definition, taken from [4], for example.

Definition 69. The (upper) *incomplete Gamma function* $\Gamma : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{R}$ is defined as the integral

$$\Gamma(k, z) = \int_z^\infty t^{k-1} e^{-t} dt.$$

In general, the parameters k and z can be complex numbers, but in the same way that we use the Erlang distribution as the specialized version of the Gamma distribution, we only use integral k and real-valued z , too. In that case, we even have the following equality:

$$\Gamma(k, z) = (k-1)! \cdot \sum_{j=0}^{k-1} \frac{z^j}{j!} e^{-z} \text{ for } k \in \mathbb{N}.$$

The equivalence that will be used most often throughout this chapter is

$$\sum_{j=0}^{k-1} \frac{(\lambda z)^j}{j!} e^{-\lambda z} = \frac{\Gamma(k, \lambda z)}{(k-1)!}.$$

25 Calculating the Expected Makespan

Despite the Erlang distribution being very similar to the exponential distribution – actually, it can be seen as a generalization of it – we cannot recreate the same calculations with the same ease

as we did in Part V. Mostly, this is due to the Erlang distribution not being memoryless while the exponential distribution has this very nice attribute. Other than that, the calculations can be carried out in the same way, with straight-forward probability theoretical concepts and the like, as seen in Part VII for example.

Let X and Y be identical Erlang distributed random variables with shape $k \in \mathbb{N}$ and rate $\lambda > 0$, i.e., $X, Y \sim E_k(\lambda)$. The probability density function and cumulative distribution function of X and Y , respectively, are

$$f_X(z) = \begin{cases} \frac{\lambda^k z^{k-1}}{(k-1)!} e^{-\lambda z} & z \geq 0 \\ 0 & z < 0, \end{cases}$$

$$F_X(z) = \begin{cases} 1 - \sum_{j=0}^{k-1} \frac{(\lambda z)^j}{j!} e^{-\lambda z} = 1 - \frac{\Gamma(k, \lambda z)}{(k-1)!} & z \geq 0 \\ 0 & z < 0, \end{cases}$$

cp. Definition 67.

Then, for the minimum $X \wedge Y$, we have

$$F_{X \wedge Y}(z) = 1 - \left(\sum_{j=0}^{k-1} \frac{(\lambda z)^j}{j!} e^{-\lambda z} \right)^2 = \left(\frac{\Gamma(k, \lambda z)}{(k-1)!} \right)^2$$

and

$$\begin{aligned} \mathbb{E}(X \wedge Y) &= \int_0^{\infty} \left(\sum_{j=0}^{k-1} \frac{(\lambda z)^j}{j!} e^{-\lambda z} \right)^2 dz \\ &= \frac{1}{2\lambda} \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} \frac{1}{2^{i+j}} \binom{i+j}{i}, \end{aligned}$$

where we used Lemma 24 on page 27 for calculating the expected value.

The more general case is when a task is active and has already been processed for some t time units, $t > 0$. The total processing time of that task can be calculated by using conditional probabilities again, like we did in Part VII. There, the remaining processing time was still uniformly distributed, albeit with different parameters. In contrast to that, the remaining processing time here is not even Erlang distributed.

Let \bar{Y} be the total processing time, and Y the remaining processing time, given that the task has already been processed for $t > 0$ time units. Then, for $z \geq 0$, we have

$$\begin{aligned}
 F_Y(z) &= \mathcal{P}(Y \leq z) \\
 &= \mathcal{P}(\bar{Y} \leq z + t \mid \bar{Y} > t) \\
 &= 1 - \mathcal{P}(\bar{Y} > z + t \mid \bar{Y} > t) \\
 &= 1 - \frac{\mathcal{P}(\bar{Y} > z + t, \bar{Y} > t)}{\mathcal{P}(\bar{Y} > t)} \\
 &= 1 - \frac{1 - \mathcal{P}(\bar{Y} \leq z + t)}{1 - \mathcal{P}(\bar{Y} \leq t)} \\
 &= 1 - \frac{\Gamma(k, \lambda(z + t))}{\Gamma(k, \lambda t)}, \tag{25.1}
 \end{aligned}$$

$$\begin{aligned}
 f_Y(z) &= \frac{\lambda^k (z + t)^{k-1} e^{-\lambda(z+t)}}{\Gamma(k, \lambda t)}, \\
 \mathbb{E}(Y) &= \int_0^\infty \frac{\Gamma(k, \lambda(z + t))}{\Gamma(k, \lambda t)} dz. \tag{25.2}
 \end{aligned}$$

Consider the random variable \bar{Y} , conditioned on $\bar{Y} > t$, then we have

$$\mathbb{E}(Y) = \frac{\Gamma(k + 1, \lambda t)}{\lambda \Gamma(k, \lambda t)} - t = \mathbb{E}(\bar{Y} \mid \bar{Y} > t) - t,$$

which is the desired result because Y describes the remaining processing time after the first t time units.

Even though Y is no longer Erlang distributed, the minimum $X \wedge Y$ can be calculated and we get

$$\begin{aligned}
 F_{X \wedge Y}(z) &= \begin{cases} 1 - \frac{\Gamma(k, \lambda z)}{(k-1)!} \cdot \frac{\Gamma(k, \lambda(z+t))}{\Gamma(k, \lambda t)} & z \geq 0 \\ 0 & z < 0, \end{cases} \\
 f_{X \wedge Y}(z) &= \begin{cases} \frac{\lambda^k e^{-\lambda z}}{(k-1)! \Gamma(k, \lambda t)} ((z+t)^{k-1} \Gamma(k, \lambda z) e^{-\lambda t} + z^{k-1} \Gamma(k, \lambda(z+t))) & z \geq 0 \\ 0 & z < 0, \end{cases} \\
 \mathbb{E}(X \wedge Y) &= \frac{1}{(k-1)! \Gamma(k, \lambda t)} \int_0^\infty \Gamma(k, \lambda z) \Gamma(k, \lambda(z+t)) dz. \tag{25.3}
 \end{aligned}$$

For the expected makespan of a configuration $(G, (y:t))$, i.e., with underlying intree G where the task y has been active for t time units, we have the recursive formula

$$\begin{aligned} C_c(G, (y:t)) = \mathbb{E}(X \wedge Y) &+ \int_0^\infty \underbrace{\mathcal{P}(X \wedge Y = X)}_{F_X(z)} f_{X \wedge Y}(z) C_c(G_x, (y:t+z)) dz \\ &+ \int_0^\infty \underbrace{\mathcal{P}(X \wedge Y = Y)}_{F_Y(z)} f_{X \wedge Y}(z) C_c(G_y, (x:t)) dz, \end{aligned} \quad (25.4)$$

with G_x and G_y as before. The fact that $\mathcal{P}(X \wedge Y = X) = F_X(z)$ if the integration variable is z is because of

$$\begin{aligned} \mathcal{P}(X \wedge Y = X) &= \mathcal{P}(t_x < t_y) \quad \forall t_y \\ &= F_X(t_y) \quad \forall t_y, \end{aligned}$$

with the same notation and arguments already seen in Part VII. In the same manner, we have $\mathcal{P}(X \wedge Y = Y) = F_Y(z)$ for the integration variable z .

For smaller examples it can also be helpful to compute the expected value of the maximum $X \vee Y$. Then, for $X \sim Y$, we have

$$\mathbb{E}(X \vee Y) = 2 \frac{k}{\lambda} - \frac{1}{2\lambda} \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} \frac{1}{2^{i+j}} \binom{i+j}{i}.$$

If X and Y are distributed differently, in particular, Y models the expected remaining processing time and is distributed as in (25.1), then

$$\mathbb{E}(X \vee Y) = \int_0^\infty 1 - \left(1 - \frac{\Gamma(k, \lambda z)}{(k-1)!}\right) \left(1 - \frac{\Gamma(k, \lambda(z+t))}{\Gamma(k, \lambda t)}\right) dz. \quad (25.5)$$

The integrals in (25.5), as well as in (25.2) and (25.3) can be calculated, but for plugging in values for k , t and λ , the formulas with the integrals left untouched produce more satisfying results.

Setting $k = 1$ in all these expressions yields the results already seen in Part V while setting $t = 0$ yields the expressions for $X \sim Y$, of course. This can be retraced with the **Mathematica** code in Listing 25.1. We use the function `Inactivate` to prevent **Mathematica** from evaluating the expressions in some of the definitions as they (probably) cannot be simplified any further, but will otherwise be attempted anyway. `Activate` then allows the evaluation, which is easy for constant values of k and t .

Lines 1 to 4 show that, for $k = 1$, the Erlang distribution with rate λ and the exponential distribution with parameter λ are identical. Lines 7 to 10 show the expected value of the minimum and the maximum of two identically Erlang distributed random variables, and their values for $k = 1$. Lines 12 to 15 are the probability density function and cumulative distribution function of Y for $\bar{Y} > t$ and $\bar{Y} \sim E_k(\lambda)$, and the corresponding functions for the exponential distribution. We

```

1  fX[k_,z_] = lambda*z^(k-1)*Exp[-lambda*z]/(k-1)!
2  FX[k_,z_] = 1-Sum[(lambda*z)^j*Exp[-lambda*z]/j!, {j,0,k-1}]
3  fX[1,z]
4  FX[1,z]
5
6  (* X~Y~E_k(lambda) *)
7  EXminX[k_] = 1/(2*lambda)*Sum[Sum[2^(-i-j)*Binomial[i+j,i], {i,0,k-1}], {j,0,k-1}]
8  EXmaxX[k_] = 2*k/lambda-1/(2*lambda)*Sum[Sum[2^(-i-j)*Binomial[i+j,i], {i,0,k-1}],
    {j,0,k-1}]
9  EXminX[1]
10 EXmaxX[1]
11
12 fY[k_,z_,t_] = lambda^k*(z+t)^(k-1)*Exp[-lambda*(z+t)]/Gamma[k,lambda*t]
13 FY[k_,z_,t_] = 1-Gamma[k,lambda*(z+t)]/Gamma[k,lambda*t]
14 fY[1,z,0]
15 FY[1,z,0]
16
17 (* X~E_k(lambda), Y~above *)
18 fXminY[k_,z_,t_] = Inactivate[ lambda^k*Exp[-lambda*z]/((k-1)!*Gamma[k,lambda*t])*
    (z+t)^(k-1)*Gamma[k,lambda*z]*Exp[-lambda*z]+z^(k-1)*Gamma[k,lambda*(z+t)] ] ]
19 FXminY[k_,z_,t_] = Inactivate[ 1 - (Gamma[k,lambda*z]/(k-1)!) *
    (Gamma[k,lambda*(z+t)]/Gamma[k,lambda*t]) ] ]
20 EXminY[k_,t_] = Inactivate[ 1/((k-1)!*Gamma[k,lambda*t]) *
    Integrate[Gamma[k,lambda*z]*Gamma[k,lambda*(z+t)],{z,0,Infinity}] ] ]
21 EXmaxY[k_,t_] = Inactivate[ Integrate[1 - (1-Gamma[k,lambda*z]/(k-1)!) *
    (1-Gamma[k,lambda*(z+t)]/Gamma[k,lambda*t]), {z,0,Infinity}] ] ]
22 Activate[fXminY[1,z,0]]
23 Activate[FXminY[1,z,0]]
24 Activate[EXminY[1,0]]
25 Activate[EXmaxY[1,0]]

```

Listing 25.1: **Mathematica** code for reproducing the formulas from Part V by setting $k = 1$ and/or $t = 0$ in the formulas from above.

set $t = 0$ because the exponential distribution is memoryless. However, leaving t as a variable works as well because it cancels out in the simplified functions anyway. At last, Lines 18 to 25 show the expressions for the minimum, maximum, and the expected values. First, for an Erlang distribution X and a distribution Y as above in Lines 12 to 15, and finally, for two exponential distributions, again with t set to 0 because of memorylessness.

26 The Generalized Erlang Distribution

The Erlang distribution can be generalized in the sense that the exponential distributions used in the sum do not have to be identical. Doing this, we can consider the *hypoexponential distribution* (or *generalized Erlang distribution*) which is the sum of several exponential distributions, each with their own rate λ_i as opposed to an Erlang distribution which is the sum of several identical exponential distributions. This means that X is a hypoexponential distribution with parameters

$\lambda_1, \dots, \lambda_k$ if

$$X = \sum_{i=1}^k X_i$$

with $X_i \sim \exp(\lambda_i)$. The simpler cases with X being the sum of only two exponential distributions can be managed more easily than the general case with arbitrary k . Still, as the Erlang distribution poses enough problems already, we do not want to focus further on even more generalizations.

27 HLF and Erlang processing times

Of course, we already know from Theorem 27 on page 36 that the Erlang distribution is not memoryless, but the question is: Is it close enough to the exponential distribution that we can still have similar results? Before we answer this question we first have to observe that each task with an Erlang distributed processing time with shape k and rate λ can be split up into a chain of k tasks that are independent, identically exponentially distributed with parameter λ . For example, see Fig. 44 with $k = 3$. Each task can be interpreted as a chain of three tasks with exponential processing times with parameter λ . For instance, the tasks v_1 and v_4 are now reduced to the two chains of green tasks on the right, respectively.

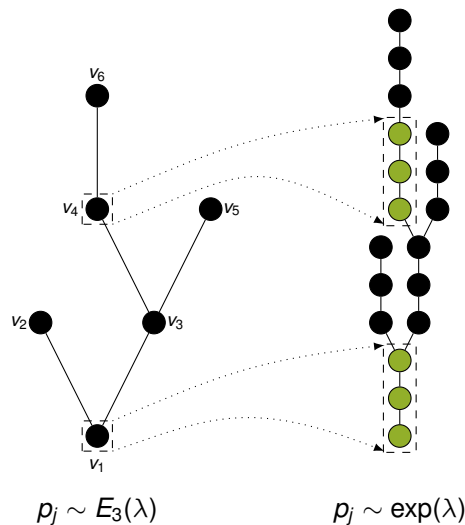


Fig. 44: The reduction from Erlang to exponential processing times.

This fits the interpretation that Erlang distributed random variables are just sums of several identically exponentially distributed random variables. We call the k successive parts of a task the *phases* of a task. Then, the big difference in handling this model is whether the scheduling strategy knows at which phases the currently active tasks are or not. If the strategy knows this,

then we can instantly reduce the scheduling problem

$$2|p_j \sim E_k(1); \text{intree} | \mathbb{E}(C_{\max})$$

to the well-known problem with exponentially distributed processing times, using the reduction seen in Fig. 44. This means that, again, *HLF* is optimal even for Erlang distributions. The more interesting case is when the scheduling strategy does not know the current phases of the active tasks. Then we cannot just reduce the problem to exponentially distributed tasks.

Chandy and Reynolds [12] claim that even in this case *HLF* is optimal, but give only a sketch of the proof for it. Apart from this, we could not find any proof in scientific publications about this scenario. As the proof is not found anymore in the scientific literature, we are a bit skeptical about this, and consider this problem to be open.

We try to approach the problem in the same way that we did in Part V:

- First, we have to generalize the flatness relation from Chapter 17 on page 38 to configurations with active tasks.
- Then, we establish that with *HLF* a flatter intree always has a smaller makespan than one that is not as flat.
- After that, we show that exactly one non-*HLF* step is worse than none.
- At last, we prove that a non-*HLF* applied to an intree can never result in a smaller makespan than an *HLF* to a flatter intree, in turn showing that more than one non-*HLF* is not improving the makespan either.

The flatness relation from Chapter 17 that works well for the case with exponential processing times, does not cover all necessary cases when dealing with Erlang processing times. The main issue is with active tasks and how to compare two configurations that are structurally identical (i.e., the underlying intree is the same) but have different active tasks and/or different times describing how long the active tasks have already been processed. To this end, we redefine flatness as follows, cp. [12].

Definition 70. The flatness is a partial relation on configurations, defined by:

- As before for configurations with no active tasks, cp. Definition 30 on page 41.
- A configuration $(G, (x:t))$ is flatter than a configuration H if $t > 0$, $G \preceq H$, and H has no active tasks.
- A configuration $(G, (x:t))$ is flatter than a configuration $(H, (y:s))$ if $s, t > 0$ and

$$G \preceq H \text{ and } G \setminus \{x\} \preceq H \setminus \{y\}.$$

The part where we have to do a lot more work than in Chapter 17 is the proof showing that flatter intrees are always better. If *HLF* is indeed optimal even for Erlang processing times, then the proof works in the same manner as in Lemma 31 on page 41, only that the expressions become much more involved and are only calculated fast enough by using **Mathematica**.

In Chapter 17, we showed that an intree which is changed by one task and is flatter results in a smaller makespan than before. Because of the memorylessness, we did not have to consider active tasks for the intrees/configurations. In the case of Erlang processing times, we have to make this consideration. This is why, in the following, we have different conjectures, each describing one particular case, instead of only one result in total. Also, the following conjectures are about configurations, whereas in Lemma 31 we considered intrees. But for exponential processing times, we argued that these two concepts are equivalent anyway.

We give proof ideas for the following conjectures that work in the same way as the proof for Lemma 31, the harder parts are again the calculations to be done for the induction base cases. These are vastly more complex for processing times according to an Erlang distribution than for the other distributions we considered before.

Conjecture 71. *Let c be a configuration with no active sources and let c' be obtained by attaching a source of c on a lower level. Then $C_c(c) \geq C_c(c')$.*

Proof Idea. We prove this by induction on the number of tasks.

Induction Base: For the sake of easier readability, the base case of this is found in Appendix F. What can be seen is that this is the problematic part in this proof idea. Although the formulas from Chapter 25 seem sound and are checked by **Mathematica**, see Listing F.1 in Appendix F on page 194, we see that these do not work properly for this application. Unfortunately, we did not manage to find other approaches that prove this base case (the same holds for the base cases of the following conjectures).

Induction Step: Let n be the number of tasks in both the underlying graphs of $c = G$ and $c' = G'$. Then, the situation is as in Fig. 45. The two outer successor configurations $(G_s, (x:t))$ and $(G'_y, (x:t))$ both have underlying intrees of only $n - 1$ tasks and thus, can be handled by a induction hypothesis. However, as these configurations now have active tasks, these are cases for the induction hypothesis in Conjecture 73. The same holds for the two inner configurations $(G_x, (s:t))$ and $(G'_x, (y:t))$. They can be handled by the induction hypothesis of Conjecture 75.

Conjecture 72. *Let c be a configuration with an active source and let c' be obtained by attaching an active source of c on a lower level in *HLF* range while staying active. Then $C_c(c) \geq C_c(c')$.*

Proof Idea. We prove this by induction on the number of tasks.

Induction Base: See the explanation in Conjecture 71.

Induction Step: Let n be the number of tasks in both the underlying graphs of $c = G$ and $c' = G'$. Then, the situation is as in Fig. 46. The two outer successor configurations $(G_s, (x:t))$ and

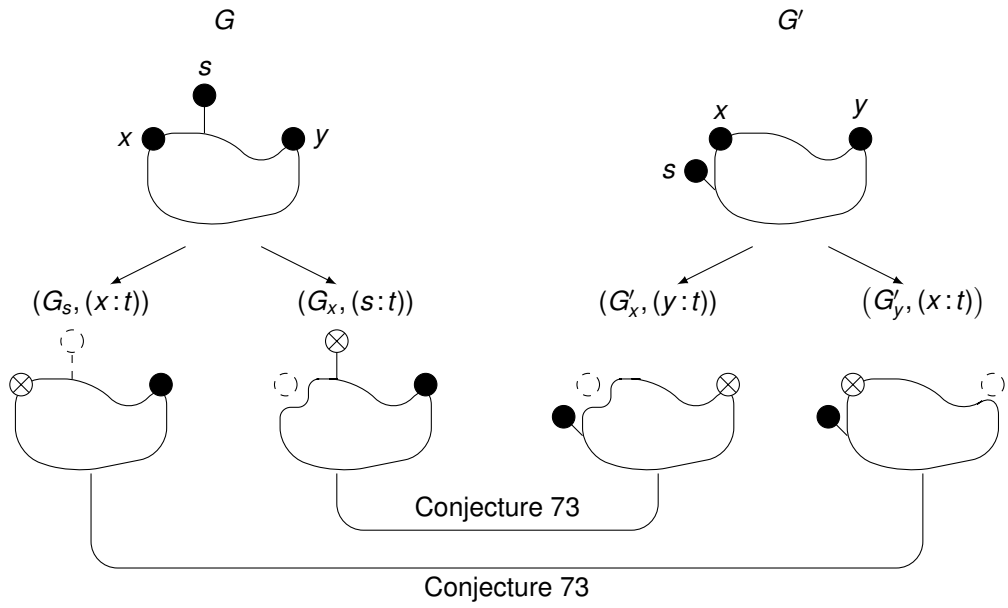


Fig. 45: The configurations G and G' and their respective successor configurations.

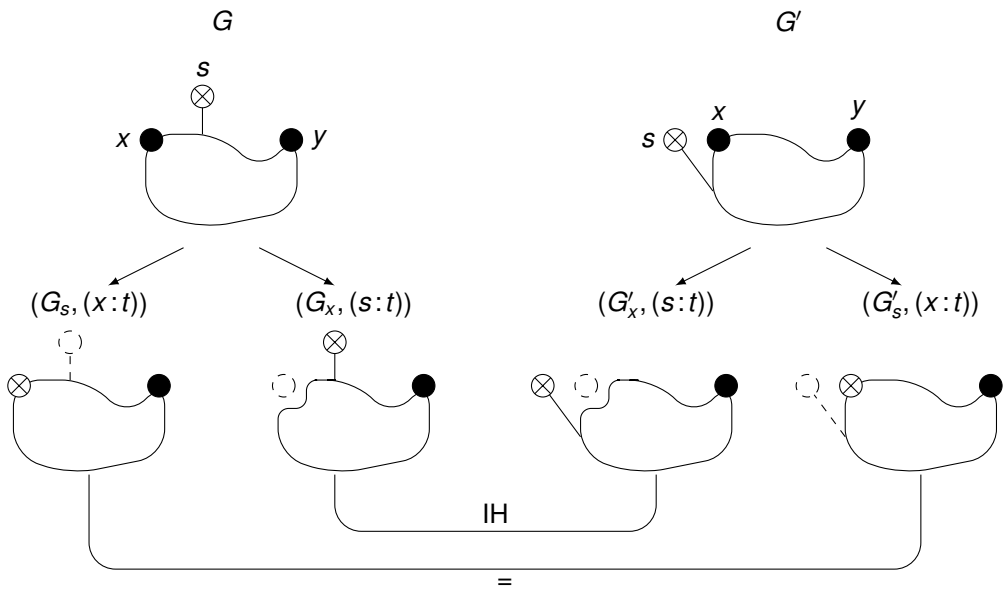


Fig. 46: G , G' , and their successor configurations.

$(G'_s, (x:t))$ are identical and thus, have the same expected makespan. The two inner successor configurations $(G_x, (s:t))$ and $(G'_x, (s:t))$ each $n - 1$ tasks each and can be handled by the induction hypothesis as the active task s of the second configuration (on the bottom) is just reattached on a lower level but stays active in the third configuration. This means, that this lemma is self-contained and does not need any of the other lemmas to be proven.

Conjecture 73. *Let c be a configuration with an active source and let c' be obtained by attaching an active source of c on a lower level and making another source in HLF range active. Then $C_c(c) \geq C_c(c')$.*

Proof Idea. We prove this by induction on the number of tasks.

Induction Base: See the explanation in Conjecture 71.

Induction Step: Let n be the number of tasks in both the underlying graphs of $c = G$ and $c' = G'$. Then, the situation is as in Fig. 47. The two outer successor configurations $(G_s, (x:t))$

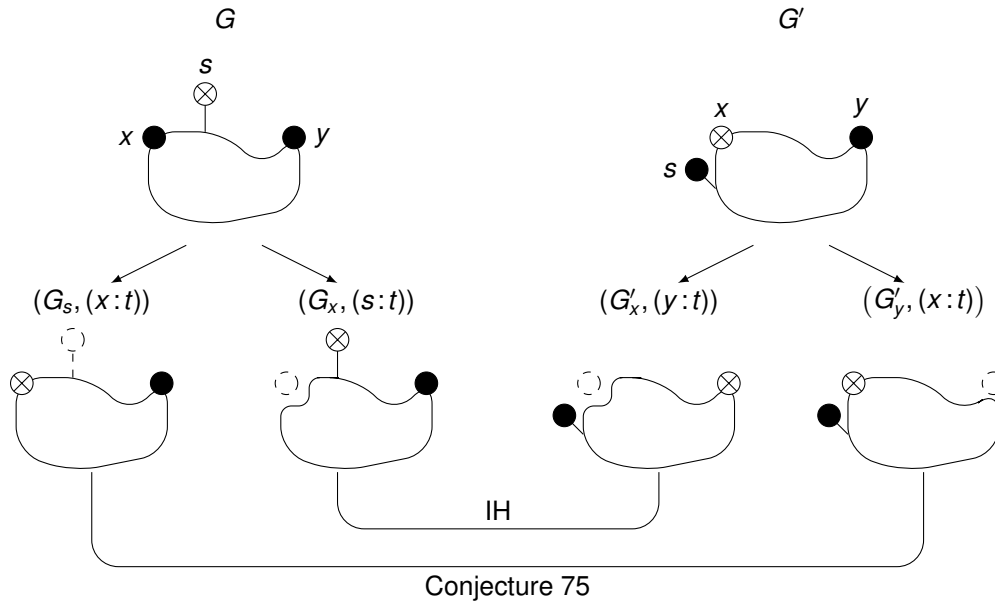


Fig. 47: G , G' , and their successor configurations.

and $(G'_y, (x:t))$ each have $n - 1$ tasks and can be handled by the induction hypothesis of Conjecture 75. This is because the non-active source y in the first configuration is just reattached to a lower level to obtain the fourth configuration. The two inner successor configurations $(G_x, (s:t))$ and $(G'_x, (y:t))$ can be handled by the IH as the active source s is reattached on a lower level while another task in HLF range y is made active.

Conjecture 74. *Let c be a configuration with an active source and let c' be obtained by attaching a non-active source of c on a lower level in HLF range and exchanging its status with the (former) active source, making it active and the other source non-active. Then $C_c(c) \geq C_c(c')$.*

Proof Idea. We prove this by induction on the number of tasks.

Induction Base: See the explanation in Conjecture 71.

Induction Step: Let n be the number of tasks in both the underlying graphs of $c = G$ and $c' = G'$. Then, the situation is as in Fig. 48. The situation here is actually the same as in

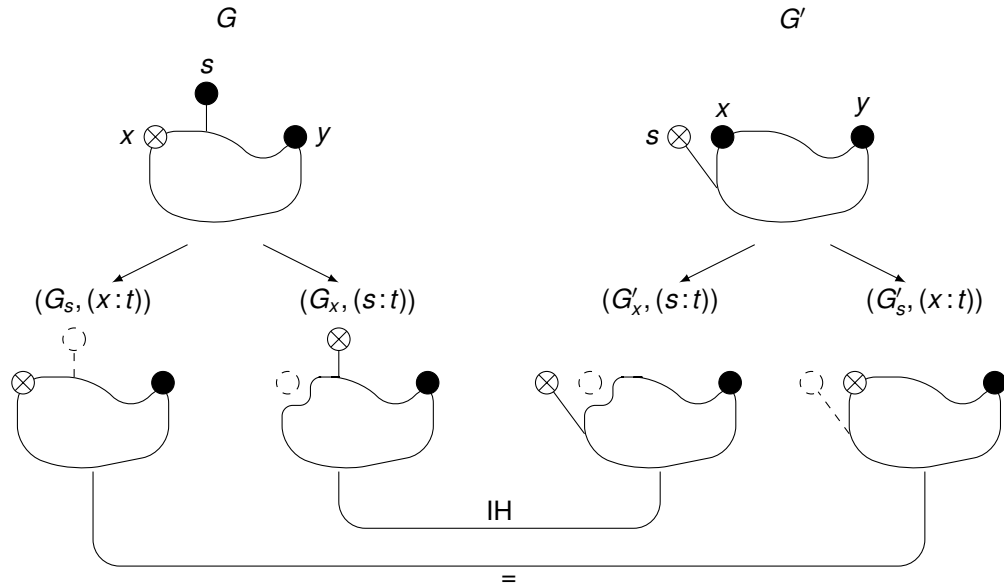


Fig. 48: G , G' , and their successor configurations.

Conjecture 72. The arguments for the proof are then identical as well, meaning that this lemma is self-contained, too.

Conjecture 75. Let c be a configuration with an active source and let c' be obtained by attaching a non-active source of c on a lower level. Then $C_c(c) \geq C_c(c')$.

Proof Idea. We prove this by induction on the number of tasks.

Induction Base: See the explanation in Conjecture 71.

Induction Step: Let n be the number of tasks in both the underlying graphs of $c = G$ and $c' = G'$. Then, the situation is as in Fig. 49. The two outer successor configurations $(G_s, (x:t))$ and $(G'_y, (x:t))$ each have $n-1$ tasks and can be handled by the induction hypothesis. The two inner successor configurations $(G_x, (s:t))$ and $(G'_x, (y:t))$ can be handled by the IH of Conjecture 73 because the active task s in the second configuration is just reattached to a lower level while another source in *HLF* range is made active to obtain the third configuration.

Again, we emphasize that the comparisons we make between the configurations c and c' in each of Conjectures 71 to 75 do not correspond to actual choices made by a strategy to produce a schedule. In particular, c and c' may not even be part of the same configuration graph.

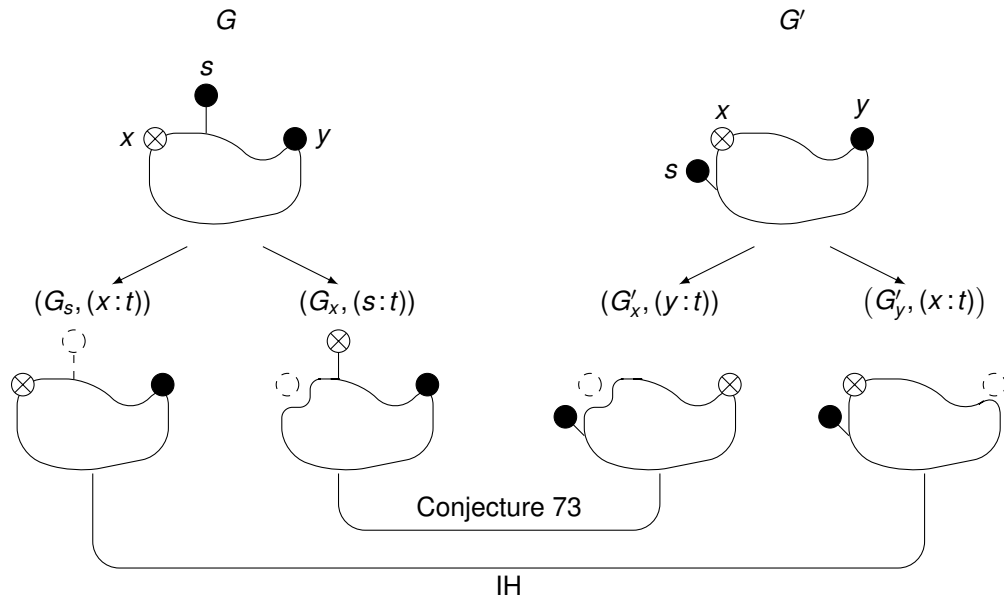


Fig. 49: G , G' , and their successor configurations.

These comparisons are purely theoretical and display the relationship of several, slightly different intrees.



General Precedence Constraints

Table of Contents

28 Directed Acyclic Graphs	131
28.1 The Coffman-Graham Algorithm is not Optimal	131
28.2 The Chandy-Reynolds Algorithm is not Optimal	137
28.3 Static, Semi-Static, and Dynamic Scheduling Strategies	139
28.4 Choosing Pairs of Sources	149
29 Calculating the Expected Makespan	154
30 sp-graphs	157

28 Directed Acyclic Graphs

In 1972, Coffman and Graham [14] proposed an optimal algorithm for the deterministic analog of the scheduling problem from Chapter 17, meaning that they proved their algorithm to be optimal for a two-processor scheduling problem withintree precedence constraints when considering makespan minimization if the tasks' processing times are identical and deterministic. Furthermore, not only did they specialize on intrees, but proved optimality of their algorithm for general *dags* (directed acyclic graphs), i.e., they solved the scheduling problem

$$2|p_i = 1; prec|C_{\max}. \quad (28.1)$$

Even though Chandy and Reynolds have probably known about this work in 1975, they could only solve the stochastic problem withintree precedence constraints [12]. However, they gave a counterexample showing that their proposed *HLF* does not work when considering general *dags*. Apart from these, we could not find any other mentionable results regarding this stochastic scheduling problem with *dags*. We try to uncover some details about why *HLF* does not work in this case and try to find implications for how an optimal strategy might look like.

So, the considered problem is

$$2|p_i \sim \exp(1); prec|\mathbb{E}(C_{\max}). \quad (28.2)$$

This can be seen as a generalization of (28.1) to stochastic processing times, or of $2|p_i \sim \exp(1);intree|\mathbb{E}(C_{\max})$ from (17.1) on page 38 to *dags* as precedence constraints. In the following two sections, we show why the optimal strategies from both Coffman-Graham and Chandy-Reynolds are not optimal for (28.2).

28.1 The Coffman-Graham Algorithm is not Optimal

Coffman and Graham [14] presented an optimal strategy that uses list scheduling. Their approach is similar to what we defined in Definition 14 on page 14, as they define priorities for the tasks as follows:

1. An arbitrary source is assigned priority 1.
2. Among all tasks whose direct successors have already been assigned priorities, compare the sequences of the priorities of their direct successors in increasing order. The task with the smallest such sequence (lexicographically) is assigned the next higher priority.
3. Repeat step 2 until all tasks have been assigned priorities.

This is similar to the level-oriented topological sorting from Definition 14 in the sense that priorities are assigned level by level here as well, so that, in the end, an *HLF* is obtained. However, the difference lies in choosing the tasks on a certain level. Whereas our definition chooses tasks from left to right, the approach of Coffman and Graham chooses tasks according to the priorities

of their successors. If ties arise, they are broken arbitrarily. The chosen priority is denoted by $\alpha(x)$ for a task x . We note that this assignment is not unique, but nevertheless, all possible assignments of these α -values will result in an optimal schedule [14].

The optimal algorithm then just schedules tasks with decreasing α -values, i.e., whenever a processor becomes idle, the available task with the highest α -value is chosen to be processed on this processor next.

Unfortunately, this algorithm fails in the stochastic setting with exponential processing times. Consider the *dag* on the left in Fig. 50. The indices of the tasks' names correspond to their respective α -values, i.e., it holds that $\alpha(x_i) = i$. An optimal schedule for the deterministic setting with unit processing times is given on the right in Fig. 50.

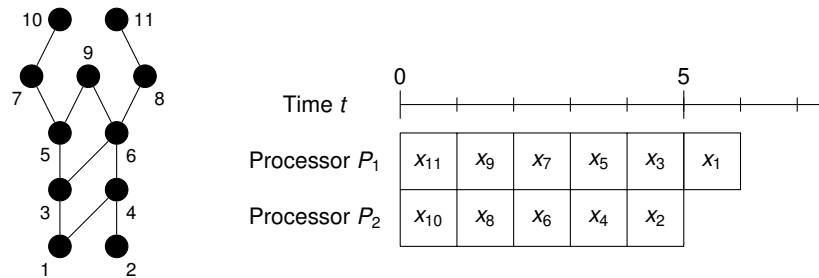


Fig. 50: A *dag* and its corresponding optimal schedule.

Now consider stochastic scheduling with exponential processing times. The highest seven layers of the configuration graph of the *dag* in Fig. 50 for Coffman's and Graham's list scheduling algorithm are shown in Fig. 51. This image (and the following images) were created using **PROSIT**, taken from Camino's master's thesis [10]¹. Each configuration has its corresponding expected makespan written below it. Active nodes are colored in orange. Other possible choices for other strategies are depicted by the gray configurations and edges, respectively. Suboptimal choices are marked with dashed edges. Here, the makespan is approximately 7.01855468. For a more detailed explanation of the tool and its features, we refer to the actual thesis [10].

¹The thesis as well as **PROSIT** can be found on www.carlos-camino.de.

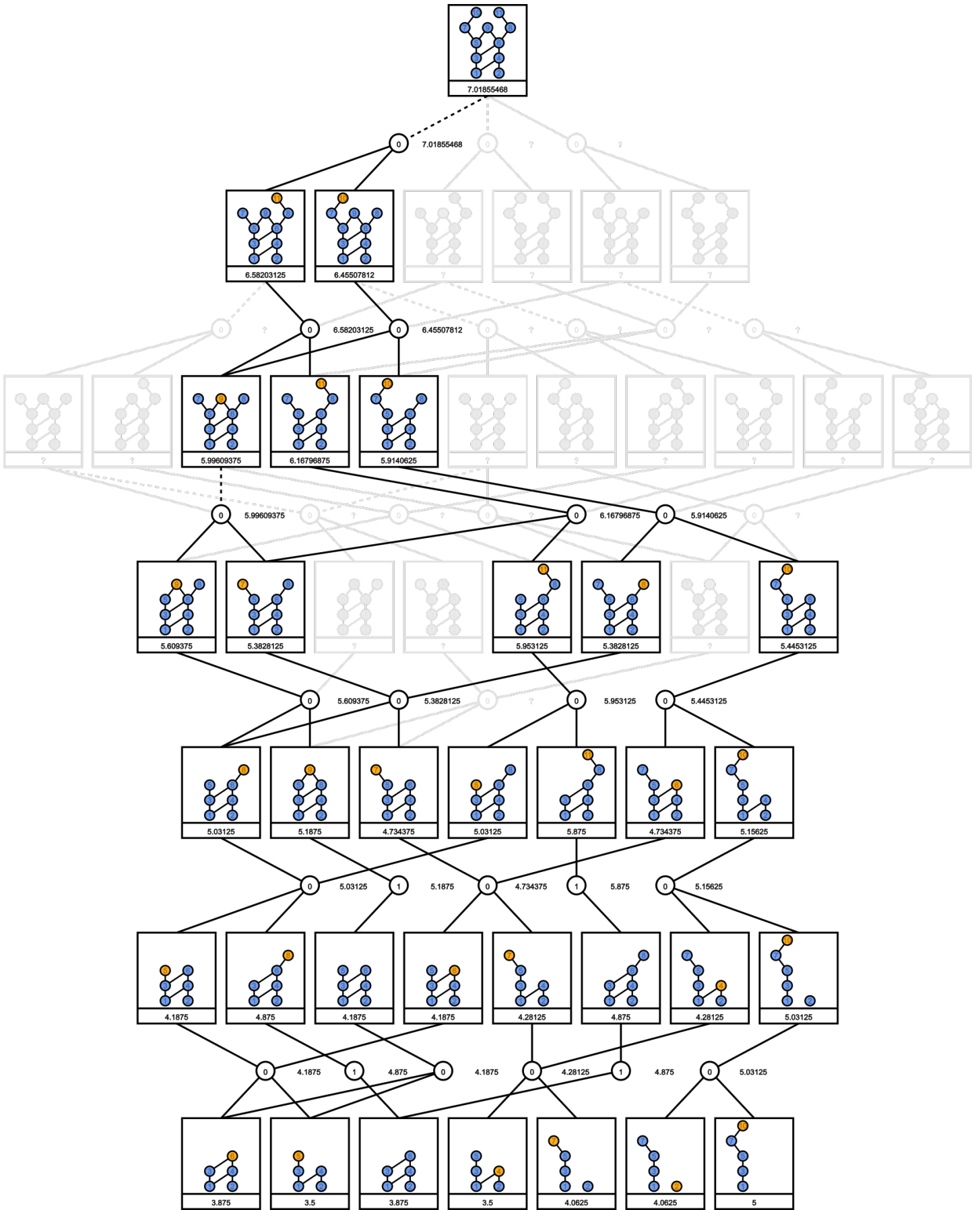


Fig. 51: Part of the configuration graph for the Coffman-Graham scheduling strategy.

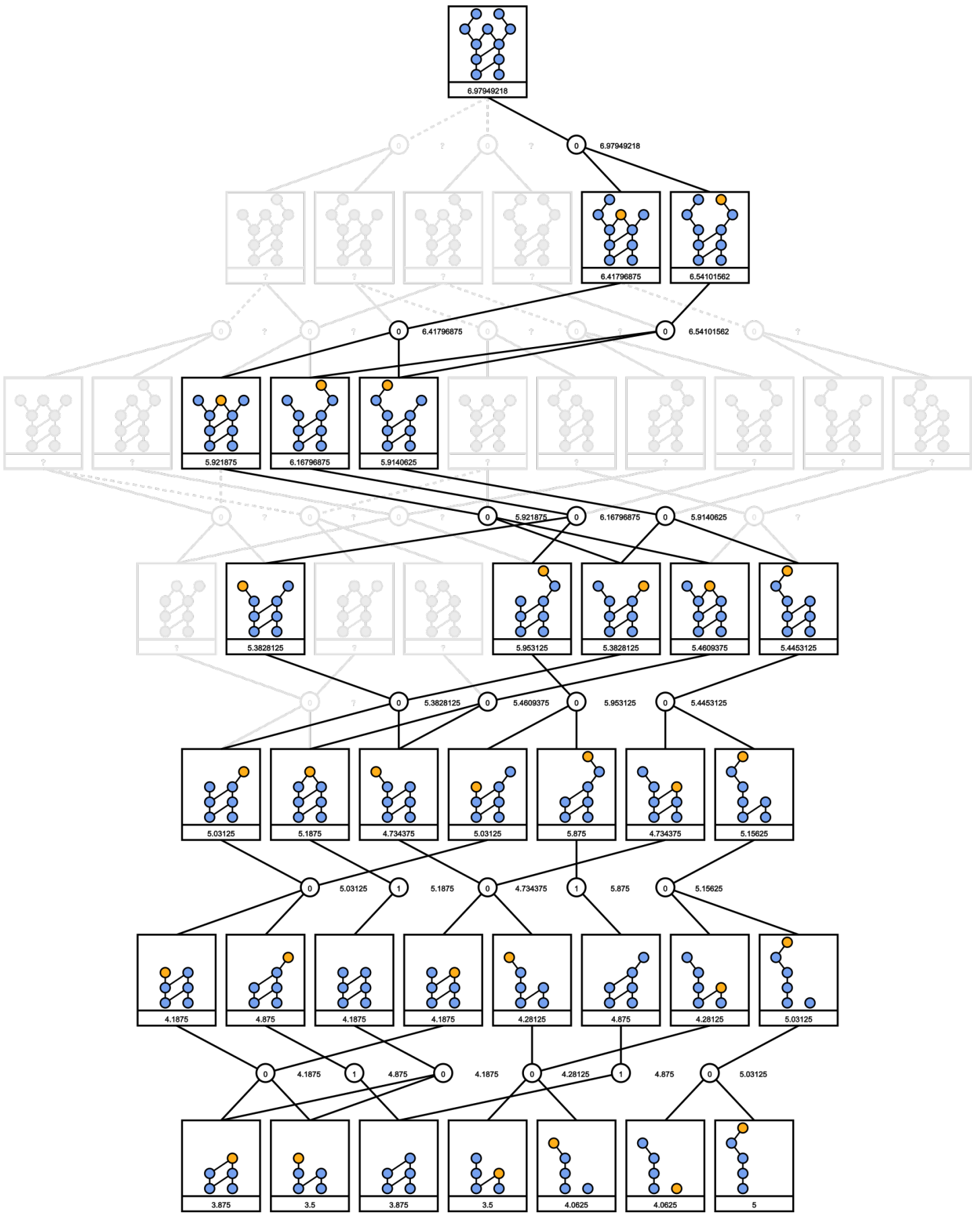


Fig. 52: Part of the configuration graph for the optimal scheduling strategy.

In contrast to that, an optimal strategy for the stochastic setting with exponential processing times chooses tasks x_{11} and x_9 to be processed at first, which is obviously not according to the “highest- α -value-first” strategy that is optimal in the deterministic setting. Fig. 52 shows the highest seven layers of the configuration graph of the optimal strategy in the stochastic setting. The makespan is approximately 6.97949218.

Note that **PROSIT** finds the optimal schedule by a brute-force method, a more elaborate strategy behind creating an optimal schedule is still unknown.

In addition to the different optimal strategies we can see the difference in the total processing time. In the deterministic case, the makespan amounts to 6, cp. Fig. 50. In the stochastic case, however, the makespan of the same graph is approximately 6.97949218, which is substantially higher. Of course, this is due to the fact that we may end up with a *degenerate* configuration, i.e., one that consists of a high chain and only a small number of other tasks on lower levels or one where many tasks are successors of only a few and have to wait until these few tasks have been fully processed to be available, see Fig. 53. Although those configurations have a very low probability to actually occur, the makespan is greatly influenced by them. In the deterministic case, there are no such degenerate configurations.

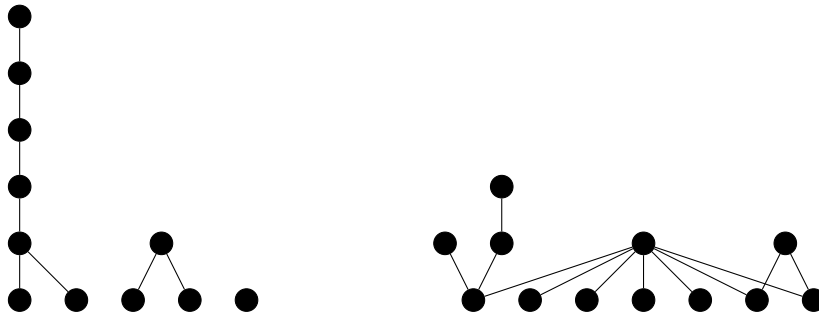


Fig. 53: Two degenerated configurations.

One approach on trying to minimize the expected makespan would be to minimize the possibilities of such degenerate configurations. In the $2|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max})$ problem, *HLF* does exactly that. Because we only deal with intrees in that case, there can never be a configuration where lots of tasks wait for only a few others to finish. This leaves only the high chains as bad cases, and their possibilities can be decreased by using *HLF*. Here, however, we can have several tasks being dependent on only a few others, see Fig. 53 on the right for an example. Another example would be task x_9 in the graph in Fig. 50, which is an ancestor of the six tasks x_1, \dots, x_6 although it is only on level 4. Now, to minimize the possibilities of these configurations, we have to find a trade-off between scheduling high-level tasks in order to avoid high chains and scheduling lower-level tasks with many descendants in order to avoid idle processors.

In the example from Fig. 50 and Fig. 52, we see that avoiding idle time is more important than avoiding high chains. In this case, the optimal strategy schedules not both highest sources, but only one and the other source on a lower level.

The proof from [14] that shows the optimality of the above strategy in the deterministic setting

$$2|p_i = 1; prec|C_{\max}$$

uses a structural decomposition of the *dag*, namely the decomposition in several blocks, which Coffman and Graham denote by the greek letter χ . These χ -blocks have certain properties:

- Each block χ_i has odd cardinality.
- Each task in block χ_i is a descendant of each task in block χ_{i+1} .
- All tasks in a χ -block are processed in one batch without any other tasks from other χ -blocks in between.

An example that is also used in [14] is pictured in Fig. 54. The χ -blocks of the *dag* on the left all have different colors. The corresponding optimal schedule shows the consecutive processing of the χ -blocks.

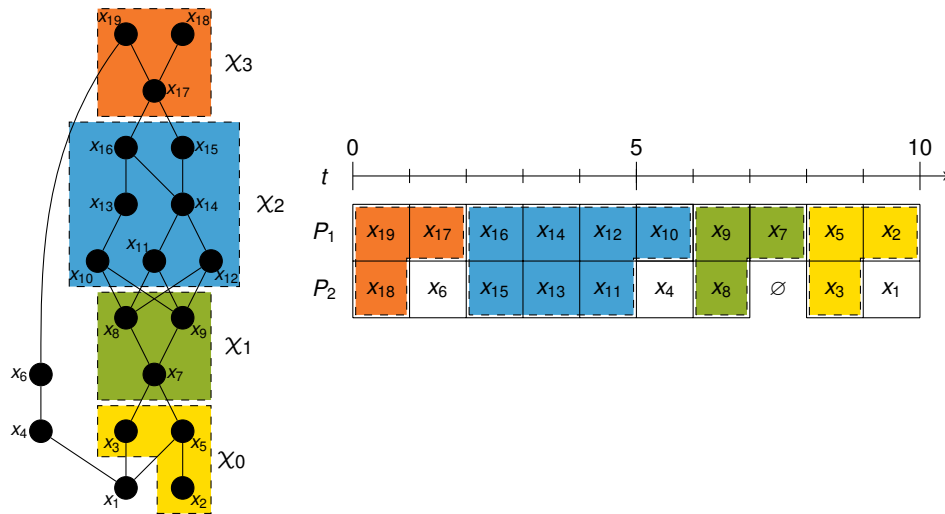


Fig. 54: A *dag*'s χ -blocks are processed in order in an optimal schedule.

Note that there may be tasks which do not belong to any χ -block, for example the three left tasks in Fig. 54. These are tasks that are kind of independent from those blocks and they **can** be processed in parallel with the tasks from some χ -block. Actually, this is a desired behavior. These “blockless” tasks, so called *fill-ins*, may be processed when there is at most one task from a χ -block available. Then, as it can never be the case that two tasks from different χ -blocks are processed simultaneously, such a fill-in can be chosen for processing.

The crucial part in the proof is that, with the first and second properties of the χ -blocks mentioned above, there is always an idle processor at the end of processing a block. But this property only holds because the processing times are deterministic and identical. This way, we know that every time unit at least one tasks finishes. In particular, while processing a block χ_i

of cardinality $2n_i + 1$, we know that both processors are busy processing $2n_i$ tasks for n_i time units before one processor becomes idle while the other processes the last task of χ_i . The idle processor can then choose to process some available fill-in (if it exists), and we know that both processors will be ready to process other tasks after exactly p more time units. But this is exactly the problem with stochastic scheduling. We can never be sure if these processors will be idle after one time unit, and, even more problematic, we can never be sure if they will both be ready simultaneously at some time point at all. So, in the worst case, a fill-in takes a very long time to be processed, and the tasks in the next χ -block have to be processed on only one processor as long as the fill-in is not finished.

28.2 The Chandy-Reynolds Algorithm is not Optimal

The approach from Chandy and Reynolds [12], namely *HLF*, does not work well here. It is proven to be optimal for exponential processing times and intree precedence constraints, but fails when considering *dags*. The example seen in Fig. 51 works also as a counterexample to *HLF*. Fig. 55 shows the highest seven layers of a configuration graph for an *HLF* schedule. Looking at the values for the makespan, we can see that this strategy is suboptimal. In addition, we see that *HLF* choices sometimes correspond to dashed edges – for example in the first step – which indicate suboptimal choices. The makespan is even worse than with the Coffman-Graham scheduling strategy from Fig. 51, and is approximately 7.046875.

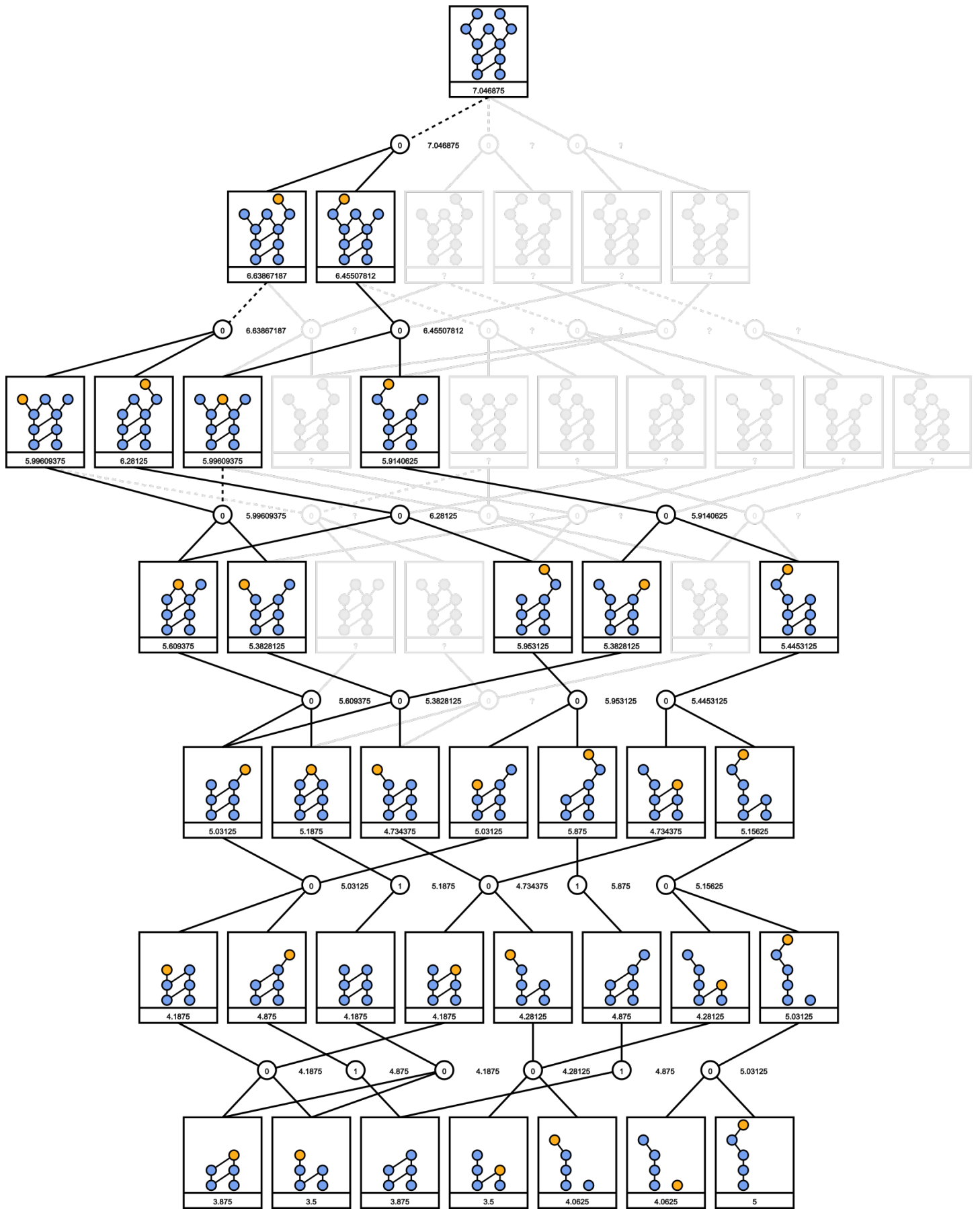


Fig. 55: Part of the configuration graph for an *HLF*.

Moreover, not only is *HLF* not optimal, but different *HLFs* yield different makespans. In the setting with onlyintree precedence constraints, all *HLFs* were proven to be equivalent in terms of the resulting makespan. With *dags*, this is not the case, see Fig. 56. The *HLF* on the left chooses the middle and the right source at the beginning, which results in an expected makespan of 4.640625, whereas the *HLF* on the right prioritizes the left and middle source, resulting in an expected makespan of 4.6875.

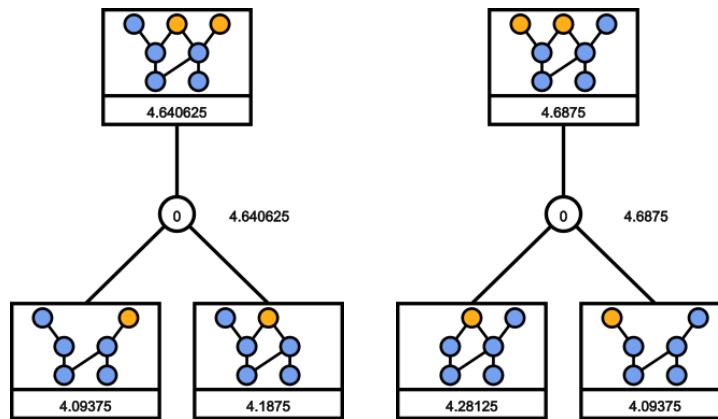


Fig. 56: Different *HLFs* result in different makespans.

This tells us that, even if for some *dags* the optimal strategy is *HLF*, then it may be the case that **not every** *HLF* is optimal. Then, we would have to deal with different types of optimalities, cp. Chapter 6 on page 13.

However, it was shown – empirically, not mathematically – that *HLF* is near-optimal: in [5], Adam, Chandy, and Dickson did several benchmark simulations on smaller examples (up to 266 tasks) and for almost all of them, *HLF* out-performs the other strategies they tested and is only a few percent away from the optimal solution. Still, this result is not theoretically proven, although it is indicated that *HLF* may actually be asymptotically optimal (a result which is actually proven for intrees and three processors, see Chapter 35 on page 174).

28.3 Static, Semi-Static, and Dynamic Scheduling Strategies

Furthermore, the optimal strategy might not even be *static*, but rather a *dynamic* scheduling strategy. This means that the priorities according to which the next task is chosen can change during the scheduling process. So far, the strategies that we have encountered were all static, because the priorities were properties of the tasks which do not change even during the scheduling process. Take the levels for example. A level of a task is only defined by the length of a shortest path to a root and that path is not altered by iteratively removing sources from the graph. To change the level of a task, its shortest path to a root has to be changed, but as it contains only tasks which are descendants of that task, they will never be available before the task itself. See Fig. 57 for an example. The left graph shows the tasks' levels, the right graph shows the

levels of the remaining tasks after some tasks on the higher levels are removed (corresponding to being chosen and completely processed during a schedule). These values for the levels are not changed.

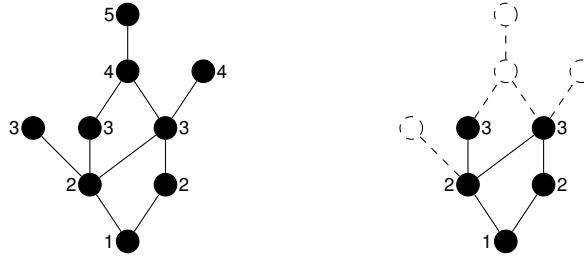


Fig. 57: The levels are static values.

The same can be said for the α -values. These are defined recursively, and bottom-up. Removing sources on higher levels of the graph does not change the α -values of the low-level tasks, see Fig. 58.

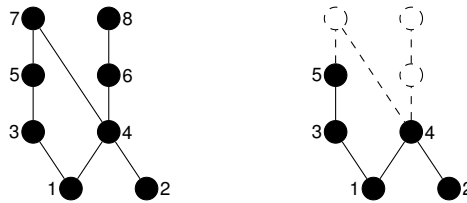


Fig. 58: The α -values are static values.

A *semi-static* property of a task is one which can change during the scheduling process (e.g., when other tasks have already been processed), but only depends on the current structure of the graph. This name comes from [41]. For example, consider the following:

1. The number of tasks that are on the same level. Whenever a task finishes, all other task on that level have to update this value, see Fig. 59. This is just a profile for a *dag*.

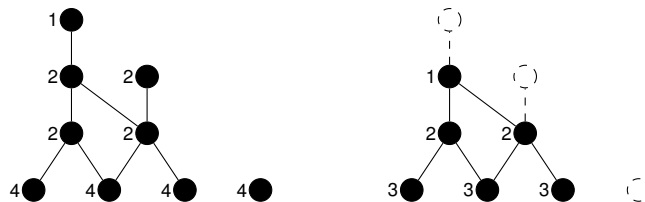


Fig. 59: The number of tasks on the same level is semi-static.

2. The number of ancestors of a task. Whenever a task finished, all its descendants have to update this value, see Fig. 60.

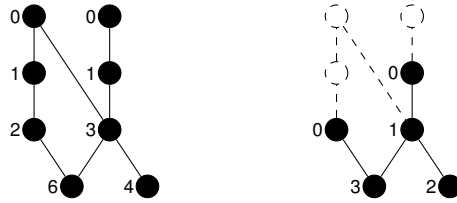


Fig. 60: The number of ancestors of a task is semi-static.

3. The tuple of the number of ancestors and the number of descendants of a task. In the same way that the number of ancestors may change during the process, this value changes, too. The number of descendants, however, remains the same, see Fig. 61.

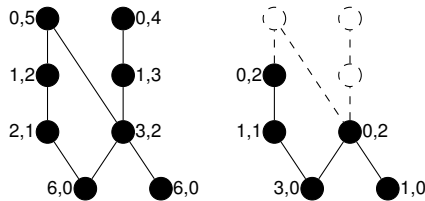


Fig. 61: The tuple of the number of ancestors and descendants of a task is semi-static.

Note that all three of those properties can also be considered in a static fashion if we calculate them once in the original graph, and then never care to update them later on. In the following section, we define and examine some scheduling strategies, static as well as dynamic.

Case Studies on some Static and Dynamic Scheduling Strategies

Considering the examples from Fig. 50, we want to find a strategy that schedules the lower source, i.e., task x_9 . At first, we examine the three example strategies that we proposed in Section 28.3.

1. Priority = number of tasks on the same level: This strategy may be optimal for the graph in Fig. 50, but it fails on the graph in Fig. 62. The suboptimal choice made by this strategy is depicted with dashed, gray lines. The optimal choice for the first step is to schedule the lower source on level 2 and only one of the top-level sources. Because it does not matter which of the top-level sources an optimal strategy chooses, there is more than one possible optimal choice. But as these are all isomorphic (because unless we give them names, we cannot tell these tasks apart), the configuration is only drawn once. The small numbers in the black circles indicate the number of isomorphic configurations. Obviously, the problem with this graph is that the one low task has too many descendants, so an optimal strategy has to choose this task in order to decrease the probability of the degenerated configuration

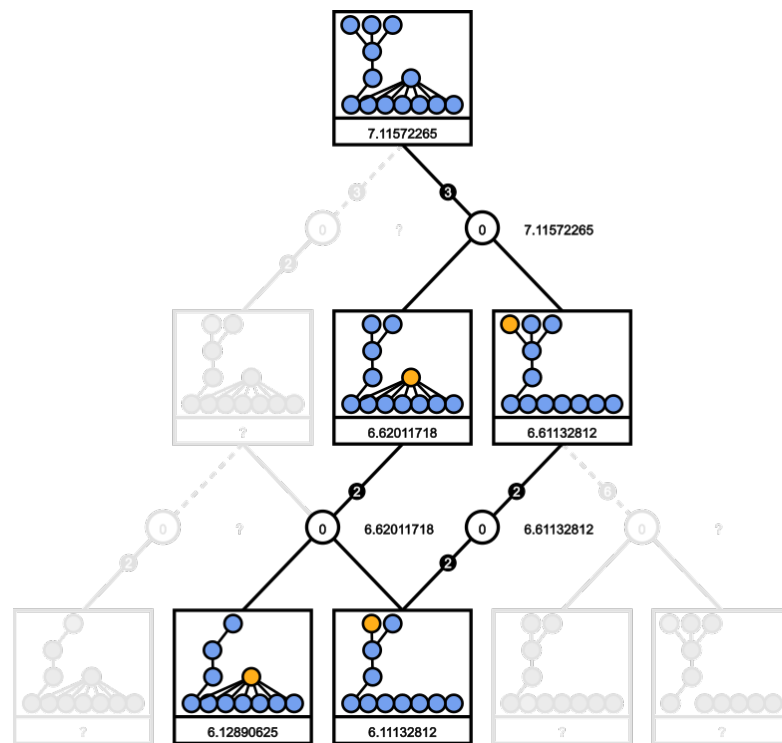


Fig. 62: Part of a configuration graph for the strategy using the number of tasks on the same level.

where all seven tasks on the first level have to wait for this task to finish while one processor is idle.

2. Priority = number of ancestors: This strategy is optimal for the graph in Fig. 50 and Fig. 62, but fails on the graph in Fig. 63. This graph is very similar to the one before, it just has one more task in the chain on the left. This means that in this graph, decreasing the probability of a high chain is more important in the first step than decreasing the probability of having the seven level-1 tasks wait for their common predecessor. For the graph before, it is just the other way around.
3. Priority = tuple of the number of ancestors and the number of descendants of a task: The first problem arises with comparing two tuples. It is not clear how these are ordered. But even if we restrict ourselves to special cases with no ancestors (which correspond to sources), then we have seen that only comparing the number of descendants does not give us an optimal strategy.

One commonality among all optimal strategies so far is that they always choose at least one task on the highest level. So, a strategy that chooses at least one highest source may be optimal. But as it turns out, this is not the case. Consider the configuration graph for an optimal schedule

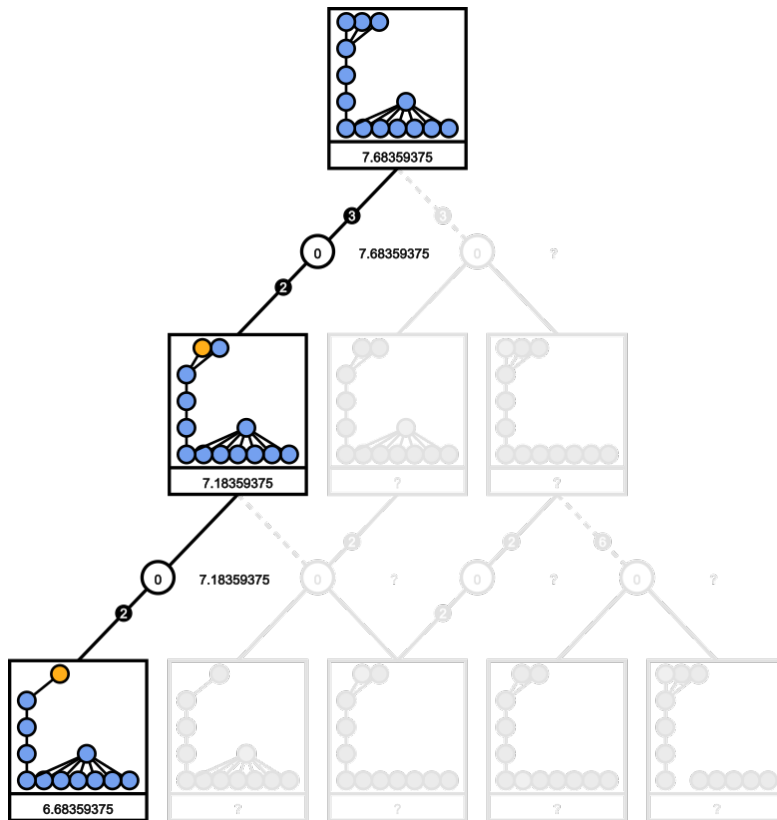


Fig. 63: Part of a configuration graph for the strategy using the number of ancestors.

in Fig. 64. In the first step, the optimal strategy is to choose both sources on level 2, and choose one of the sources on level 3 only after one task is finished.

28.3.1 Level and Number of Descendants of a Task

As seen in Fig. 62 and Fig. 63, two crucial properties of the tasks are the level and the number of descendants. Considered alone, neither of them can be taken as a priority. But considered in parallel, there are some patterns visible when trying different smaller examples. This section shows such a pattern and discusses this priority.

Define as a priority the tuple of the level of a task and the number of its descendants. These priorities are written as $x|y$ where x denotes the level of a task, and y denotes the number of its descendants, see Fig. 65 for some examples. The white node in the left graph is on level 4 and has five descendants (depicted by the blue nodes), i.e., its values are $4|5$. The white node in the right graph is on level 3 and has three descendants, i.e., its values are $3|3$.

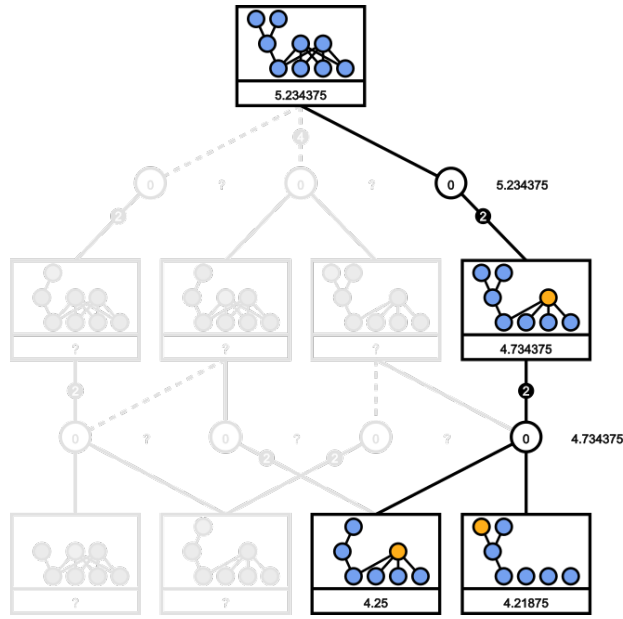


Fig. 64: Part of a configuration graph for a strategy always choosing one of the highest sources.

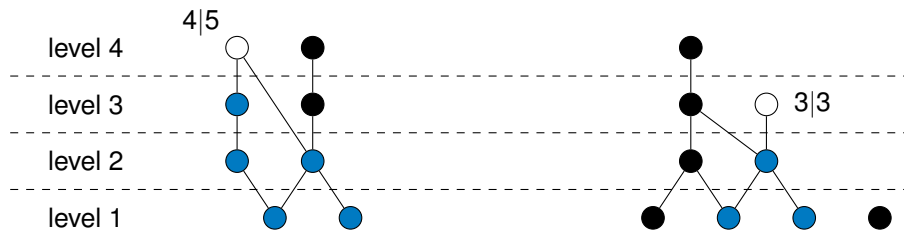


Fig. 65: The tuple of the level and the number of descendants of a task.

As we argued before, the important part when dealing with tuples is the metric, i.e., how we are supposed to compare tuples. As a symbol for comparing two priorities we have

$$x|y < x'|y' \iff \text{a task corresponding to the first tuple } x|y \text{ has strictly smaller priority than a task corresponding to the second tuple } x'|y'. \quad (28.3)$$

In the following, we compare tasks with different values to see if a pattern emerges. The observations are all done using **PROSIT** to compute an optimal strategy. For example, we want to show that $2|3 < 3|2$. This means that we construct a graph where we have one source on level 2 with three descendants, and two sources on level 3 with two descendants each, or vice versa, see Fig. 66. As the optimal strategy chooses both level-3 sources, we know that these have a higher priority than the level-2 source. Basically, we provide the strategy with two choices: either both sources with identical values are chosen, or only one of those and a source with

other values are chosen. In the first case, the priority of the two level-3 sources with identical values is higher than the other on level 2. In the second case, it is the other way around.

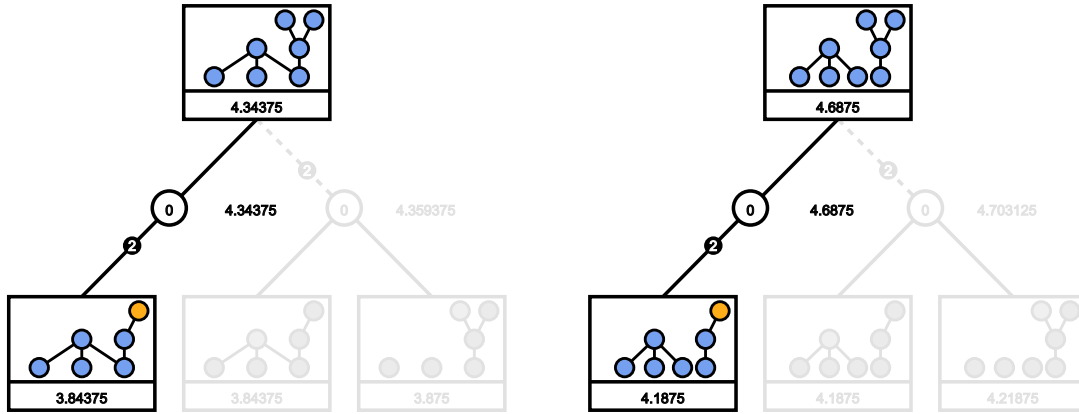


Fig. 66: The optimal strategy for these graphs implies that $2|3 < 3|2$.

Some examples of relationships between other smaller priority values can be seen in Fig. 67.

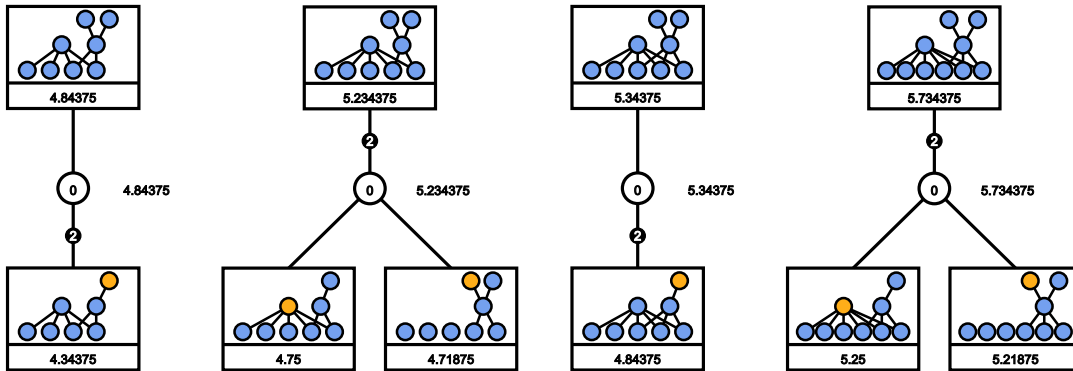


Fig. 67: Four graphs that show (from left to right): $2|4 < 3|3$, $3|3 < 2|5$, $2|5 < 3|4$, and $3|4 < 2|6$.

Altogether, we obtain the some relationships for certain priority values of tasks below level 5, which can be seen in Fig. 68. The priority increases from left to right, meaning that tasks with values more to the right have a higher priority than tasks with values to the left. Furthermore, for these values, the priority relation “ $<$ ” is transitive. So, for example, it holds that $2|4 < 4|3$ because we have $2|4 < 3|3 < 2|5 < 3|4 < 4|3$. Values corresponding to the same level are depicted on the same row, with a row for each level. A zig-zag pattern is visible for these small values.

As we can see in Fig. 66 and Fig. 67, the actual structure of the graph does not change the relationship of the two respective values. This means that no matter how a level-3 task with four descendants occurs in the graph, it will always have a smaller priority than a level-2 task with

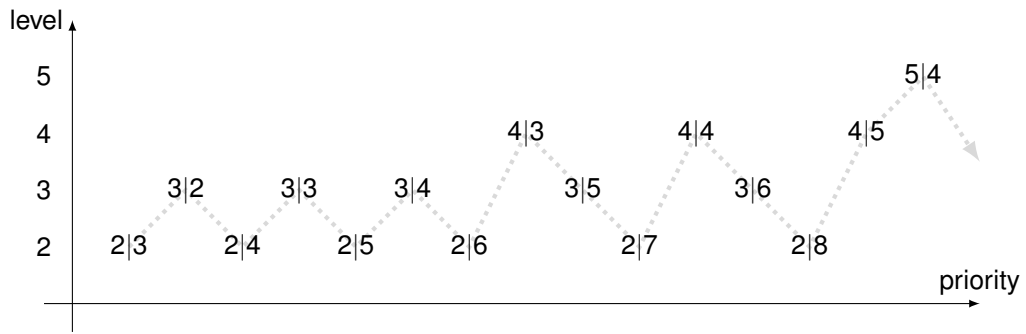


Fig. 68: The order of some $x|y$ values.

six descendants. Fig. 69 gives four different possibilities to have a task with values $3|4$ (which is always the topmost one). These four possibilities are all, up to isomorphisms.

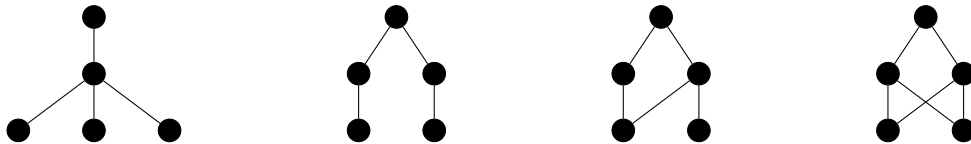


Fig. 69: The four different possibilities for a task with values $3|4$.

However, for larger graphs, the structure of the graph itself does become important. The number of descendants alone does not describe the structure well enough, because for larger graphs there are even more possibilities for the structure to look like with same values for the respective tasks. The smallest counterexample which we could find is between tasks with values $3|8$ and tasks with values $5|5$. Depending on how the eight descendants of the level-3 task are arranged, the priority is higher than that for the level-5 tasks in one case, but lower in the other, see Fig. 70. For the graph on the left, the optimal strategy chooses the level-5 sources, whereas choosing the level-3 source over one of the level-5 sources is optimal for the graph on the right. This proves that even these two values in combination are not enough to characterize an optimal strategy. Moreover, the crucial part is not even that the reachable roots are distributed among paths via several level-2 tasks, as we can see in Fig. 71. The two graphs differ only in the removal of one edge in the left graph. So it appears that not only the number of reachable roots, but the number of paths to these roots are important as well.

From what **PROSIT** can manage to calculate, we suspect that the relation $x|y < x|(y + 1)$ still holds, regardless of the number of paths from the level- x sources to the roots. We compared tasks on level 3 with up to 17 descendants and up to 64 different paths to roots. The absolute difference between the optimal strategy and the suboptimal strategy is less than 10^{-5} , but still we have that

$$3|16 < 3|17,$$

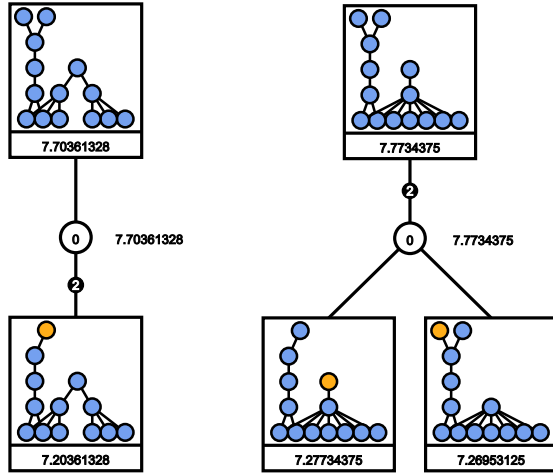


Fig. 70: The tuples 5|5 and 3|8 are incomparable.

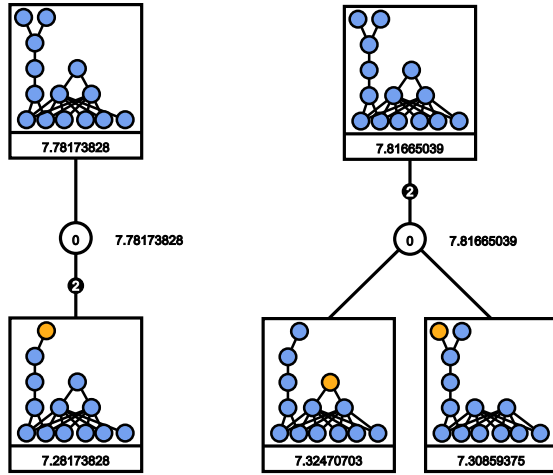


Fig. 71: Only one edge makes the difference for the optimal strategy.

where the task with only 16 descendants has the highest possible number of different paths to its roots (which means connecting it to the complete bipartite subgraph $K_{8,8}$), and the task with 17 descendants has the smallest number of different paths to its roots (which means having the maximal number of roots), see Fig. 72. The configuration graph which was used to obtain the above result is omitted here because the graph is too messy to see anything.

28.3.2 Number of Induced Paths of a Task

We call a path from a task to any of the roots reachable from it an *induced path*. The number of induced paths cannot be used as a priority either. For a smaller number of tasks, we can

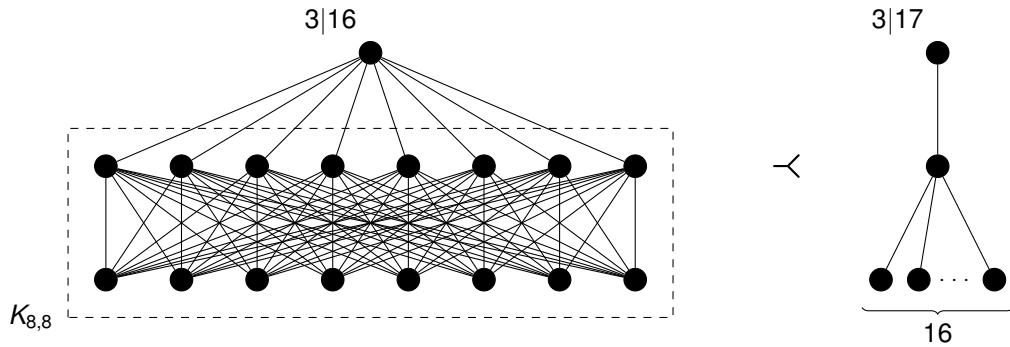


Fig. 72: Even for a big difference in the number of edges, $3|16 < 3|17$.

see a pattern emerging in the same way that we have seen in Fig. 68 for the priority with the level and the number of descendants. However, if the number of tasks increases, then there are counterexamples for which the number of induced paths or even the tuple of level and number of induced paths does not work, see Fig. 73 for example.

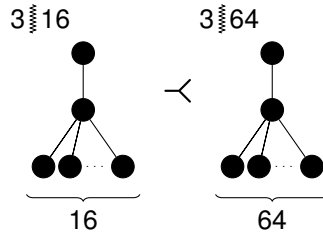


Fig. 73: The number of induced paths of a task as a priority.

We introduce the notation $x||y$ for a task on level x with y induced paths. It is obvious that a level-3 task with one (direct) successor and 16 descendants on level 1 has a lower priority than a level-3 task with one successor and 64 descendants on level 1, giving $3||16 < 3||64$ if we adopt the priority notation $<$ from (28.3), see Fig. 73. After all, we only added more tasks on the lowest level as descendants. But then, with Fig. 72 we get that

$$3||16 < 3||64 < 3|16.$$

Thus, level-3 tasks with 16 and 64 induced paths are not comparable.

Even more peculiar, if we consider Fig. 72, then the probability that some degenerate configuration is reached is the same in both graphs. For example, take the configuration where there are only the level-1 tasks and a chain of height 5 left (this configuration is denoted by $\ell_{5,4}$ in [10]). Fig. 74 shows the probability of reaching that configuration when starting with the left graph from Fig. 71 while applying *HLF*, which is the optimal strategy here. The values in the boxes indicate the probability of a configuration. The probability of reaching the configuration

$\ell_{5,4}$ (bottom right) is $\frac{3}{16}$. Fig. 75 shows the same probability for the right graph from Fig. 71 while applying the non-optimal *HLF*, and Fig. 76 shows that while applying the optimal strategy. In all three cases, the probability of reaching the degenerate configuration $\ell_{5,4}$ is the same, i.e., it is $\frac{3}{16}$. This means, that even the occurrence probabilities of some degenerate configurations cannot offer some valuable information about choosing tasks for a strategy.

Of course, all these different configuration graphs are not so different. The underlying graphs differ only by one edge after all. Nevertheless, this is sufficient to show that we cannot use the occurrence probabilities alone to define a strategy, or to get some more information about the optimal strategy.

Altogether, we could see that even tuples of values that seemed promising do not hold much value when considering larger graphs. We could have considered triples or larger combinations, but had the feeling that there is always another characterization or another value for a task that needs to be included in the combination. These values may then only be hand-crafted to fit some previous counterexamples and/or too complex in the sense that the calculation of these priority combinations are no longer efficient to be used to find an optimal strategy. This may indicate that not even a semi-static strategy can be optimal, but rather a *dynamic* strategy that takes into account some of the previously made choices.

28.4 Choosing Pairs of Sources

An optimal strategy minimizes the expected makespan. Moreover, it somehow minimizes the probability of degenerate configurations. The question is always: At what cost? What if there are several degenerate configurations that can be reached? One problem that may arise is that the chosen tasks take very long to be processed and no other tasks can be processed while the processors are busy. So, one idea is to consider the subconfigurations of a pair of sources and their subgraph of descendants, see Fig. 77 and Fig. 78 for example. Doing this for every pair of sources and comparing the expected makespan of the respective, *induced* subconfigurations may leave a value that may be used as a priority (or part of it). As it turns out, this is not the case. See Fig. 78 for a depiction of this behavior: Neither the subconfiguration with the highest makespan, nor the one with the lowest corresponds to the two sources that will be chosen by the optimal strategy.

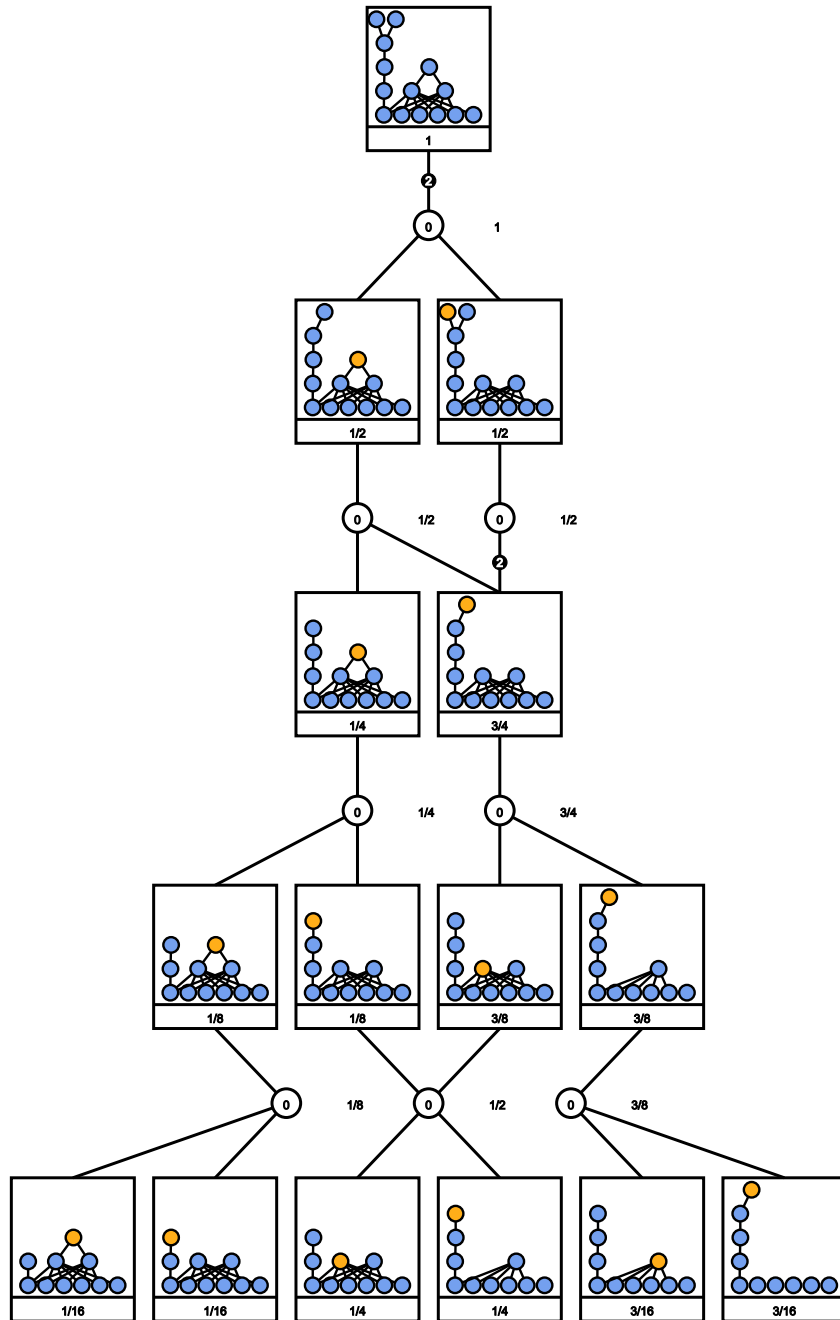


Fig. 74: Part of the configuration graph of an optimal *HLF* schedule for the left graph in Fig. 71.

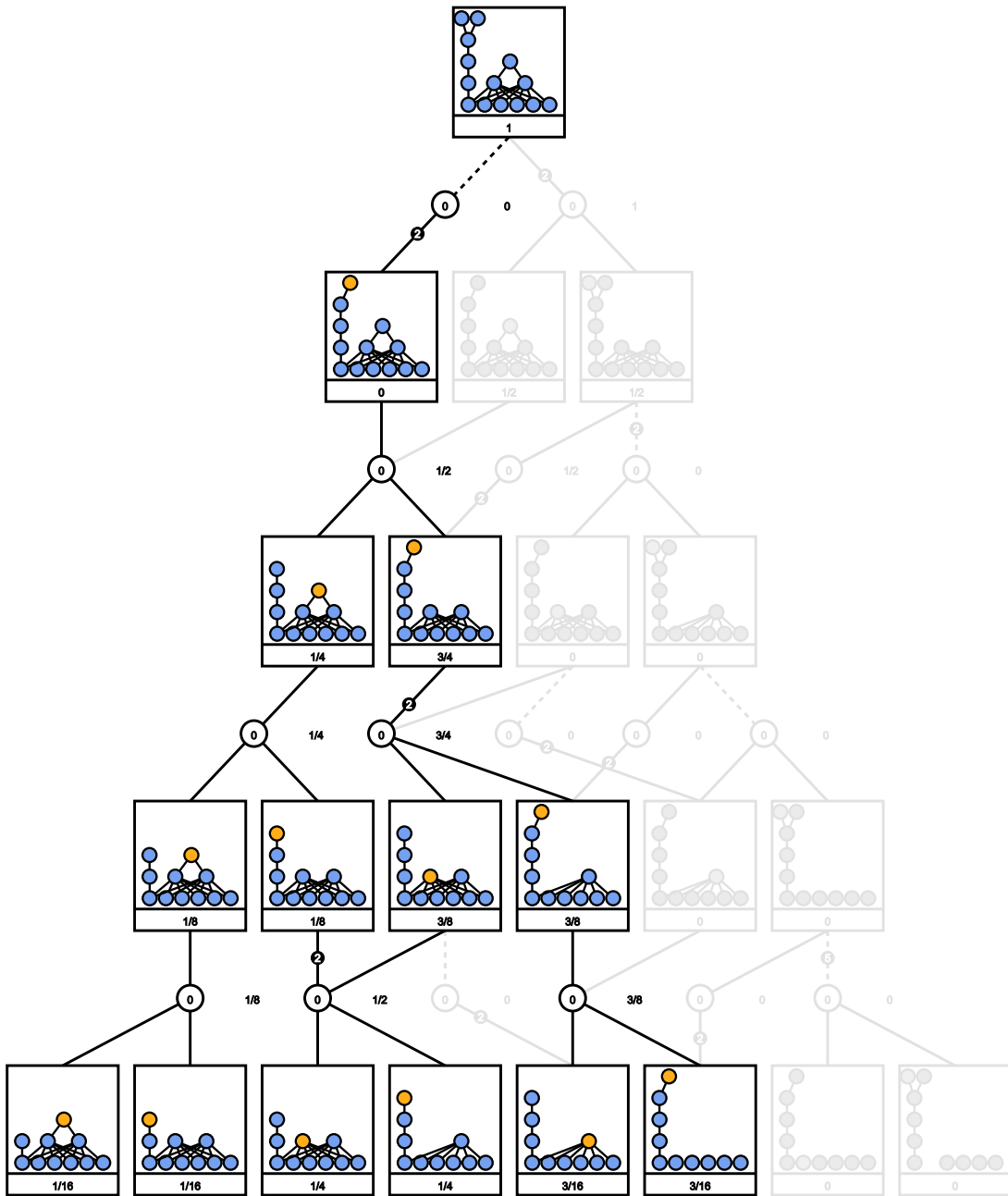


Fig. 75: Part of the configuration graph of an non-optimal *HLF* schedule for the right graph in Fig. 71.

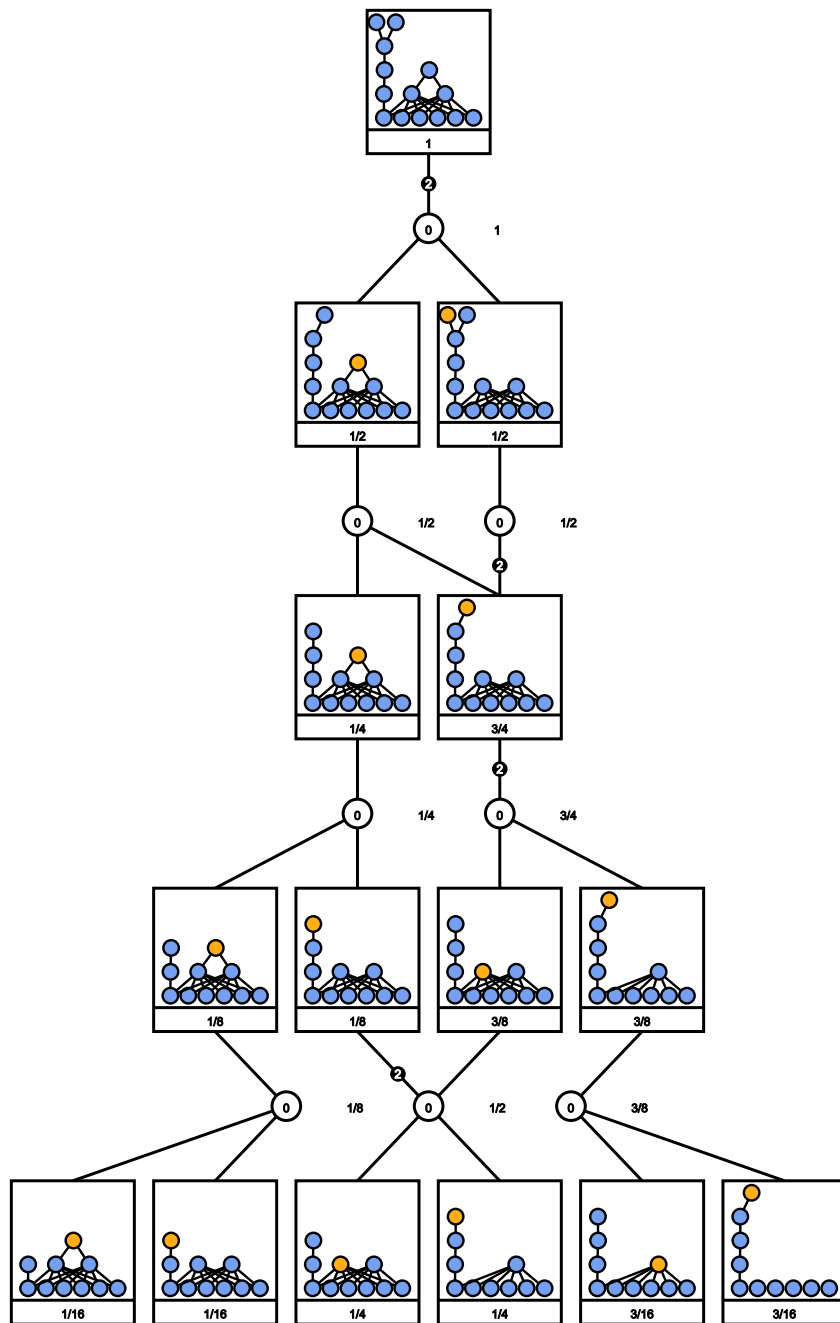


Fig. 76: Part of the configuration graph of an optimal schedule for the right graph in Fig. 71.

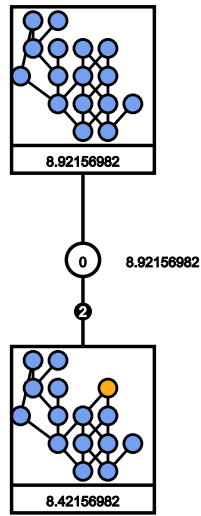


Fig. 77: A dag with 16 tasks, which is optimally scheduled by a non-*HLF*.

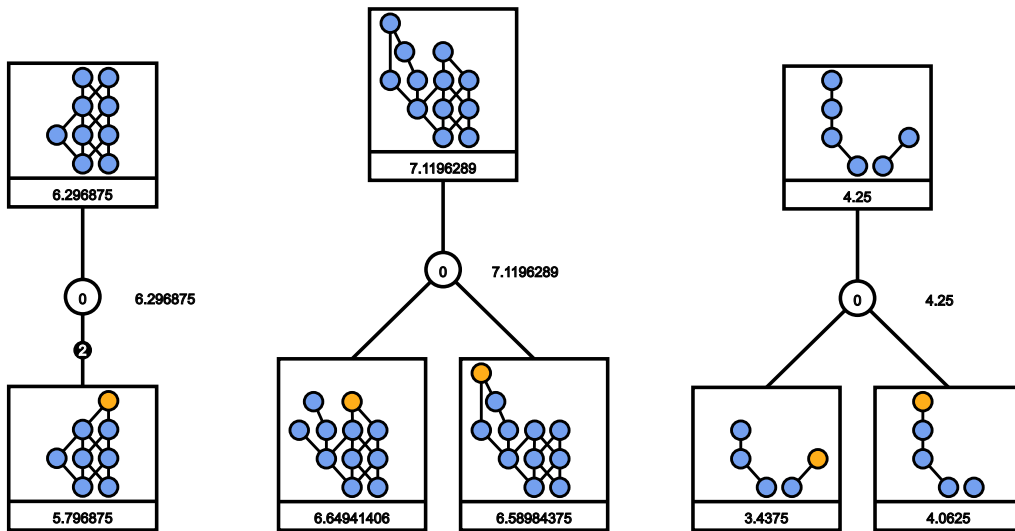


Fig. 78: Induced subconfigurations of pairs of sources do not hold any value as a priority.

29 Calculating the Expected Makespan

In the same manner as in Chapter 16 on page 36, we want to derive a formula (or several equivalent ones) which expresses the optimal expected makespan. Many of the expressions forintree precedence constraints heavily used the fact that *HLF* is the proven optimal strategy. This is an issue in the case of *dag* precedence constraints as we do not know the optimal strategy. However, for some simpler examples of a specific structure, we can easily calculate the optimal expected makespan.

Consider a level-2 task with i successors, denoted by M_i in the following, and called an M -graph. Furthermore consider i singular tasks, denoted by \underline{i} , see Fig. 79 for an example of both.

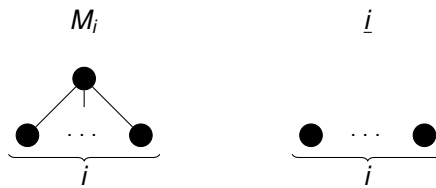


Fig. 79: An M -graph and i singular tasks.

The formulas for the makespans of these two types of *dags* are very simple. For \underline{i} , we have

$$C_c(\underline{i}) = 1 + \frac{i-1}{2}, \quad (29.1)$$

whereas for M_i , we have

$$C_c(M_i) = 1 + C_c(\underline{i}) = 2 + \frac{i-1}{2}. \quad (29.2)$$

It gets more interesting if we consider a sequence of M -graphs, see Fig. 80 for example. In particular, sequences of M -graphs form an *outforest*.

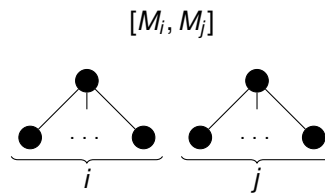


Fig. 80: A *dag* consisting of two M -graphs.

The makespan of this *dag* is much more interesting. In detail, it holds that

$$C_c([M_i, M_j]) = \frac{1}{2} + \frac{1}{2}C_c(\underline{i}, M_j] + \frac{1}{2}C_c([M_i, \underline{j}])$$

with

$$\begin{aligned}
C_c([i, M_j]) &= \frac{1}{2} + \frac{1}{2} \underbrace{C_c(i+j)}_{\substack{(29.1) i+j-1 \\ = \\ \frac{i+j-1}{2} + 1}} + \frac{1}{2} C_c([i-1, M_j]) \\
&= \frac{1}{2} + \frac{1}{2} \left(\frac{i+j-1}{2} + 1 \right) + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} \left(\frac{i+j-2}{2} + 1 \right) + \frac{1}{2} C_c([i-2, M_j]) \right) \\
&= \frac{1}{2} + \frac{1}{4} + \frac{1}{2} \left(\frac{i+j-1}{2} + 1 \right) + \frac{1}{4} \left(\frac{i+j-2}{2} + 1 \right) + \frac{1}{4} C_c([i-2, M_j]) \\
&\dots \\
&= \sum_{k=1}^i \left(\frac{1}{2} \right)^k \left(2 + \frac{i+j-k}{2} \right) + \left(\frac{1}{2} \right)^i C_c(\underbrace{[0, M_j]}_{=M_j}) \\
&\stackrel{(29.2)}{=} \sum_{k=1}^i \left(\frac{1}{2} \right)^k \left(2 + \frac{i+j-k}{2} \right) + \left(\frac{1}{2} \right)^i \left(2 + \frac{j-1}{2} \right). \tag{29.3}
\end{aligned}$$

The next step is to add another graph M -graph to the sequence, and calculate the makespan again. Now, this is the first time when we have to consider which choices the optimal strategy makes in order to recurse to the correct configurations. But, for obvious reasons, we have that the top task of M_i is always prioritized over the top task of M_j if $i \geq j$. Additionally, a level-2 task is always prioritized over a singular level-1 task. With these rules in mind, we can consider more than just two of those graphs in a sequence, namely $[M_i, M_j, M_k]$ with $i \geq j \geq k$, w.l.o.g.:

$$\begin{aligned}
C_c([M_i, M_j, M_k]) &= \frac{1}{2} + \frac{1}{2} C_c([i, M_j, M_k]) + \frac{1}{2} C_c([M_i, \underline{j}, M_k]) \\
&= \frac{1}{2} + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([i, \underline{j}, M_k]) + \frac{1}{2} C_c([i, M_j, \underline{k}]) \right) \\
&\quad + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([i, \underline{j}, M_k]) + \frac{1}{2} C_c([M_i, \underline{j}, \underline{k}]) \right) \\
&= 1 + \frac{1}{2} C_c([i+j, M_k]) + \frac{1}{4} C_c([i+k, M_j]) + \frac{1}{4} C_c([j+k, M_i]) \tag{29.4} \\
&\stackrel{(29.3)}{=} 1 + \frac{1}{2} \left(\sum_{\ell=1}^{i+j} \left(\frac{1}{2} \right)^\ell \left(2 + \frac{i+j+k-\ell}{2} \right) + \left(\frac{1}{2} \right)^{i+j} \left(2 + \frac{k-1}{2} \right) \right) \\
&\quad + \frac{1}{4} \left(\sum_{\ell=1}^{i+k} \left(\frac{1}{2} \right)^\ell \left(2 + \frac{i+k+j-\ell}{2} \right) + \left(\frac{1}{2} \right)^{i+k} \left(2 + \frac{j-1}{2} \right) \right) \\
&\quad + \frac{1}{4} \left(\sum_{\ell=1}^{j+k} \left(\frac{1}{2} \right)^\ell \left(2 + \frac{j+k+i-\ell}{2} \right) + \left(\frac{1}{2} \right)^{j+k} \left(2 + \frac{i-1}{2} \right) \right).
\end{aligned}$$

As we can observe, the makespan of a sequence of three M -graphs can be reduced to a weighted sum of the makespans of *dags* of the form seen in Fig. 79, i.e., an M -graph and some more singular tasks. In Fig. 81, we can see the initial levels of the corresponding configuration graph. We can infer from (29.4) that this reduction works for an M -graph sequence of any

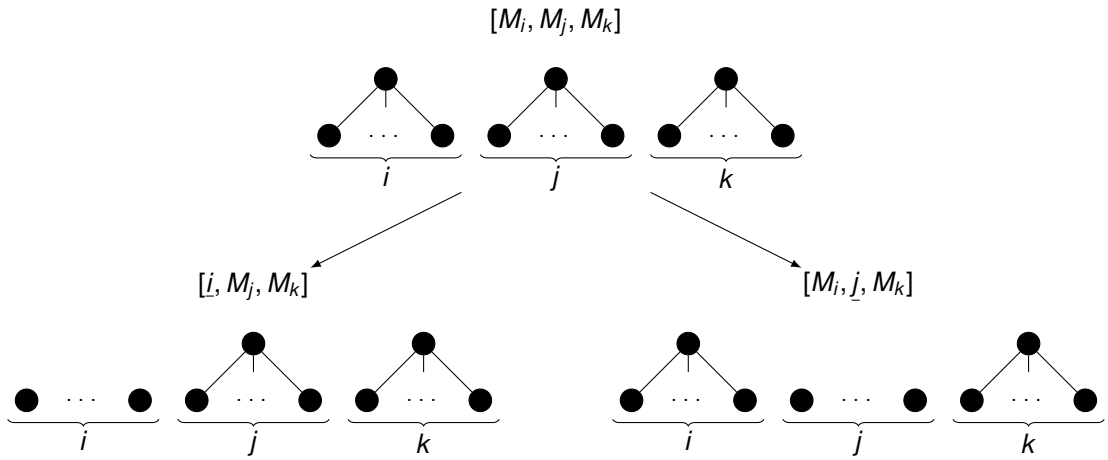


Fig. 81: The recursive reduction of a sequence of three M -graphs.

length. For each recursion level, the number of M -graphs in the recursive calls decrease by 1 until only one remains and we can use the equation derived in (29.3).

Next, we can examine some generalizations of M -graphs with more than one source. For example, we have

$$C_c(K_{2,i}) = C_c \left(\begin{array}{c} \text{graph} \\ i \end{array} \right) = \frac{1}{2} + C_c(M_i) = 2 + \frac{i}{2},$$

$$C_c(K_{j,i}) = C_c \left(\begin{array}{c} \text{graph} \\ i \end{array} \right) = \frac{j-1}{2} + C_c(M_i) = 1 + \frac{m+n}{2},$$

again using the notation $K_{j,i}$ for a complete bipartite graph with independent sets of sizes j and i . Even if we do not have a complete bipartite graph, but a graph where the k -th level-2 task has m_k successors, $k \in \{1, \dots, j\}$, and in total there are i level-1 tasks, we can easily have an expression for the makespan. This kind of incomplete bipartite graph is shown in Fig. 82, with j nodes on the top and i nodes on the bottom. The k -th top node is connected to m_k bottom nodes (not necessarily nodes which are drawn next to each other, as could be interpreted from the figure), of which there are i in total.

The optimal strategy prioritizes the source by the number of successors, which means that after $i-2$ tasks are finished, only the two sources with the fewest successors remain. Then, after the next task finishes, an M -graph remains, either the task with the fewest successors, or the task with the second-fewest successors. Let G be the graph from Fig. 82, and let m_{\min} and $m_{2-\min}$ be the minimal and second-minimal number of successors, respectively, i.e., the two smallest of all

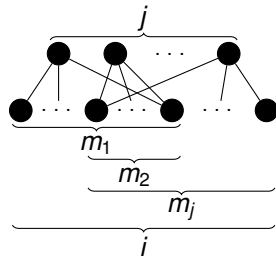


Fig. 82: An incomplete bipartite graph with $j + i$ nodes.

the m_k values. Then, it holds that

$$\begin{aligned}
 C_c(G) &= \frac{j-1}{2} + \frac{1}{2} C_c(M_{m_{\min}}) + \frac{1}{2} C_c(M_{m_{2-\min}}) \\
 &= \frac{j-1}{2} + \frac{1}{2} \left(2 + \frac{m_{\min} - 1}{2} \right) + \frac{1}{2} \left(2 + \frac{m_{2-\min} - 1}{2} \right) \\
 &= 1 + \frac{j}{2} + \frac{m_{\min} + m_{2-\min}}{4}.
 \end{aligned}$$

Unfortunately, for general *dags* other than *M*-graphs and the like, calculating the optimal expected makespan is quite hard. Mostly, this is because we do not know the optimal strategy. This is why a decomposition of a *dag* such as in Section 18.9 on page 81 will not work, we cannot rule out that a much lower task is prioritized over a top-level task, and thus, we do not know which tasks to include in the different parts of the decomposition, see Fig. 83. If every source has to be included in the worst case, then the decomposition has only one part which covers the whole *dag*. Of course, this does not help at all.

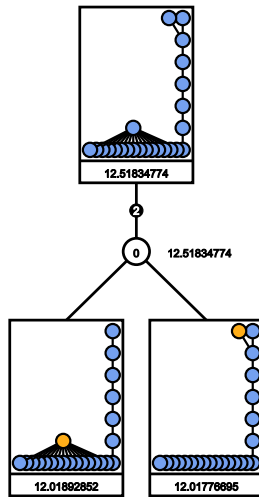


Fig. 83: The level-2 task with 16 successors is prioritized over the level-7 tasks, i.e., $7|6 \prec 2|16$.

30 sp-graphs

A special subclass of *dags* are the *series-parallel graphs* (or short: sp-graphs). These graphs are defined as *two-terminal graphs* (i.e., having exactly one source and one sink) that are built recursively using only two composition operations:

Parallel Composition The disjoint union of two two-terminal graphs, where both the sources and the sinks are merged. See Fig. 84 for an example. The merged nodes are colored in blue.

Series Composition The disjoint union of two two-terminal graphs, where the source of the one is merged with the sink of the other. See Fig. 84 for an example. Note that there are two possible series compositions.

Altogether, a two-terminal sp-graph is one that is built using only the two above operations and starting with copies of the $K_2 = [2]$, i.e., a chain of two nodes. For a more detailed introduction to sp-graphs, see [19] for example.

The usage of sp-graphs as precedence constraints is interesting insofar as for some other scheduling problems, the variant with sp-graphs is easier to solve than the corresponding variant with *dags*, see [36, 38] for example. Unfortunately, for the problem at hand, this restriction does not help us that much. While the overall optimal strategy might be easier to find than for general *dags*, at least we can rule out *HLF*, again. As a counterexample, consider the graph in Fig. 85. The subgraph consisting only of the black nodes is very similar to the graph in Fig. 62. The white nodes are added to obtain an sp-graph.

The reason why we cannot use the graph in Fig. 62 as a counterexample is that it contains the graph in Fig. 86 on the left as a subgraph, which cannot be created as an sp-graph with the levels as given in the figure. Adding the white source node results in a two-terminal graph at least. Although it is possible to create the nearly identical sp-graph in Fig. 86 on the right, the levels of the tasks are not the same. The corresponding precedence relation is also not the same.

As a matter of fact, the number of small sp-graphs that can be created while also restricting ourselves to not changing the levels of the tasks during a composition (apart from adding an offset in case of a series composition, of course) is small. Mostly, because a more elaborate graph like the one in Fig. 86 on the left is not allowed. This way, the sp-graphs we consider are much less complex than general ones.

Although we do not have the optimal strategy, we can determine the expected makespan for some sp-graphs. In particular, for those whose optimal strategy is *HLF*. For example, we have

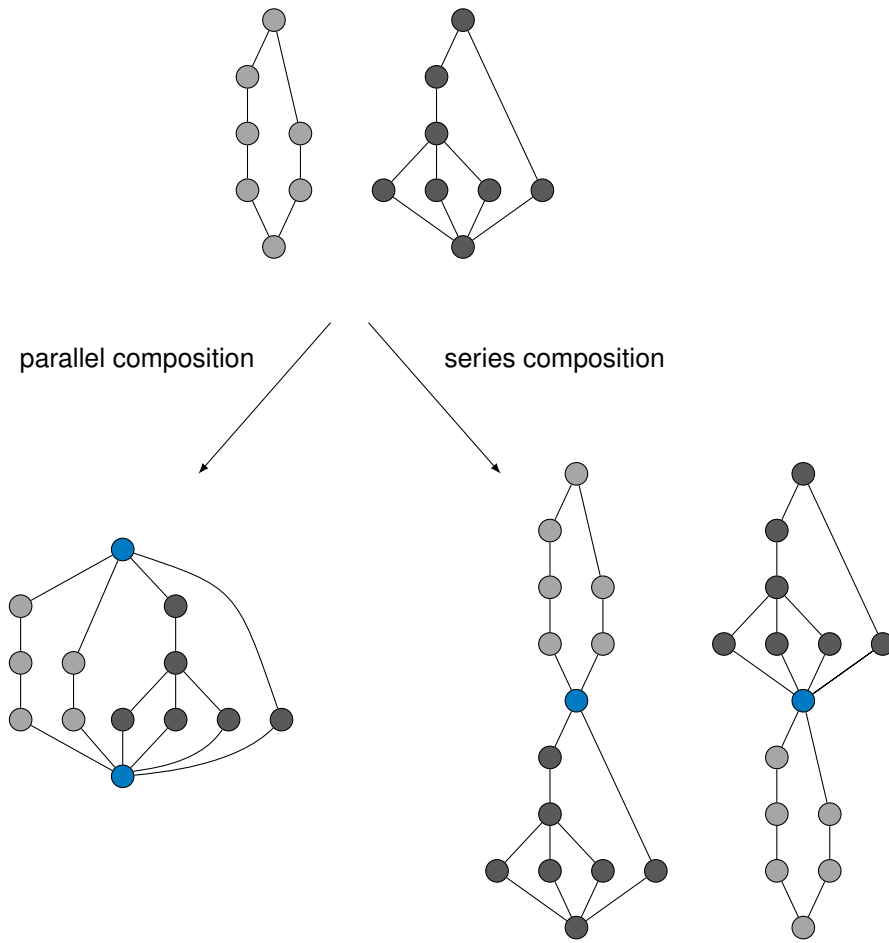


Fig. 84: The parallel and series compositions of two different two-terminal sp-graphs.

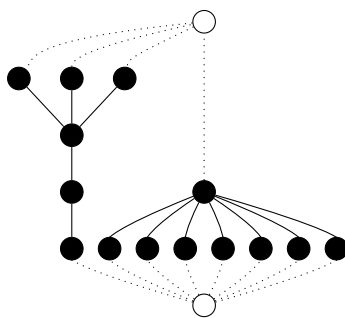


Fig. 85: A counterexample to *HLF* for sp-graphs.

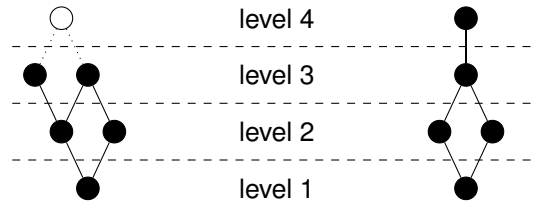


Fig. 86: Not all *dags* can be created as an sp-graph.

the following (the number of tasks in the “diamond graph” depicted by the ellipsis ... is i):

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \ \bullet \\ \bullet \ \bullet \\ \bullet \end{array} \right) &= 1 + C_c(i) + 1 \\
 &= 3 + \frac{i-1}{2}, \\
 C_c \left(\begin{array}{c} \bullet \\ \bullet \ \bullet \\ \bullet \ \bullet \ \bullet \\ \bullet \end{array} \right) &= 1 + C_c([M_i, \underline{1}]) + 1, \\
 C_c \left(\begin{array}{c} \bullet \\ \bullet \ \bullet \\ \bullet \ \bullet \ \bullet \\ \bullet \end{array} \right) &= 1 + C_c \left(\begin{array}{c} \bullet \\ \bullet \ \bullet \\ \bullet \end{array} \right) \\
 &= 1 + \frac{i-1}{2} + \underbrace{C_c \left(\begin{array}{c} \bullet \\ \bullet \ \bullet \\ \bullet \end{array} \right)}_{=C_c(Y_{1,2,1})} \\
 &= 2 + \frac{i-1}{2} + C_c([2, 1]).
 \end{aligned}$$

More complex examples are not really useful, as we would have to calculate the whole configuration graph in order to obtain the makespan. Furthermore, the application of the makespan of an sp-graph is not instantly obvious as well. If we considered a task in a graph and the subgraph rooted at it, i.e., the task and all its descendants, and the corresponding makespan of this sp-graph as a priority for this task, we could examine this further. However, the issues with this approach are: 1) we do not know how to efficiently calculate the makespan of larger, more complex sp-graphs (like the one in Fig. 85, for example), and 2) for a given task in a *dag*, its subgraph need not be an sp-graph, but can be a general *dag* again. This is why we do not investigate sp-graphs further.



Three Processors

Table of Contents

31 Difference to the Deterministic Setting	163
32 Minimal Counterexamples	163
32.1 Supergraphs of Counterexamples	166
32.2 Y-subgraphs of Counterexamples	167
33 Times of Busy and Idle Processors	173
34 More Differences to the Two Processor Case	174
35 <i>HLF</i> is Asymptotically Optimal	174

For several well-known problems the transition of a variable between the values two and three seems to be quite influential in terms of computational complexity. Take the well-known satisfiability problem¹: the problem of 2-Satisfiability is easily solved [35], whereas the problem of 3-Satisfiability is proven to be **NP**-hard [16, 40, 33]. The k -Coloring problem is another example², where the transition between $k = 2$ and $k = 3$ is substantial in terms of complexity. 2-Coloring is just checking for bipartiteness, whereas 3-Coloring is **NP**-hard [33]. So, the question remains whether the scheduling problem of interest is of the mentioned type, or is actually easily solvable for three processors as well. So far, there have not been any proofs regarding the complexity of the three processor problem

$$3|p_i \sim \exp(1); \text{intree}|\mathbb{E}(C_{\max}), \quad (30.1)$$

leaving both options to be possible. However, due to our observations – and due to decades of research without any mentionable output regarding this problem – we suspect that the three processor problem is indeed not as easily solvable as its two processor variant.

31 Difference to the Deterministic Setting

Hu [31] showed that for deterministic processing times *HLF* is optimal, not only for two or three processors, but for an arbitrary and fixed number of processors when regarding intree precedences, i.e., the problem

$$Pm|p_i = p; \text{intree}|C_{\max}.$$

For the two processor problem with stochastic processing times, we have the result of Chandy and Reynolds [49], and even more generalizations can be made while *HLF* still stays the optimal strategy [9]. But as we will see in the next section, *HLF* is no longer optimal when considering exponential processing times and more than two processors.

32 Minimal Counterexamples

One thing that we can say for sure is that the optimal strategy used for the two processor problem, namely *HLF*, is not optimal when we have three processors at our disposal. Consider the two intrees from Fig. 87 and Fig. 88. They are minimal (with respect to the number of tasks) counterexamples for why *HLF* is not always optimal. All smaller intrees, i.e., all intrees with up to ten tasks, are optimally scheduled by any *HLF*. Fig. 87 shows the first three layers of the configuration graph of one of those intrees. There are three (apart from isomorphisms) different choices that a strategy can make in the first step. The *HLF* choice is depicted on the left side with the dashed line (denoting a non-optimal choice): it chooses the two sources on the highest

¹Is there a variable assignment that satisfies a given boolean formula?

²Is there a way of coloring the vertices of a given graph with k colors such that no two neighboring vertices share the same color?

level and the source on the second-highest level. The optimal choice is depicted in the middle with the solid black lines, and the third option is on the left and in gray. The difference between *HLF* and the optimal strategy is small in terms of the resulting expected makespan (less than 0.005), however *HLF* is still strictly worse.

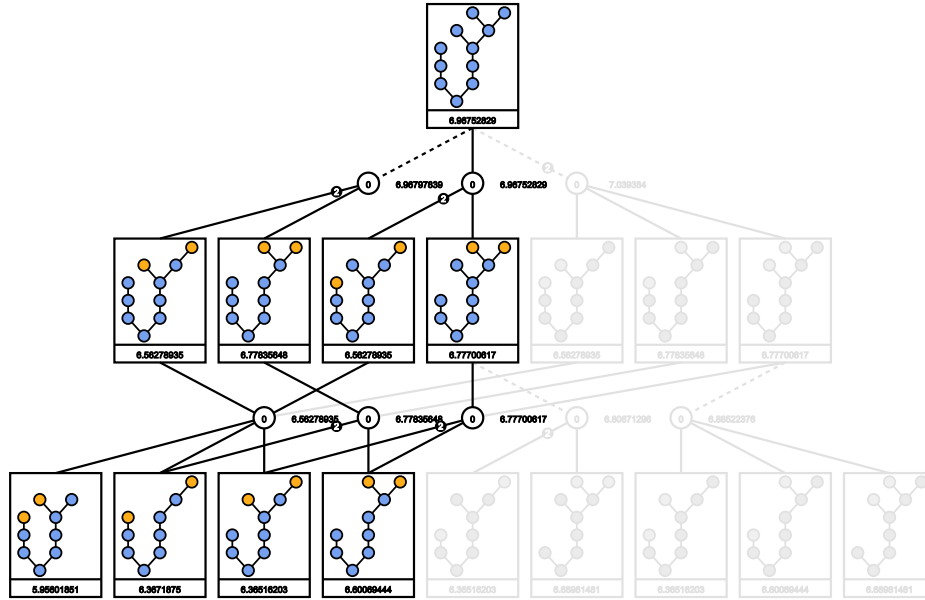


Fig. 87: One of the minimal counterexamples to *HLF*.

The second minimal counterexample from Fig. 88 is similar: the optimal strategy chooses the lowest sources in the first step, which is not done by any *HLF*.

Optimal strategies can be found by using a brute-force method that considers all possibilities and just remembers the best one. Obviously, this algorithm is very inefficient – exponential in the size of the input graph – but for smaller examples with not more than 20 tasks or so, it is fast enough for finding the optimum. Maaß discusses a few exponential-time algorithms in [41]. The ultimate goal of all work focusing on the scheduling problem (30.1) is to successfully classify the complexity of finding an optimal solution to it. As we stated before, so far, no results come to mind. We have seen that *HLF* is not the strategy to go, but we cannot rule out that there is some other – probably a lot more elaborate – strategy find an optimal schedule. And then, the question remains what the complexity of that strategy actually is. For example, we already know that an exponential-time brute-force approach will yield the optimal solution, as we stated before.

Unfortunately, the complexity of (30.1) has been an open problem for some decades now – at least since 1975, when it was mentioned by Chandy and Reynolds in [12]. Much to our regret, this work does not change that fact. However, we try to get a better feeling of the structures of some counterexamples to *HLF*, like in Lemma 76 or Conjecture 77. This may assist with two things: On one hand, we may understand exactly what makes *HLF* a suboptimal strategy

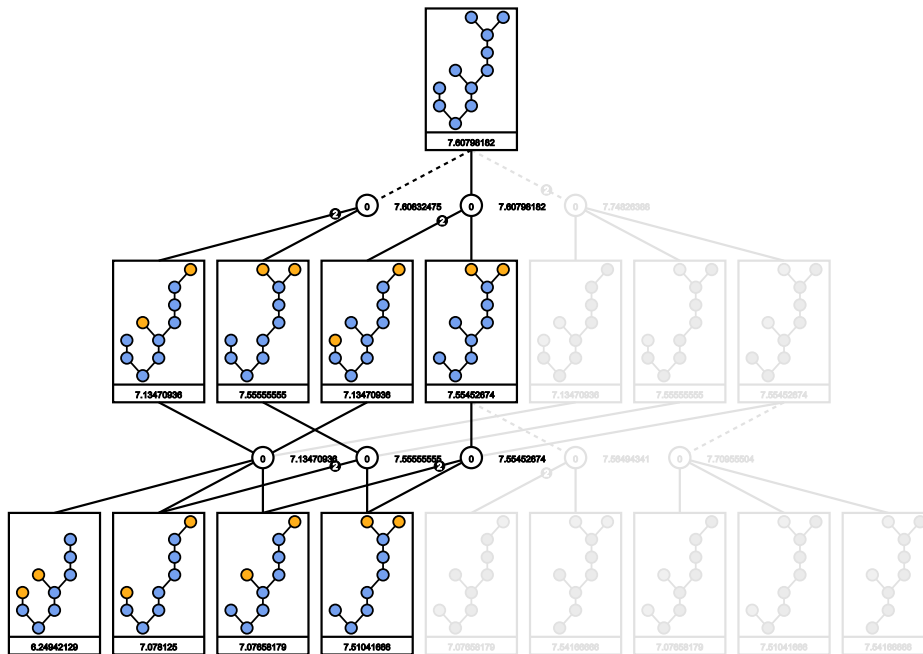


Fig. 88: Another smallest counterexample to *HLF*.

when adding the third processor, and on the other hand, we may find an optimal strategy (or approaches for one) for (30.1).

One property for an optimal scheduling strategy that is already established is that it must not be static [41], like *HLF* for example. It is even shown that a semi-static strategy does not work either, i.e., a strategy that, at every subconfiguration, may take that subconfiguration's structure into consideration for choosing the next sources. Thus, an optimal schedule can only be achieved by using a completely dynamic scheduling strategy, i.e., one that takes already made choices into account. Of course, that does not work at the very beginning of a schedule, because then there are no choices which were already made. So, we try to focus more on those initial configurations as these pose enough of a problem already.

In the following sections, we try to shed light on some counterexamples and their corresponding characteristics, such as the structure of certain reachable configurations, their respective probabilities, and different types of optimality. All simulations as seen in the figures in this section were done using either **tasks time** written by Müller in his master's thesis [43] or **PROSIT** [10], depending on which output is suited better for the respective use. Here, **PROSIT** has the advantage that it can show different strategies all in one picture, whereas **tasks time** can display more interesting values.

32.1 Supergraphs of Counterexamples

Starting from the minimal counterexamples in Fig. 87 and Fig. 88, we search for whole families of counterexamples that have some specific property. Consider the 11-task intree from Fig. 87, and add one more task to it anywhere it may fit. The resulting supergraphs that are also not optimally scheduled by *HLF* can be seen in Fig. 89. The added nodes to obtain the supergraphs are filled with white. All other such 12-node supergraphs are optimally scheduled by *HLF*, apart from isomorphism, of course.

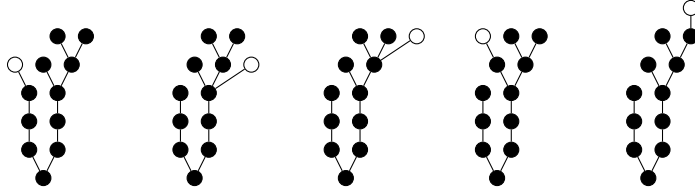


Fig. 89: Supergraphs of the graph in Fig. 87 which are also not optimally scheduled by *HLF*.

We do the same for the intree from Fig. 88 and obtain the intrees in Fig. 90.

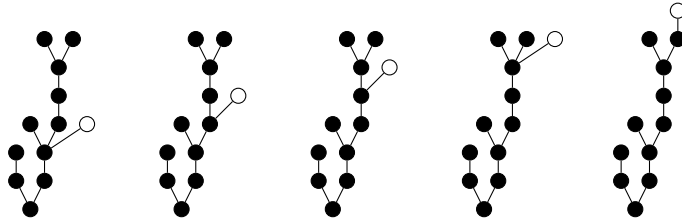


Fig. 90: Supergraphs of the graph in Fig. 88 which are also not optimally scheduled by *HLF*.

In fact, we can generate infinitely many counterexamples to *HLF* with the following Lemma.

Lemma 76. *Let G be any counterexample graph to *HLF* for the three-processor problem, and let L° be the level of the third highest source in G . Then the graph family*

$$\mathcal{G} = \{G^\circ = G \text{ with } x \text{ tasks added above level } L^\circ, x \in \mathbb{N}\}$$

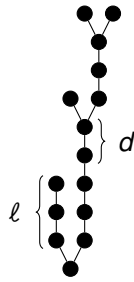
*is a family of counterexamples against *HLF*.*

Proof. For all $G^\circ \in \mathcal{G}$, it holds that G occurs as a subconfiguration with a probability strictly greater than zero when applying *HLF*. But then, this means that G° is not optimally scheduled by any *HLF*. \square

While simulating many examples, we stumbled upon some counterexamples to *HLF* that (supposedly) can be grouped into another family of counterexamples.

Conjecture 77. *Let $G_{d,\ell}$ be a graph as in Fig. 91. Then the graph family*

$$\{G_{d,\ell} : d \geq 0 \text{ and } \ell \geq 2(d+1)\}$$

Fig. 91: $G_{d,\ell}$.

is a family of counterexamples to HLF.

So far, we do not have any proof that this conjecture holds for graphs with larger d and ℓ , but we strongly suspect that this is indeed the case.

32.2 Y-subgraphs of Counterexamples

32.2.1 Fix one Source

An unfavorable case that can happen during scheduling of the tasks is that a task on a high level takes many time steps to be processed. In this case, it may be that many more tasks on lower levels have to wait for either this task to be finished because they are descendants of it, or they are sources but have to wait for this one processor to become idle. So, one approach to consider is to fix one source and calculate the rest of the graph as a two processor problem. This models the absolute worst case, where this one source takes longer than all other tasks combined. While this is not very likely in most cases, it is always a possibility. The idea is to compare the values and try to reduce the possibility of such a degenerate configuration.

There are different characteristics that we can focus on:

- The expected makespan of the chain with the fixed source.
- The probability of reaching that configuration.
- The expected makespan of the remaining graph/inforest after the chain with the fixed source is removed.
- The product of the probability of reaching that configuration with the sum of the expected makespans of the chain with the fixed source and the remaining inforest.

The first value only depends on the level of the source. The second value is a little bit complicated, but can be computed in the same way as in Section 18.7 on page 66, for example using the alternative approach of the end of the section. The problem with this is that it actually depends on the strategy that is used. For some strategies, certain configurations may never occur, so the values for the probabilities that we present here in the following are calculated using the

strategy that randomly selects a source (all sources are equally likely). Of course, we can argue whether this makes sense or if we should rather try different strategies and compare the corresponding probabilities of them. The third value can be calculated using one of the many different approaches presented in Chapter 18. The fourth value combines all three values before, and the idea behind that value is that this is the weighted factor of that degenerate configuration within the calculation for the expected makespan of the whole graph. The table in Fig. 92 shows the values of the above characteristics for the subconfigurations where one source is fixed and it and its dominating path are removed from the graph. Most values are rounded, but given with a precision error of 10^{-4} . The underlying graph is the one from Fig. 87, i.e., one of the smallest counterexamples to *HLF*. The optimal strategy initially chooses to process the gray tasks x_1 , x_2 , and x_4 .

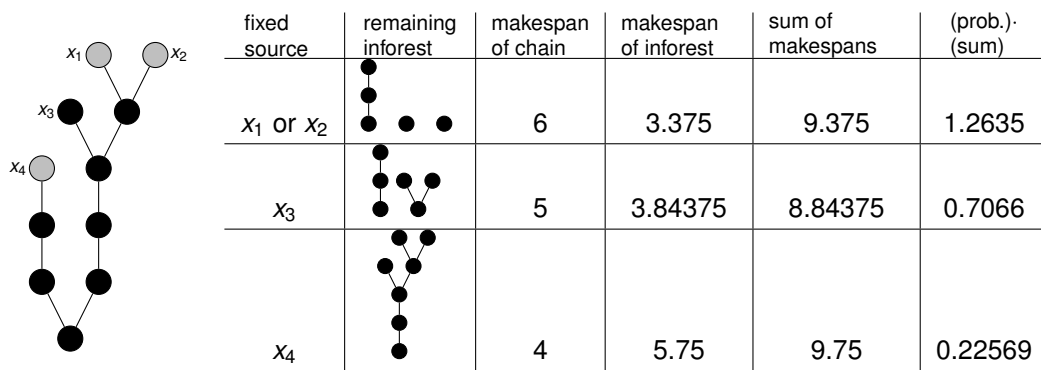


Fig. 92: The product of probability and sum of makespans does not work as a priority.

We observe that the values in the last column cannot be used as a priority for choosing optimal tasks alone. In this case, we would have to choose the tasks with the highest and the lowest values. If we used the values in the second-to-last column as a priority, we see that in this case, we would choose the correct sources that lead to an optimal makespan. However, consider the graph on the left in Fig. 93. It is optimally scheduled by *HLF*, initially choosing the gray tasks. The different values for its subconfigurations are in the table on the right. From these we can see that using the sum of the makespans of the chain and the remaining inforest cannot be used as a priority for an optimal scheduling strategy.

In Fig. 93 we omit the last column with the product of the probability with the sum of the makespans mainly because it does not yield a reasonable output that might be put into an optimal strategy anyway (as we have seen in the table in Fig. 92).

32.2.2 Fix two Sources

Given a graph, consider subgraphs with exactly two sources, both of which are sources in the original graph as well. These Y-graphs, cp. Section 18.8, are optimally scheduled by any reasonable strategy that does not leave processors unnecessarily idle, and their expected makespans can be calculated quite easily, see Sections 18.6, 18.6.2 and 18.8 on pages 57 ff. The idea

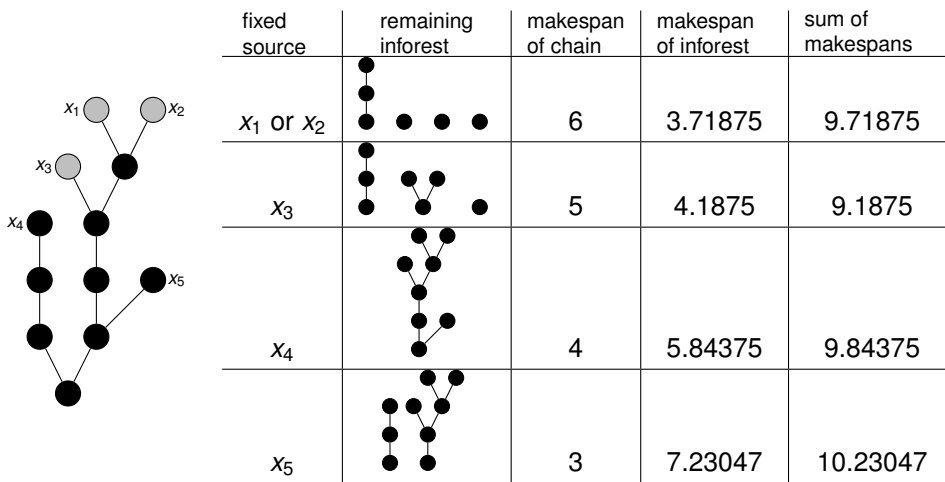


Fig. 93: The sum of makespans alone does also not work as a priority.

is to see how different subconfigurations, or subgraphs, respectively, and their makespans behave according to whether *HLF* or a non-*HLF* is optimal. This means that, in contrast to Section 32.2.1 where we fixed one source and considered the remaining inforest as a two processor problem, we fix two sources and consider the remaining inforest as a one processor problem. This models the case where two tasks take very long to be processed and the other tasks only have one processor to be processed. See Fig. 94 for an example.

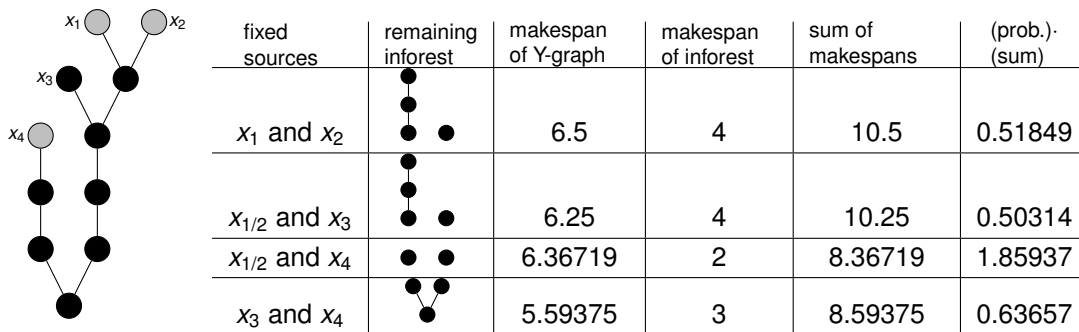


Fig. 94: The values for the example graph from Fig. 92, this time with two fixed sources.

The observation here is similar to the case where we fixed only one source: the makespans alone do not seem to provide any insight into how an optimal strategy chooses its tasks. Neither does the weighted sum of the makespans in the last column. Now that we consider tuples of tasks, it is not even obvious how to compare these values to help some strategy to choose the tasks. We can consider all three tuples of three tasks and compare their values to all the others. In this case, this does not help us anything. The second-to-last column, however, may be used to derive an optimal strategy since the values do work as a priority here. But again, as

before, we see that these values do not provide anything when considering another example, see Fig. 95.

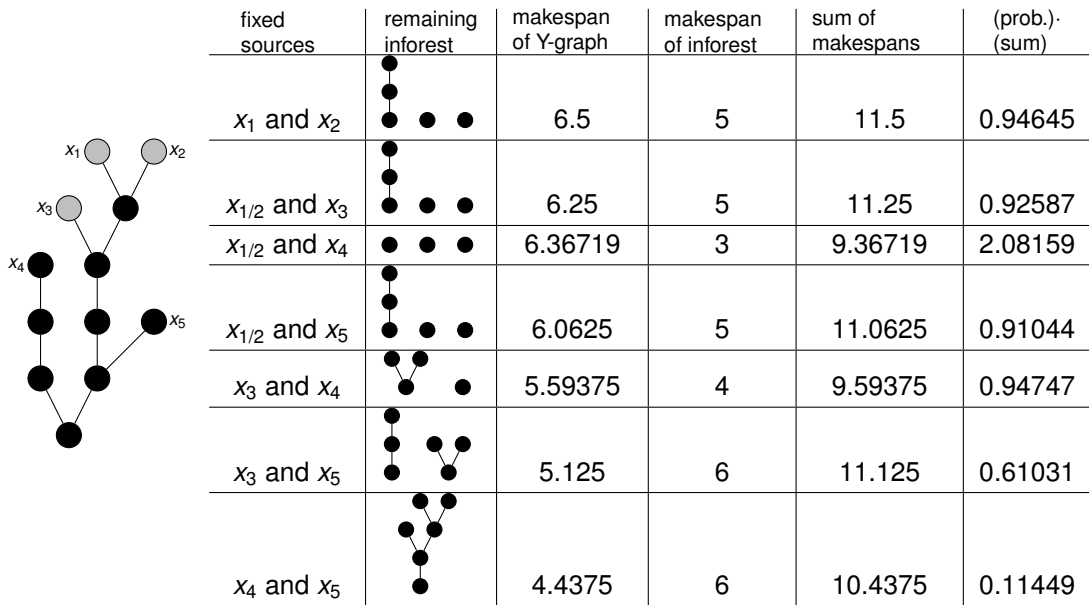


Fig. 95: The values for the example graph from Fig. 93, this time with two fixed sources.

32.2.3 Probability of Reaching Certain Y-subgraphs under Different Strategies

Consider the counterexample seen in Fig. 88. Then, if we compare the two configuration graphs of the optimal strategy and *HLF*, we see that the probability of certain configurations of Y-subgraphs differ vastly. In the end, this is what makes the difference between the optimal and a suboptimal strategy (among other things): the optimal strategy reduces the probability of these degenerate configurations. See Fig. 96 and Fig. 97 for examples, the values at the top of each configuration corresponds to its respective occurrence probability – these differ when applying different strategies, or course. The examples show a huge difference between the probabilities of reaching the $Y_{1,2,6}$ -configuration (third level from the top and third from the left in each of the figures), it is $0.\bar{4}$ for *HLF*, but $0.\bar{2}$ for the optimal strategy, i.e., it is only half as likely to be reached in an optimal schedule than it is in an *HLF* schedule. However, the $Y_{3,1,4}$ -configuration is twice as likely to be reached in an optimal schedule with probabilities $0.\bar{148}$ against $0.\bar{074}$, but this does not seem to prevent the strategy from being optimal.

Fig. 96 and Fig. 97 were created using **tasks time** from [43]. This tool was chosen for the visualization here because it shows the probabilities according to the chosen strategy. Tasks that are chosen by the respective strategy and/or are currently active are marked with the crossed nodes as usual. Here, we do not make the distinction between chosen tasks (formerly marked by filled gray nodes) and active tasks (crossed nodes), as it would blow up the figures even more. The value at the top of each configuration is its occurrence probability.

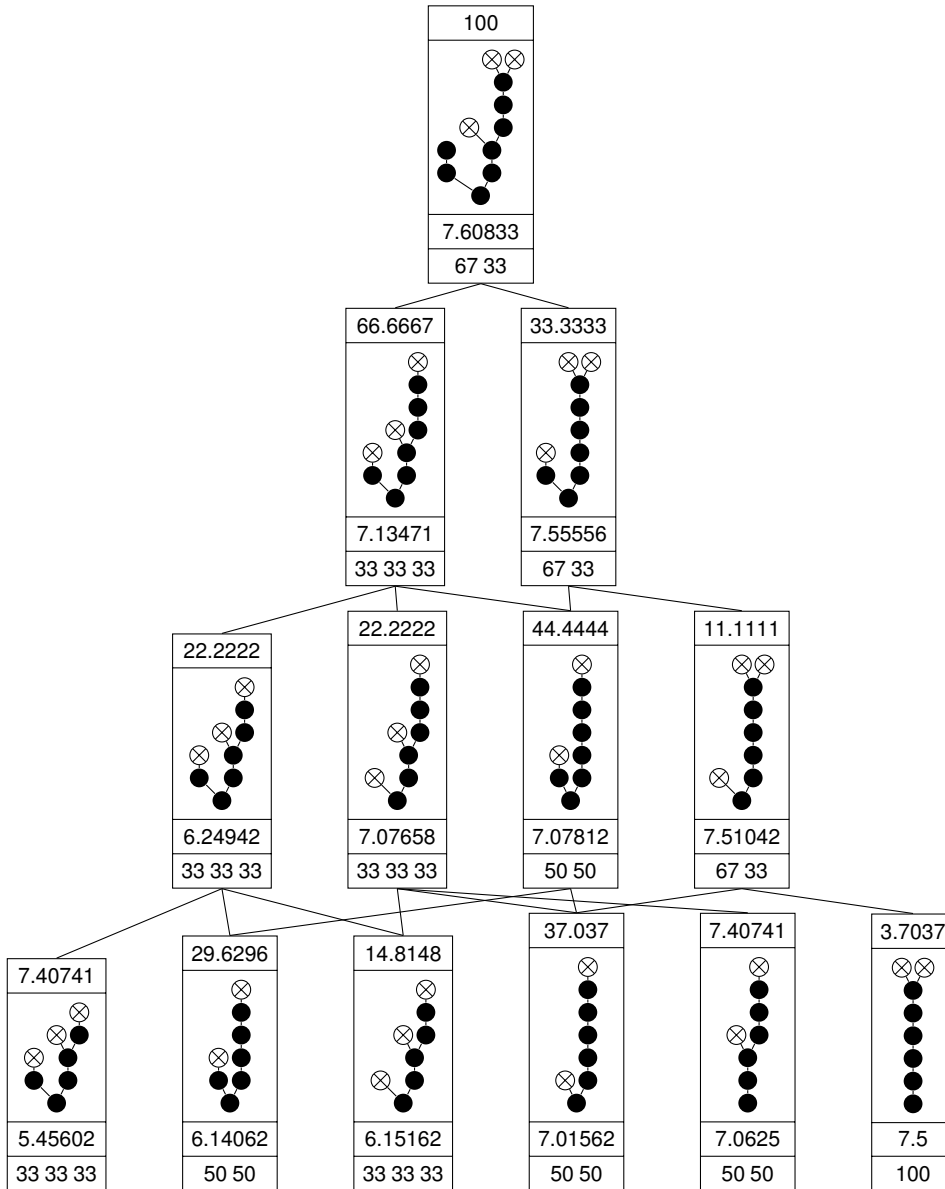


Fig. 96: Part of the configuration graph of the graph in Fig. 88 using *HLF*.

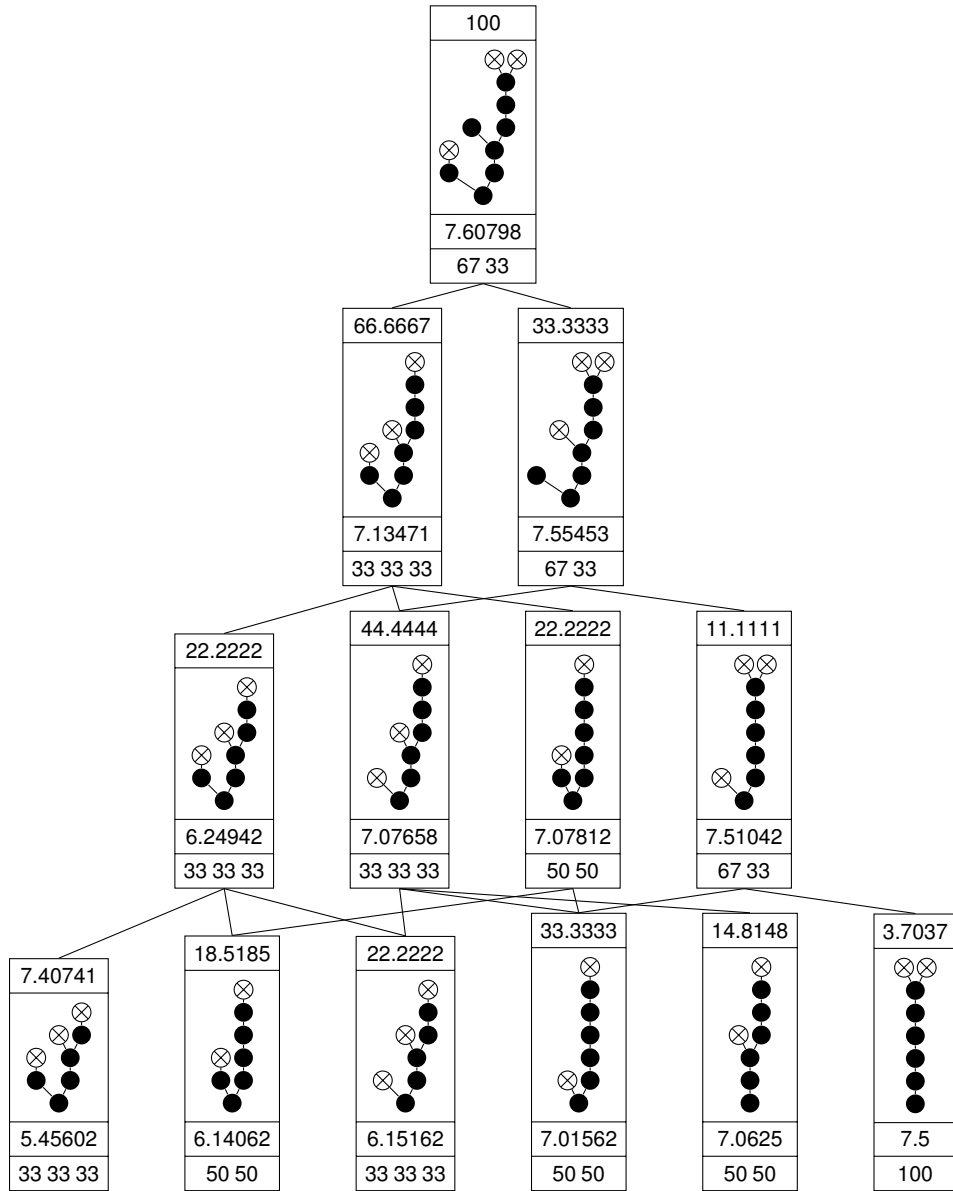


Fig. 97: Part of the configuration graph of the graph in Fig. 88 using the optimal strategy.

The problem is how can we distinguish between Y-subgraphs to tell which ones are the important (or crucial) ones to focus on. Also, it may very well be the case that we cannot deduce any crucial information from the Y-subgraphs alone.

33 Times of Busy and Idle Processors

Suppose the expected processing time of a given intree for any scheduling strategy \mathcal{S} is given by $T^{\mathcal{S}}$, then we can define the expected time this strategy \mathcal{S} keeps exactly three, exactly two, and exactly one processor(s) busy, denoted by $T_3^{\mathcal{S}}, T_2^{\mathcal{S}}, T_1^{\mathcal{S}}$, respectively. The intuitive approach to maximize $T_3^{\mathcal{S}}$ does not yield an optimal strategy, see Fig. 98 on the left. This graph gives us two (reasonable) different strategies to choose the tasks in the first step. The optimal schedule, i.e., *HLF*, has $T_3^* = 0.7$, whereas the suboptimal schedule has $T_3 = 0.8518$. Similar to that, trying to minimize $T_1^{\mathcal{S}}$ does not yield an optimal strategy, either, see Fig. 98 on the right. For this graph, we can choose between different *HLFs* in the first step. The optimal strategy chooses the rightmost source and two of the other three sources to obtain a schedule that has $T_1^* = 2.592$, whereas a suboptimal schedule that is obtained by not choosing the rightmost source has $T_3 = 2.5$. By T^* and T_i^* , $i = 1, 2, 3$, we denote the corresponding values for the optimal strategy. Again, note that this does not necessarily mean that these are the smallest (or highest) values that any scheduling strategy can achieve.

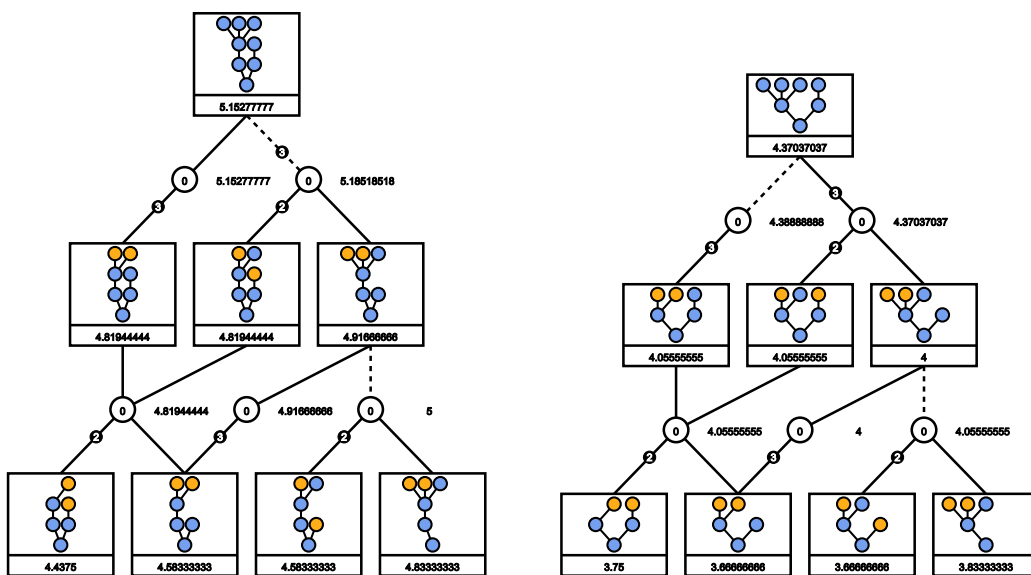


Fig. 98: Optimizing T_1 or T_3 does not necessarily yield an optimal schedule.

Another thing that can be seen in Fig. 98 on the right is that different *HLFs* can result in different expected makespans. So, unlike in the two processor case, not all *HLFs* are equal in the three processor case, i.e., Lemma 29 on page 39 does not hold here.

However, if we consider a combination of both values, in particular, we want to have the value $T_1 - T_3$ minimized – hoping that a longer expected time on only one processor can be countered by a longer time on three processors – then this value is indeed minimized by all optimal strategies in the example graphs presented here. There is no further evidence for larger and more complex graphs, but we may conjecture that this condition holds as well. Nevertheless, we have not gone into more detail on this, because even if all optimal strategies had something in common that can be calculated with only the T_i values, then it would still not be clear how to derive a strategy out of it as these values can not be easily calculated without knowing the strategies beforehand. Maybe there is a connection of the T_i^* values with the structure of the graph. Then, this may be a promising approach to focus on.

34 More Differences to the Two Processor Case

In Fig. 98 we have seen that not every *HLF* results in the same makespan. In addition to this, another result from the two processor case does not hold either. Consider the two intrees in Fig. 99. They both have the same profile $(1, 2, 2, 2, 1, 1, 1, 2)$, but the left one is optimally scheduled by *HLF*, while the right one is optimally scheduled by a non-*HLF*. The gray tasks mark the ones which are initially chosen by the optimal strategy, respectively. This is in accordance with the observations made in Chapter 17 in Fig. 7 on page 40, where we argued that the profile is only a reasonable characteristic of a graph when we have at most two processors available. For more than two processors, the profile can not give the crucial information about the number of sources.

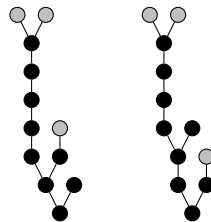


Fig. 99: Same profiles do not infer same optimal strategies.

35 *HLF* is Asymptotically Optimal

Despite all the negative results brought forward about *HLF* in the sections before, it still is a very good strategy, though not optimal. Papadimitriou and Tsitsiklis [45] showed that for an

arbitrary, but fixed number of processors, any *HLF* will be asymptotically optimal. This means that, as the number of tasks tends to infinity, the ratio of the optimal expected makespan and the expected makespan stemming from *HLF* tends to 1. Practically, this may be good enough when dealing with large inputs anyway and opens the question whether trying to solve the problem (30.1) is worth it as nowadays efficient approximations of optimal results are very wide-spread in practice. For the theoreticians, this is definitely no satisfying answer.

Conclusion

In this thesis, in particular in Part V, we had a detailed look into the results proposed by Chandy and Reynolds about the $2|p_j \sim \exp(1); \text{intree} | \mathbb{E}(C_{\max})$ scheduling problem. We rewrote their proof showing that any *HLF* is the optimal strategy, and took a lengthy discourse on the calculation of the optimal expected makespan for a given input intree in Chapter 16. Several, equivalent approaches on how to actually calculate this have been proposed. In the special case of only two chains as the precedence constraints, we gave a formula based on combinatorial methods in Theorem 43, but also another formula that was obtained by using the recursive nature of the problem and generating functions in Theorem 45. For the more general case with more than two chains, we used the structure of the configuration graph to define a so-called stair function and provided a formula for the optimal expected makespan in Theorem 49. This stair function proved to be useful in the general case as well where we gave a recursive way for calculating the expected makespan via a special decomposition of an intree in Corollary 54.

Furthermore, we showed that *HLF* is the optimal strategy if the underlying distribution describing the tasks' processing times is memoryless, no matter if it is continuous or discrete, in Part VI. An interesting task for future work could be to determine whether this is actually an equivalence and *HLF* is only optimal if and only if the distribution is memoryless. Chandy and Reynolds [12] stated that *HLF* is still optimal for Erlang processing times, but never backed up that claim with a proof, indicating that this might not be true. Any of these possibilities would be a great result, which we unfortunately did not manage to show. However, we examined the scheduling problem with Erlang processing times and came closer to believing that *HLF* is indeed optimal for Erlang processing times in Chapter 27.

As for the scheduling problem with uniform processing times (discrete or continuous) we strongly believe that *HLF* is not the optimal strategy, but, could not succeed in finding a counterexample. We provided some observations that lead us to believe that *HLF* may not work, but cannot support this conjecture any further. A tool was written to examine several strategies and their corresponding makespan [22], but the method of calculating the makespan becomes quite inefficient for larger input intrees. We suspect that counterexamples to *HLF* for uniform processing times have at least 10 tasks, which cannot be found in short time on a standard machine with the given tool.

Moreover, we did case studies on several types of strategies for general precedence constraints, ruling out several characteristics of a graph to be used as a priority.

We followed with insights into the scheduling problem with three processors, exponential processing times and intree precedence constraints, again showing that *HLF* is not optimal. Although we could not establish an optimal scheduling strategy, we examined several approaches and proved their non-optimality in this case.



Detailed Calculations

A Equivalence of Approaches for Exponential Processing Times

In Chapter 16 on page 36, we gave several approaches for how to calculate the expected makespan of

$$2|p_j \sim \exp(1); \text{intree} | \mathbb{E}(C_{\max}).$$

In the following, we want to show the equivalence of the three approaches from Theorem 43, Lemma 37, and Theorem 45 on pages 50 ff. for a small example intree. For this, consider exponential processing times, two processors, and the intree [5, 3] from Fig. 100.

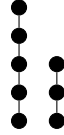


Fig. 100: The intree [5, 3].

Calculation 1. Consider the graph from Fig. 100. Then the optimal expected makespan is, using (18.5) on page 59,

$$\begin{aligned} C_c([5, 3]) &= 2^{-9} \left(\sum_{s'=1}^5 2^{s'} \binom{7-s'}{2} (8+s') + \sum_{t'=1}^3 2^{t'} \binom{7-t'}{4} (8+t') \right) \\ &= 2^{-9} (2190 + 558) \\ &= \frac{687}{128}, \end{aligned}$$

and, using the recursive formula from Lemma 37,

$$\begin{aligned} C_c([5, 3]) &= \frac{1}{2} + \frac{1}{2} C_c([4, 3]) + \frac{1}{2} C_c([5, 2]) \\ &= \frac{1}{2} + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([3, 3]) + \frac{1}{2} C_c([4, 2]) \right) + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([4, 2]) + \frac{1}{2} C_c([5, 1]) \right) \\ &= 1 + \frac{1}{4} C_c([3, 3]) + \frac{1}{2} C_c([4, 2]) + \frac{1}{4} C_c([5, 1]) \\ &= 1 + \frac{1}{4} \left(\frac{1}{2} + C_c([3, 2]) \right) + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([3, 2]) + \frac{1}{2} C_c([4, 1]) \right) \\ &\quad + \frac{1}{4} \left(\frac{1}{2} + \frac{1}{2} C_c([4, 1]) + \frac{1}{2} \underbrace{C_c([5, 0])}_{=5} \right) \\ &= \frac{17}{8} + \frac{1}{2} C_c([3, 2]) + \frac{3}{8} C_c([4, 1]) \\ &= \frac{17}{8} + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c([2, 2]) + \frac{1}{2} C_c([3, 1]) \right) + \frac{3}{8} \left(\frac{1}{2} + \frac{1}{2} C_c([3, 1]) + \frac{1}{2} \underbrace{C_c([4, 0])}_{=4} \right) \\ &= \frac{53}{16} + \frac{1}{4} C_c([2, 2]) + \frac{7}{16} C_c([3, 1]) \end{aligned}$$

$$\begin{aligned}
&= \frac{53}{16} + \frac{1}{4} \left(\frac{1}{2} + C_c([2, 1]) \right) + \frac{7}{16} \left(\frac{1}{2} + \frac{1}{2} C_c([2, 1]) + \frac{1}{2} \underbrace{C_c([3, 0])}_{=3} \right) \\
&= \frac{138}{32} + \frac{15}{32} C_c([2, 1]) \\
&= \frac{138}{32} + \frac{15}{32} \left(\frac{1}{2} + \frac{1}{2} C_c([1, 1]) + \frac{1}{2} \underbrace{C_c([2, 0])}_{=2} \right) \\
&= \frac{321}{64} + \frac{15}{64} C_c([1, 1]) \\
&= \frac{321}{64} + \frac{15}{64} \left(\frac{1}{2} + \underbrace{C_c([1, 0])}_{=1} \right) \\
&= \frac{687}{128},
\end{aligned}$$

giving an example that, indeed, the two ways to calculate the expected makespan are equivalent. The third equivalent way from Theorem 45 gives us

$$\begin{aligned}
C_c([5, 3]) &= \sum_{k=0}^5 \sum_{\ell=0}^3 \left(\frac{1}{2} \right)^{k+\ell+1} \binom{k+\ell}{\ell} + \sum_{k=0}^5 \left(\frac{1}{2} \right)^{k+4} \binom{k+3}{3} (5-k) \\
&\quad + \sum_{\ell=0}^3 \left(\frac{1}{2} \right)^{\ell+6} \binom{\ell+5}{5} (3-\ell) - \left(\frac{1}{2} \right)^9 \binom{8}{3} \\
&= \frac{451}{128} + \frac{443}{256} + \frac{57}{256} - \frac{7}{64} \\
&= \frac{687}{128}.
\end{aligned}$$

B Equivalence of Approaches for Geometric Processing Times

For geometric processing times, we also have several approaches for how to calculate the expected makespan. In the same manner as in Appendix A, we want to give an example for the equivalence of the approaches from (19.1) and (19.2) on pages 93-92. We use the intree in Fig. 101 for this.



Fig. 101: An intree with expected makespan $\frac{190}{27}$.

Calculation 2. Consider the intree in Fig. 101, and geometric processing times with parameter $p = \frac{1}{2}$, then with (19.2), we have

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) &= \frac{4}{3} + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \end{array} \right) + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) \\ &= \frac{4}{3} + \frac{1}{3} \cdot \frac{56}{9} + \frac{1}{3} \cdot \frac{56}{9} + \frac{1}{3} \cdot \frac{14}{3} \\ &= \frac{190}{27}, \end{aligned}$$

where we used the intermediate results

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) &= \frac{4}{3} + 2 \cdot \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{1}{3} C_c \left(\bullet \right) \\ &= \frac{4}{3} + \frac{2}{3} \cdot 4 + \frac{1}{3} \cdot 2 \\ &= \frac{14}{3}, \end{aligned}$$

and

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \end{array} \right) &= \frac{4}{3} + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \end{array} \right) + \frac{1}{3} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) \\ &= \frac{4}{3} + \frac{1}{3} \cdot 6 + \frac{1}{3} \cdot \frac{14}{3} + \frac{1}{3} \cdot 4 \\ &= \frac{56}{9}. \end{aligned}$$

Considering the other approach from (19.1), for the same problem, we have

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) &= \sum_{k=1}^{\infty} \sum_{\ell=1}^{\infty} \left(\frac{1}{2} \right)^2 \left(1 - \frac{1}{2} \right)^{k+\ell-2} (\min(k, \ell) + C_c(G^\circ)) \\ &= \sum_{k=1}^{\infty} \sum_{\ell=k+1}^{\infty} \left(\frac{1}{2} \right)^{k+\ell} \left(k + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \end{array} \right) \right) \\ &\quad + \sum_{k=1}^{\infty} \sum_{\ell=1}^{k-1} \left(\frac{1}{2} \right)^{k+\ell} \left(\ell + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \end{array} \right) \right) \\ &\quad + \sum_{k=1}^{\infty} \left(\frac{1}{2} \right)^{2k} \left(k + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \end{array} \right) \right) \\ &= \sum_{k=1}^{\infty} \sum_{\ell=k+1}^{\infty} \left(\frac{1}{2} \right)^{k+\ell} \left(k + \frac{56}{9} \right) \\ &\quad + \sum_{k=1}^{\infty} \sum_{\ell=1}^{k-1} \left(\frac{1}{2} \right)^{k+\ell} \left(\ell + \frac{56}{9} \right) \\ &\quad + \sum_{k=1}^{\infty} \left(\frac{1}{2} \right)^{2k} \left(k + \frac{14}{3} \right) \\ &= \frac{190}{27}, \end{aligned}$$

where we need to calculate the same intermediate results again, using the other approach this time:

$$\begin{aligned}
 C_c(\bullet\bullet\bullet) &= \sum_{k=1}^{\infty} \sum_{\ell=k+1}^{\infty} \left(\frac{1}{2}\right)^{k+\ell} \left(k + \underbrace{C_c(\bullet)}_{=4} \right) \\
 &\quad + \sum_{k=1}^{\infty} \sum_{\ell=1}^{k-1} \left(\frac{1}{2}\right)^{k+\ell} (\ell + C_c(\bullet\bullet)) \\
 &\quad + \sum_{k=1}^{\infty} \left(\frac{1}{2}\right)^{2k} \left(k + \underbrace{C_c(\bullet)}_{=2} \right) \\
 &= \frac{14}{3},
 \end{aligned}$$

and

$$\begin{aligned}
 C_c(\bullet\bullet\bullet\bullet) &= \sum_{k=1}^{\infty} \sum_{\ell=k+1}^{\infty} \left(\frac{1}{2}\right)^{k+\ell} \left(k + \underbrace{C_c(\bullet\bullet\bullet)}_{=6} \right) \\
 &\quad + \sum_{k=1}^{\infty} \sum_{\ell=1}^{k-1} \left(\frac{1}{2}\right)^{k+\ell} (\ell + C_c(\bullet\bullet\bullet\bullet)) \\
 &\quad + \sum_{k=1}^{\infty} \left(\frac{1}{2}\right)^{2k} (k + C_c(\bullet\bullet)) \\
 &= \frac{56}{9}.
 \end{aligned}$$


C Induction Bases for Exponential Processing Times

The following calculations show the details of the induction base in Lemma 31 on page 41.

Calculation 3. [The expected makespan of $\bullet\bullet\bullet\bullet$]

We have

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) &= \frac{1}{2} + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array} \right) \\
 &= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{1}{2} \underbrace{C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)}_{=3} \\
 &= \frac{5}{2} + \frac{1}{2} \left(\frac{1}{2} + \underbrace{C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right)}_{=2} \right) \\
 &= \frac{5}{2} + \frac{5}{4} \\
 &= 3\frac{3}{4}.
 \end{aligned}$$


Calculation 4. [The expected makespan of ]
For the second intree in Lemma 31, it holds that

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) &= \frac{1}{2} + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array} \right) \\
 &= \frac{1}{2} + \frac{1}{2} + \underbrace{C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)}_{=3} \\
 &= 4.
 \end{aligned}$$


Putting together Calculations 3 and 4 gives us

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) < C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right).$$

For the induction base Lemma 33, we have the following calculations.

Calculation 5. [The expected makespan of ]
It holds that

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) &= \frac{1}{2} + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \end{array} \right) \\
 &= \frac{1}{2} + \frac{1}{2} + C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) \\
 &= 1 + \frac{1}{2} + C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) \\
 &= 3\frac{1}{2}.
 \end{aligned}$$


Calculation 6. [The expected makespan of  using non-*HLF*] The assumption for the non-*HLF* is that only in the very first step, the choices are not according to “highest-level-first”. Thus,

$$\begin{aligned}
 C_c^{\text{non-}HLF} \left(\text{tree} \right) &= \frac{1}{2} + C_c \left(\text{tree} \right) \\
 &= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} C_c \left(\text{tree} \right) + \frac{1}{2} \underbrace{C_c \left(\text{tree} \right)}_{=3} \\
 &= \frac{5}{2} + \frac{1}{2} \left(\frac{1}{2} + \underbrace{C_c \left(\text{tree} \right)}_{=2} \right) \\
 &= \frac{5}{2} + \frac{5}{4} \\
 &= 3\frac{3}{4}.
 \end{aligned}$$

With Calculations 5 and 6, we have

$$C_c \left(\text{tree} \right) < C_c^{\text{non-}HLF} \left(\text{tree} \right),$$

proving the first half of the induction base of Lemma 33. For the second half, we have the following.

Calculation 7. [The expected makespan of 

We get

$$\begin{aligned}
 C_c \left(\text{tree} \right) &= \frac{1}{2} + \frac{1}{2} C_c \left(\text{tree} \right) + \frac{1}{2} C_c \left(\text{tree} \right) \\
 &= \frac{1}{2} + \frac{1}{2} \left(\frac{1}{2} + C_c \left(\text{tree} \right) \right) + \frac{1}{2} \left(\frac{1}{2} + \frac{1}{2} C_c \left(\text{tree} \right) + \frac{1}{2} \underbrace{C_c \left(\text{tree} \right)}_{=3} \right) \\
 &= \frac{1}{2} + \frac{1}{4} + \frac{3}{4} C_c \left(\text{tree} \right) + \frac{1}{4} + \frac{3}{4} \\
 &= \frac{7}{4} + \frac{3}{4} \left(\frac{1}{2} + \underbrace{C_c \left(\text{tree} \right)}_{=2} \right) \\
 &= 3\frac{5}{8}.
 \end{aligned}$$

And we have

$$C_c \left(\text{tree} \right) < C_c^{\text{non-}HLF} \left(\text{tree} \right),$$

hence, the induction base holds.


D Induction Bases for Geometric Processing Times

With these calculations we want to show the details of the inequality in (20.1) from Lemma 60 in Chapter 20 on page 95.

First, we observe that

$$\frac{1}{1 - (1 - p)^2} = \frac{1}{p(2 - p)},$$

which will simplify the following calculations.


Calculation 8. [The expected makespan of ]

First, we calculate an intermediate result we need.

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c (\bullet) \\ &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} \cdot \frac{3}{p} + \frac{p}{2-p} \cdot \frac{2}{p} \\ &= \frac{7-4p}{p(2-p)}. \end{aligned}$$

Then we have

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) \\ &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} \cdot \frac{7-4p}{p(2-p)} + \frac{p}{2-p} \cdot \frac{3}{p} \\ &= \frac{5p^2 - 17p + 16}{p(2-p)^2}. \end{aligned}$$

Calculation 9. [The expected makespan of ]

For the second value, we calculate the intermediate results

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c (\bullet) + \frac{p}{2-p} C_c (\bullet) \\ &= \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} \cdot \frac{2}{p} + \frac{p}{2-p} \cdot \frac{1}{p} \\ &= \frac{5-3p}{p(2-p)} \end{aligned}$$

and

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1}{p(2-p)} + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c (\bullet) \\ &= \frac{1}{p(2-p)} + \frac{1-p}{2-p} \cdot \frac{3}{p} + \frac{1-p}{2-p} \cdot \frac{5-3p}{p(2-p)} + \frac{p}{2-p} \cdot \frac{2}{p} \\ &= \frac{p^2 - 6p + 8}{p(2-p)^2}. \end{aligned}$$

We use these two to obtain

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1}{p(2-p)} + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) \\ &= \frac{1}{p(2-p)} + \frac{1-p}{2-p} \cdot \frac{7-4p}{p(2-p)} + \frac{1-p}{2-p} \cdot \frac{p^2-6p+8}{p(2-p)^2} + \frac{p}{2-p} \cdot \frac{3}{p} \\ &= \frac{2p^2-11p+13}{p(2-p)^2}. \end{aligned}$$

Calculation 10. The inequality (20.1) can be proven by observing that

$$5p^2 - 17p + 16 - (2p^2 - 11p + 13) = 3p^2 - 6p + 3 = 3(1-p)^2$$

which is clearly nonnegative. Furthermore, $C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right)$ and $C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)$ are equal if and only if $p = 1$, which is a degenerate case anyway.

What follows are calculations for the details in Lemma 61 on page 96.

Calculation 11. It holds that

$$C_c \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)$$

and

$$C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right).$$

We show that $C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) < C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)$ and $C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) < C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)$, respectively. For the first inequality, we observe that the difference between the values is

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) - C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{1-p}{2-p} \left(C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) - C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) \right)$$

and using (19.1) to calculate these values gives us

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{5-3p}{p(2-p)} < \frac{6-3p}{p(2-p)} = \frac{3}{p} = C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right), \quad (\text{D.1})$$

and thus,

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) < C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right).$$

For the second inequality, we do the same and consider the difference $C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) - C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right)$, which we have already proven to be negative in (D.1). Hence,

$$C_c \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) < C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right).$$

Calculation 12. We have

$$C_c \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{1}{p(2-p)} + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right)$$

and

$$C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) = \frac{1}{p(2-p)} + 2 \cdot \frac{1-p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) + \frac{p}{2-p} C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right).$$


Again, we look at the difference of those values and obtain

$$\begin{aligned} C_c \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) - C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1-p}{2-p} \left(C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) - C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) \right) \\ &\quad + \frac{p}{2-p} \left(C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) - C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) \right). \end{aligned}$$

But we know from Calculation 11 that the values in the parentheses are both negative, i.e.,

$$C_c \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right) < C_c^{\text{non-HLF}} \left(\begin{array}{c} \circ \\ \bullet \\ \bullet \\ \bullet \end{array} \right).$$

E Induction Bases for Uniform Processing Times

Calculation 13. [The expected makespan of 

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) &= \frac{1}{4} \left(a + C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{2} \left(a + C_c \left(\begin{array}{c} a \\ \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{4} \left(b + C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) \right) \\ &= \frac{1}{8} (2a + 3a + 3b + 2b + 3a + 3b + 4a) + \frac{1}{2} \begin{cases} 2a + 4b & 2a > b \\ a + 4b & 2a < b \\ 2a + 3b & 2a = b \end{cases} \\ &= \frac{1}{2} (3a + 2b) + \frac{1}{2} \begin{cases} 2a + 4b & 2a > b \\ a + 4b & 2a < b \\ 2a + 3b & 2a = b, \end{cases} \end{aligned}$$

where we used the intermediate results from Calculations 14 to 18.

Calculation 14. [The expected makespan of 

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \end{array} \right) = \frac{3a + 3b}{2}.$$

Calculation 15. [The expected makespan of $\begin{matrix} a \\ \otimes \\ \bullet \\ \bullet \end{matrix}$]

$$C_c \left(\begin{matrix} a \\ \otimes \\ \bullet \\ \bullet \end{matrix} \right) = \begin{cases} b - a + C_c \left(\begin{matrix} b - a \\ \otimes \\ \bullet \end{matrix} \right) & \mathbf{C: } a > b - a \iff 2a > b \\ \frac{1}{2} \left(a + C_c \left(\begin{matrix} 2a \\ \otimes \\ \bullet \end{matrix} \right) \right) + \frac{1}{2} \left(b - a + C_c \left(\begin{matrix} b - a \\ \otimes \\ \bullet \end{matrix} \right) \right) & \mathbf{D: } a < b - a \iff 2a < b \\ \frac{1}{2} \left(a + C_c \left(\begin{matrix} 2a \\ \otimes \\ \bullet \end{matrix} \right) \right) + \frac{1}{2} (a + C_c (\bullet)) & \mathbf{F: } a = b - a \iff 2a = b \end{cases}$$

$$= \begin{cases} b - a + \frac{1}{2}(4a + 2b) & 2a > b \\ \frac{1}{2} \cdot 2b + \frac{1}{2}(a + 2b) & 2a < b \\ \frac{1}{2} \cdot 2b + \frac{1}{2}(2a + b) & 2a = b \end{cases}$$

$$= \begin{cases} a + 2b & 2a > b \\ \frac{1}{2}(a + 4b) & 2a < b \\ \frac{1}{2}(2a + 3b) & 2a = b. \end{cases}$$

Calculation 16. [The expected makespan of $\begin{matrix} b - a \\ \otimes \\ \bullet \\ \bullet \end{matrix}$]


$$C_c \left(\begin{matrix} b - a \\ \otimes \\ \bullet \\ \bullet \end{matrix} \right) = \begin{cases} \frac{3}{2}(a + b) - (b - a) & b - a < a \\ \frac{2}{2}(a + b) + b - (b - a) & b - a \geq a. \end{cases}$$

Calculation 17. [The expected makespan of $\begin{matrix} 2a \\ \otimes \\ \bullet \\ \bullet \end{matrix}$]

$$C_c \left(\begin{matrix} 2a \\ \otimes \\ \bullet \\ \bullet \end{matrix} \right) = \frac{2}{2}(a + b) + b - 2a.$$

Calculation 18. [The expected makespan of $\begin{matrix} \bullet \\ \bullet \end{matrix}$]

$$C_c (\bullet) = \frac{2}{2}(a + b).$$

Calculation 19. [The expected makespan of 


$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \quad \bullet \\ \bullet \end{array} \right) &= \frac{1}{4} (a + C_c (\begin{array}{c} \bullet \\ \bullet \end{array})) + \frac{1}{2} \left(a + C_c \left(\begin{array}{c} a \\ \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{4} (b + C_c (\begin{array}{c} \bullet \\ \bullet \end{array})) \\
 &= \frac{1}{4} \left(a + \frac{1}{2}(2a + 3b) \right) + \frac{1}{2} \left(a + \begin{cases} 2b & 2a \neq b \\ 2a + b & 2a = b \end{cases} \right) + \frac{1}{4} \left(b + \frac{1}{2}(2a + 3b) \right) \\
 &= \frac{1}{8} (2a + 2a + 3b + 2b + 2a + 3b + 4a) + \frac{1}{2} \begin{cases} 2b & 2a \neq b \\ 2a + b & 2a = b \end{cases} \\
 &= \frac{1}{4} (5a + 4b) + \frac{1}{2} \begin{cases} 2b & 2a \neq b \\ 2a + b & 2a = b. \end{cases}
 \end{aligned}$$

The first intermediate result needed for Calculation 19 is the following.

Calculation 20. [The expected makespan of 

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \quad \bullet \\ \bullet \end{array} \right) &= \frac{1}{4} (a + C_c (\bullet)) + \frac{1}{2} \left(a + C_c \left(\begin{array}{c} a \\ \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{4} (b + C_c (\bullet)) \\
 &= \frac{1}{4} \left(a + \frac{a+b}{2} + 2a + a + b + 2b - 2a + b + \frac{a+b}{2} \right) \\
 &= \frac{1}{2} (2a + 3b)
 \end{aligned}$$

The other intermediate result needed in the first equality of Calculation 19 is given in the next calculation, where it itself has two more intermediate results, which are provided by Calculations 22 and 23.

Calculation 21. [The expected makespan of 

$$\begin{aligned}
 C_c \left(\begin{array}{c} a \\ \bullet \\ \bullet \end{array} \right) &= \begin{cases} b - a + C_c \left(\begin{array}{c} b-a \\ \bullet \\ \bullet \end{array} \right) & \mathbf{C}: a > b - a \iff 2a > b \\ \frac{1}{2} \left(a + C_c \left(\begin{array}{c} 2a \\ \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{2} \left(b - a + C_c \left(\begin{array}{c} b-a \\ \bullet \\ \bullet \end{array} \right) \right) & \mathbf{D}: a < b - a \iff 2a < b \\ \frac{1}{2} \left(a + C_c \left(\begin{array}{c} a \\ \bullet \\ \bullet \end{array} \right) \right) + \frac{1}{2} (a + C_c (\bullet)) & \mathbf{F}: a = b - a \iff 2a = b \end{cases} \\
 &= \begin{cases} 2b & 2a > b \\ \frac{1}{2} (2b + 2b) & 2a < b \\ \frac{1}{2} (2a + b + 2a + b) & 2a = b \end{cases} \\
 &= 2b.
 \end{aligned}$$

Calculation 22. [The expected makespan of $\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix}$]

$$\begin{aligned}
 C_c \left(\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix} \right) &= \begin{cases} b - a + C_c \left(\begin{matrix} b - a \\ \otimes \bullet \\ \bullet \end{matrix} \right) & \mathbf{C: } 2a > b \\
 \frac{1}{2} \left(a + C_c \left(\begin{matrix} 2a \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) + \frac{1}{2} \left(b - a + C_c \left(\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) & \mathbf{D: } 2a < b \\
 \frac{1}{2} \left(a + C_c \left(\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) + \frac{1}{2} (a + C_c(\bullet)) & \mathbf{F: } 2a = b \end{cases} \\
 &= \begin{cases} a + b & 2a > b \\
 \frac{1}{4}(-a + 3b) + \frac{1}{4}(-3a + 5b) & 2a < b \\
 \frac{1}{4}(a + 3b) + \frac{1}{4}(3a + b) & 2a = b \end{cases} \\
 &= \begin{cases} a + b & 2a \geq b \\
 2b - a & 2a < b \end{cases}
 \end{aligned}$$

Calculation 23. [The expected makespan of $\begin{matrix} b - a \\ \otimes \bullet \\ \bullet \end{matrix}$]

$$\begin{aligned}
 C_c \left(\begin{matrix} b - a \\ \otimes \bullet \\ \bullet \end{matrix} \right) &= \begin{cases} \frac{1}{2} \left(a - (b - a) + C_c \left(\begin{matrix} 2a - b \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) \\
 \quad + \frac{1}{4} \left(a + C_c \left(\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) + \frac{1}{4} (a + C_c(\bullet)) & \mathbf{E: } b - a < a \\
 \frac{1}{2} \left(a + C_c \left(\begin{matrix} a \\ \otimes \bullet \\ \bullet \end{matrix} \right) \right) + \frac{1}{2} (a + C_c(\bullet)) & \mathbf{F: } b - a \geq a \end{cases} \\
 &= \begin{cases} \frac{1}{2}(a + b) + \frac{1}{4} \left(\frac{a}{2} + \frac{3b}{2} \right) + \frac{1}{4} \left(\frac{3a}{2} + \frac{b}{2} \right) & 2a > b \\
 \frac{1}{2} \left(\frac{a}{2} + \frac{3b}{2} \right) + \frac{1}{2} \left(\frac{3a}{2} + \frac{b}{2} \right) & 2a \leq b \end{cases} \\
 &= a + b
 \end{aligned}$$

Putting all above calculations together, we obtain the following.

Calculation 24. Let $2a > b$, then

$$C_c \left(\begin{matrix} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \end{matrix} \right) = \frac{1}{4}(5a + 8b) < \frac{1}{4}(10a + 12b) = C_c \left(\begin{matrix} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \end{matrix} \right)$$

Let $2a < b$, then

$$C_c \left(\begin{matrix} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \end{matrix} \right) = \frac{1}{4}(5a + 8b) < \frac{1}{4}(8a + 12b) = C_c \left(\begin{matrix} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \end{matrix} \right).$$

Let $2a = b$, then

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) = \frac{1}{4}(9a + 6b) < \frac{1}{4}(10a + 10b) = C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right).$$

Thus,

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) < C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right).$$

F Induction Bases for Erlang Processing Times

The calculations in this chapter are for the induction bases in Conjectures 71 to 75 on pages 123 ff.

In the following, we only calculate the results for $k = 2$ and $k = 3$. Doing these calculations for an arbitrary k proved to be too demanding for **Mathematica**, which is why we only restricted ourselves to these two special cases. Furthermore, we consider Y as the distribution denoting the remaining processing time of an active task, whereas \bar{Y} denotes that task's total processing time as defined and used in cp. Chapter 25 on page 116.

First of all, we can simplify the expressions from Chapter 25 because we restrict ourselves only to $k = 2$ and $k = 3$. Moreover, we use $\lambda = 1$ as we have argued that it is a scaling factor, see (24.3).

$k = 2$ Since the probability density function and cumulative distribution function vanish for values less than 0, we only give them for $z \geq 0$ in the following. Then, we have

$$\begin{aligned} f_X(z) &= \frac{1^2 X^1}{1!} e^{-1z} = ze^{-z}, \\ F_X(z) &= 1 - \Gamma(2, z), \\ \mathbb{E}(X \wedge X) &= \frac{1}{2} \sum_{j=0}^1 \sum_{i=0}^1 \frac{1}{2^{i+j}} \binom{i+j}{i} = \frac{5}{4}, \end{aligned}$$

where we could also write $\Gamma(2, z) = (1+z)e^{-z}$. In addition, for a task that has been active for t time units, i.e., $\bar{Y} > t$, it holds that

$$\begin{aligned} f_Y(z) &= \frac{(z+t)e^{-(z+t)}}{\Gamma(2, t)}, \\ F_Y(z) &= 1 - \frac{\Gamma(2, z+t)}{\Gamma(2, t)}, \\ \mathbb{E}(Y) &= \int_0^{\infty} \frac{\Gamma(2, z+t)}{\Gamma(2, t)} dz \\ &= 1 + \frac{1}{1+t}. \end{aligned} \tag{F.1}$$

For the distribution of the minimum $X \wedge Y$, we obtain

$$\begin{aligned}
 f_{X \wedge Y}(z) &= \frac{e^{-z}}{\Gamma(2, t)} \left((z+t)\Gamma(2, z) e^{-t} + z\Gamma(2, z+t) \right), \\
 F_{X \wedge Y}(z) &= 1 - \frac{\Gamma(2, z)\Gamma(2, z+t)}{\Gamma(2, t)}, \\
 \mathbb{E}(X \wedge Y) &= \frac{1}{\Gamma(2, t)} \int_0^{\infty} \Gamma(2, z)\Gamma(2, z+t) dz \\
 &= \frac{5+3t}{4+4t}. \tag{F.2}
 \end{aligned}$$

The maximum $X \vee Y$ can also be calculated, and its expected value is

$$\begin{aligned}
 \mathbb{E}(X \vee Y) &= \int_0^{\infty} 1 - (1 - \Gamma(2, z)) \left(1 - \frac{\Gamma(2, z+t)}{\Gamma(2, t)} \right) dz \\
 &= \frac{11+9t}{4+4t}. \tag{F.3}
 \end{aligned}$$

k = 3 Again, we only give the formulas for $z \geq 0$. Hence,

$$\begin{aligned}
 f_X(z) &= \frac{z^2}{2} e^{-z}, \\
 F_X(z) &= 1 - \frac{\Gamma(3, z)}{2}, \\
 \mathbb{E}(X \wedge X) &= \frac{1}{2} \sum_{j=0}^2 \sum_{i=0}^2 \frac{1}{2^{i+j}} \binom{i+j}{i} = \frac{33}{16}.
 \end{aligned}$$

In presence of an active task, whose remaining processing time is denoted by Y and total processing time is denoted by \bar{Y} , with $\bar{Y} > t$, we have

$$\begin{aligned}
 f_Y(z) &= \frac{(z+t)^2 e^{-(z+t)}}{\Gamma(3, t)}, \\
 F_Y(z) &= 1 - \frac{\Gamma(3, z+t)}{\Gamma(3, t)}, \\
 \mathbb{E}(Y) &= \int_0^{\infty} \frac{\Gamma(3, z+t)}{\Gamma(3, t)} dz \tag{F.4}
 \end{aligned}$$

And, for $X \wedge Y$,

$$\begin{aligned}
 f_{X \wedge Y}(z) &= \frac{e^{-z}}{2\Gamma(3, t)} \left((z+t)^2 \Gamma(3, z) e^{-t} + z^2 \Gamma(3, z+t) \right), \\
 F_{X \wedge Y}(z) &= 1 - \frac{\Gamma(3, z)\Gamma(3, z+t)}{2\Gamma(3, t)}, \\
 \mathbb{E}(X \wedge Y) &= \frac{1}{2\Gamma(3, t)} \int_0^{\infty} \Gamma(3, z)\Gamma(3, z+t) dz. \tag{F.5}
 \end{aligned}$$

The expected value of the maximum is

$$\mathbb{E}(X \vee Y) = \int_0^{\infty} 1 - \left(1 - \frac{\Gamma(3, z)}{2}\right) \left(1 - \frac{\Gamma(3, z+t)}{\Gamma(3, t)}\right) dz. \quad (\text{F.6})$$

To retrace the steps taken above and verify the results, we have used **Mathematica** with the following expressions, see Listings F.1 and F.2.

```

1  fX[z_] = Simplify[PDF[ErlangDistribution[2,1], z], Assumptions->z>0]
2
3  Simplify[CDF[ErlangDistribution[2,1], z], Assumptions->z>0]
4  FX[z_] = 1-Gamma[2,z] (*This is identical to the line above.*)
5  expXminX = Expectation[Min[X,Y], {X \[Distributed] ErlangDistribution[2,1], Y
6  \[Distributed] ErlangDistribution[2,1]}]
7  expXminX == 1/2*Sum[Sum[2^(-i-j)*Binomial[i+j,i], {i,0,1}], {j,0,1}] (*check if
8  formula is true*)
9
10 Probability[Ybar <= z+t \[Conditioned] Ybar > t > 0, Ybar \[Distributed]
11 ErlangDistribution[2,1], Assumptions->t>0 && z>0]
12
13 FY[z_,t_] = 1-Gamma[2,z+t]/Gamma[2,t] (*This is identical to the line above.*)
14 fY[z_,t_] = Simplify[D[FY[z,t],z]]
15 expY[t_] = Integrate[1-FY[z,t], {z,0,Infinity}, Assumptions->t>0]
16
17 FXminY[z_,t_] = Simplify[1-(1-FX[z])*(1-FY[z,t]), Assumptions->z>0 && t>0]
18 fXminY[z_,t_] = Simplify[D[FXminY[z,t],z]]
19 fXminY[z,t] == Exp[-z]/Gamma[2,t]*((z+t)*Gamma[2,z]*Exp[-t]+z*Gamma[2,z+t]) (*check
20 if formula is true*)
21 expXminY[t_] = Integrate[1-FXminY[z,t], {z,0,Infinity}]
22
23 expXmaxY[t_] = Simplify[Integrate[1-FX[z]*FY[z,t], {z,0,Infinity}],
24 Assumptions->t>0]

```

Listing F.1: **Mathematica** expressions for the values from case $k = 2$ from above.

Calculations for the Induction Base of Conjecture 71

All the following integrals have been computed using **Mathematica**.


First, we consider the case $k = 2$.

```

1  fX[z_] = Simplify[PDF[ErlangDistribution[3,1], z], Assumptions->z>0]
2
3  Simplify[CDF[ErlangDistribution[3,1], z], Assumptions->z>0]
4  FX[z_] = 1-Gamma[3,z]/2 (*This is identical to the line above.*)
5  expXminX = Expectation[Min[X,Y], {X \[Distributed] ErlangDistribution[3,1], Y
   \[Distributed] ErlangDistribution[3,1]}]
6  expXminX == 1/2*Sum[Sum[2^(-i-j)*Binomial[i+j,i], {i,0,2}], {j,0,2}] (*check if
   formula is true*)
7
8  Probability[Ybar <= z+t \[Conditioned] Ybar > t > 0, Ybar \[Distributed]
   ErlangDistribution[3,1], Assumptions->t>0 && z>0]
9  FY[z_,t_] = 1-Gamma[3,z+t]/Gamma[3,t] (*This is identical to the line above.*)
10 fY[z_,t_] = Simplify[D[FY[z,t],z]]
11 expY[t_] = Integrate[1-FY[z,t], {z,0,Infinity}, Assumptions->t>0]
12
13 FXminY[z_,t_] = Simplify[1-(1-FX[z])*(1-FY[z,t]), Assumptions->z>0 && t>0]
14 fXminY[z_,t_] = Simplify[D[FXminY[z,t],z]]
15 fXminY[z,t] == Exp[-z]/(2*Gamma[3,t])*((z+t)^2*Gamma[3,z]*Exp[-t]+z^2*Gamma[3,z+t])
   (*check if formula is true*)
16 expXminY[t_] = Integrate[1-FXminY[z,t], {z,0,Infinity}]
17
18 expXmaxY[t_] = Simplify[Integrate[1-FX[z]*FY[z,t], {z,0,Infinity}],
   Assumptions->t>0]


```

Listing F.2: **Mathematica** expressions for the values from case $k = 3$ from above.

Calculation 25. [The expected makespan of ]

We have that $\mathbb{E}(X \wedge X) = \frac{5}{4}$. This yields

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \bullet \bullet \\ | \\ \bullet \end{array} \right) &= C_c(\bullet \bullet \bullet) + \underbrace{2k}_{=4} \\
 &= \mathbb{E}(X \wedge X) + C_c \left(\begin{array}{c} \bullet \\ \otimes \\ \bullet \end{array} \right) + 4 \\
 &= 4 + \frac{5}{4} + \mathbb{E}(X \vee Y) \quad \text{with the total processing time } \bar{Y} > \frac{5}{4} \\
 &\stackrel{(F.3)}{=} 4 + \frac{5}{4} + \frac{11 + 9 \cdot \frac{5}{4}}{4 + 4 \cdot \frac{5}{4}} \\
 &= \frac{139}{18} \\
 &\approx 7.7\bar{2}.
 \end{aligned}$$

Calculation 26. [The expected makespan of 

In order to calculate the second value we need the intermediate results

$$\begin{aligned} C_c \left(\begin{array}{c} 5/4 + z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) &= \mathbb{E}(Y) + \underbrace{2k}_{=4} \quad \text{with } \bar{Y} > \frac{5}{4} + z \\ &\stackrel{(F.1)}{=} 4 + 1 + \frac{1}{1 + \frac{5}{4} + z} \\ &= 5 + \frac{4}{9 + 4z}, \end{aligned}$$

$$\begin{aligned} C_c \left(\begin{array}{c} z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) &= \mathbb{E}(X \vee Y) + k \quad \text{with } \bar{Y} > z \\ &\stackrel{(F.3)}{=} 2 + \frac{11 + 9z}{4 + 4z}, \end{aligned}$$

and

$$\mathbb{E}(X \wedge Y) \stackrel{(F.2)}{=} \frac{5 + 3 \cdot \frac{5}{4}}{4 + 4 \cdot \frac{5}{4}} = \frac{35}{36}.$$

With these, we obtain

$$\begin{aligned} C_c \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) &= \mathbb{E}(X \wedge X) + C_c \left(\begin{array}{c} 5/4 \\ \otimes \\ \bullet \\ \bullet \end{array} \right) \\ &\stackrel{(25.4)}{=} \frac{5}{4} + \mathbb{E}(X \wedge Y) + \int_0^\infty F_X(z) f_{X \wedge Y}(z) C_c \left(\begin{array}{c} 5/4 + z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) dz \\ &\quad + \int_0^\infty F_Y(z) f_{X \wedge Y}(z) C_c \left(\begin{array}{c} z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) dz \quad \text{with } \bar{Y} > \frac{5}{4} \\ &= \frac{5}{4} + \frac{35}{36} + \frac{1}{\Gamma(2, \frac{5}{4})} \int_0^\infty (1 - \Gamma(2, z)) e^{-z} \\ &\quad \left(\left(z + \frac{5}{4} \right) \Gamma(2, z) e^{-5/4} + z \Gamma(2, z + \frac{5}{4}) \right) \left(5 + \frac{4}{9 + 4z} \right) dz \\ &\quad + \frac{1}{\Gamma(2, \frac{5}{4})} \int_0^\infty \left(1 - \frac{\Gamma(2, z + \frac{5}{4})}{\Gamma(2, \frac{5}{4})} \right) e^{-z} \\ &\quad \left(\left(z + \frac{5}{4} \right) \Gamma(2, z) e^{-5/4} + z \Gamma(2, z + \frac{5}{4}) \right) \left(2 + \frac{11 + 9z}{4 + 4z} \right) dz \\ &\approx \frac{5}{4} + \frac{35}{36} + \frac{0.852284}{\Gamma(2, \frac{5}{4})} + \frac{1.167}{\Gamma(2, \frac{5}{4})} \\ &\approx 5.35571. \end{aligned}$$

With Calculations 25 and 26 we have

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) > C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right)$$

for $k = 2$, showing the induction base for Conjecture 71 for this case. Listing F.3 shows the **Mathematica** expressions for Calculations 25 and 26. In order to execute these lines of code without errors, the code from Listing F.1 has to be executed first.


```

1 Intree1 = Simplify[4 + expXminX + expXmaxY[5/4]]
2
3 c1[z_] = 4 + expY[5/4+z]
4 c2[z_] = 2 + expXmaxY[z]
5 Intree2 = N[expXminX + expXminY[5/4] + Integrate[FX[z]*fXminY[z,5/4]*c1[z],
      {z,0,Infinity}] + Integrate[FY[z,5/4]*fXminY[z,5/4]*c2[z], {z,0,Infinity}], 10]
6
7 Intree1 > Intree2 (*Check validity of induction base.*)

```


Listing F.3: **Mathematica** expressions for the values from Calculations 25 and 26.

Next, we do the same calculations for $k = 3$.

Calculation 27. [The expected makespan of 

We have that $\mathbb{E}(X \wedge X) = \frac{33}{16}$. This yields

$$\begin{aligned}
 C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) &= C_c(\bullet \bullet \bullet) + \underbrace{2k}_{=6} \\
 &= \mathbb{E}(X \wedge X) + C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \bullet \\ \bullet \end{array} \right) + 6 \\
 &= 6 + \frac{33}{16} + \mathbb{E}(X \vee Y) \quad \text{with the total processing time } \bar{Y} > \frac{33}{16} \\
 &\stackrel{(F.6)}{=} 6 + \frac{33}{16} + \int_0^{\infty} 1 - \left(1 - \frac{\Gamma(3, z)}{2} \right) \left(1 - \frac{\Gamma(3, z + \frac{33}{16})}{\Gamma(3, \frac{33}{16})} \right) dz \\
 &= \frac{487539}{42512} \\
 &\approx 11.46826778.
 \end{aligned}$$

Calculation 28. [The expected makespan of ]

First, we derive the intermediate results that we need later on.

$$\begin{aligned}
 C_c \left(\begin{array}{c} 33/16 + z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) &= \mathbb{E}(Y) + \underbrace{2k}_{=6} \quad \text{with } \bar{Y} > \frac{33}{16} \\
 &\stackrel{(F.4)}{=} 6 + \int_0^{\infty} \frac{\Gamma(3, u + \frac{33}{16} + z)}{\Gamma(3, \frac{33}{16} + z)} du \\
 &= 7 + \frac{32(65 + 16z)}{2657 + 32z(49 + 8z)},
 \end{aligned}$$

$$\begin{aligned}
 C_c \left(\begin{array}{c} z \\ \otimes \\ \bullet \\ \bullet \end{array} \right) &= \mathbb{E}(X \vee Y) + k \quad \text{with } \bar{Y} > z \\
 &\stackrel{(F.6)}{=} 3 + \int_0^{\infty} 1 - \left(1 - \frac{\Gamma(3, u)}{2} \right) \left(1 - \frac{\Gamma(3, u + z)}{\Gamma(3, z)} \right) du \\
 &= 3 + \frac{63 + 5z(11 + 5z)}{8e^z \Gamma(3, z)},
 \end{aligned}$$

and

$$\begin{aligned}
 \mathbb{E}(X \wedge Y) &\stackrel{(F.5)}{=} \frac{1}{2\Gamma(3, \frac{33}{16})} \int_0^{\infty} \Gamma(3, z) \Gamma(3, z + \frac{33}{16}) dz \quad \text{with } \bar{Y} > \frac{33}{16} \\
 &= \frac{29271 e^{-33/16}}{2048 \Gamma(3, \frac{33}{16})} \\
 &\approx 1.377070004.
 \end{aligned}$$

Using these, we get

$$\begin{aligned}
C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) &= \mathbb{E}(X \wedge X) + C_c \left(\begin{array}{c} 33/16 \\ \bullet \\ \bullet \bullet \end{array} \right) \\
&\stackrel{(25.4)}{=} \frac{33}{16} + \mathbb{E}(X \wedge Y) + \int_0^\infty F_X(z) f_{X \wedge Y}(z) C_c \left(\begin{array}{c} 33/16 + z \\ \bullet \\ \bullet \bullet \end{array} \right) dz \\
&\quad + \int_0^\infty F_Y(z) f_{X \wedge Y}(z) C_c \left(\begin{array}{c} z \\ \bullet \\ \bullet \bullet \end{array} \right) dz \\
&= \frac{33}{16} + \mathbb{E}(X \wedge Y) + \int_0^\infty \left(1 - \frac{\Gamma(3, z)}{2} \right) \frac{e^{-z}}{2\Gamma(3, \frac{33}{16})} \cdot \\
&\quad \left(\left(z + \frac{33}{16} \right)^2 \Gamma(3, z) e^{-33/16} + z^2 \Gamma(3, z + \frac{33}{16}) \right) \cdot \\
&\quad \left(7 + \frac{32(65 + 16z)}{2657 + 32z(49 + 8z)} \right) dz \\
&\quad + \int_0^\infty \left(1 - \frac{\Gamma(3, z + \frac{33}{16})}{\Gamma(3, \frac{33}{16})} \right) \frac{e^{-z}}{2\Gamma(3, \frac{33}{16})} \cdot \\
&\quad \left(\left(z + \frac{33}{16} \right)^2 \Gamma(3, z) e^{-33/16} + z^2 \Gamma(3, z + \frac{33}{16}) \right) \\
&\quad \left(3 + \frac{63 + 5z(11 + 5z)}{8e^z \Gamma(3, z)} \right) dz \\
&\approx \frac{33}{16} + 1.377070004 + \frac{3.81676}{2\Gamma(3, \frac{33}{16})} + \frac{7.51814}{2\Gamma(3, \frac{33}{16})} \\
&\approx 7.73462.
\end{aligned}$$

Comparing the makespans from Calculations 27 and 28 gives us the induction base of Conjecture 71 for $k = 3$:

$$C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right) > C_c \left(\begin{array}{c} \bullet \\ \bullet \bullet \\ \bullet \bullet \bullet \end{array} \right).$$

Again, we did the calculations with **Mathematica**, too, see Listing F.4. To be executed without error, the code from Listing F.2 has to be executed first.

However, a value of less than 9 is not to be expected. We can easily calculate that the optimal expected makespan of a chain of three tasks is 9. The intree used for the induction base case is a chain of three tasks with two additional sources added to it whose optimal expected makespan cannot be smaller than the one of the chain itself, obviously. This leaves us to believe that even a seemingly simple thing such as calculating the expected makespan of a small example, even for fixed k and λ , is a lot more complex than our approaches before. Unfortunately, we did

```

1 Intree1 = Simplify[6 + expXminX + expXmaxY[33/16]]
2
3 c1[z_] = 6 + expY[33/16+z]
4 c2[z_] = 3 + expXmaxY[z]
5 Intree2 = N[expXminX + expXminY[33/16] + Integrate[FX[z]*fXminY[z,33/16]*c1[z],
      {z,0,Infinity}] + Integrate[FY[z,33/16]*fXminY[z,33/16]*c2[z], {z,0,Infinity}],
      10]
6
7 Intree1 > Intree2 (*Check validity of induction base.*)

```

Listing F.4: **Mathematica** expressions for the values from Calculations 27 and 28.

not manage to come up with a different approach on how to calculate the expected makespan and hope that future work will succeed in fixing the missing parts of the proof ideas for our conjectures in Conjecture 71.

Because of this conundrum, we did not bother to calculate the base cases for Conjectures 72 to 75 as the approach does not seem to work here anyway.

Bibliography

- [1] *The on-line encyclopedia of integer sequences*. published electronically at <http://oeis.org/A000108>[51]. Catalan numbers.
- [2] *The on-line encyclopedia of integer sequences*. published electronically at <http://oeis.org/A001850>[51]. Central Delannoy numbers.
- [3] M. ABRAMOWITZ, *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*, Dover Publications, Incorporated, 1974.
- [4] M. ABRAMOWITZ, I. STEGUN, AND N. SFETCU, *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*, Nicolae Sfetcu, 2014.
- [5] T. L. ADAM, K. M. CHANDY, AND J. R. DICKSON, *A comparison of list schedules for parallel processing systems*, Commun. ACM, 17 (1974), pp. 685–690.
- [6] A. V. AHO, M. R. GAREY, AND J. D. ULLMAN, *The transitive reduction of a directed graph*, SIAM Journal on Computing, 1 (1972), pp. 131–137.
- [7] A. V. AHO AND J. E. HOPCROFT, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 1974.
- [8] P. BRUCKER, *Scheduling Algorithms*, Springer Berlin Heidelberg, fifth ed., 2007.
- [9] J. BRUNO, *On scheduling tasks with exponential service times and in-tree precedence constraints*, Acta Informatica, 22 (1985), pp. 139–148.
- [10] C. CAMINO, *Stochastic scheduling with two processors and arbitrary precedence relations*, Master’s Thesis, Technische Universität München, July 2015.
- [11] E. J. CARTAN, *Œuvres complètes. Partie II. , [Algèbre, systèmes différentiels et problèmes d’équivalence]*, ed. du CNRS, Paris, 1984. Cette édition est la reproduction photomécanique de l’édition Gauthiers-Villars parue en 1953.
- [12] K. M. CHANDY AND P. F. REYNOLDS, *Scheduling partially ordered tasks with probabilistic execution times*, in Proceedings of the Fifth ACM Symposium on Operating Systems Principles, SOSP '75, New York, NY, USA, 1975, ACM, pp. 169–177.
- [13] W. CLARK, W. N. POLAKOV, AND F. W. TRABOLD, *The Gantt Chart: a Working Tool of Management*, Ronald Press, 1923.
- [14] E. G. COFFMAN AND R. L. GRAHAM, *Optimal scheduling for two-processor systems*, Acta Informatica, 1 (1972), pp. 200–213.
- [15] L. COMTET, *Advanced Combinatorics: The Art of Finite and Infinite Expansions*, Springer Netherlands, 1974.

- [16] S. A. COOK, *The complexity of theorem-proving procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, New York, NY, USA, 1971, ACM, pp. 151–158.
- [17] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction To Algorithms*, MIT Press, 2001.
- [18] J. DU AND J. T. Y. LEUNG, *Scheduling tree-structured tasks with restricted execution times*, Inf. Process. Lett., 28 (1988), pp. 183–188.
- [19] R. J. DUFFIN, *Topology of series-parallel networks*, Journal of Mathematical Analysis and Applications, 10 (1965), pp. 303–318.
- [20] M. DZIEMIAŃCZUK, *Counting lattice paths with four types of steps*, Graphs and Combinatorics, 30 (2013), pp. 1427–1452.
- [21] M. DZIEMIAŃCZUK, *Generalizing delannoy numbers via counting weighted lattice paths*, in Integers: Annual, vol. 2013, De Gruyter, 2013, pp. 764–796.
- [22] M. ELASHMAWI, *Stochastic scheduling with uniformly distributed processing times and in-tree precedence constraints*, Bachelor's Thesis, Technische Universität München, November 2015.
- [23] W. FELLER, *An Introduction to Probability Theory and its Applications*, Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics, Wiley, 1971.
- [24] ———, *An Introduction to Probability Theory and its Applications, Volume 2*, Wiley India Pvt. Limited, 2008.
- [25] O. FORSTER, *Analysis 1: Differential- und Integralrechnung einer Veränderlichen*, vieweg studium; Grundkurs Mathematik, Vieweg+Teubner Verlag, 2013.
- [26] M. R. GAREY AND D. S. JOHNSON, *“Strong” NP-Completeness results: Motivation, examples, and implications*, J. ACM, 25 (1978), pp. 499–508.
- [27] H. W. GOULD, *Table for combinatorial numbers and associated identities: Table 2*. published electronically at <http://www.math.wvu.edu/~gould/Vol.8.PDF>, 2010.
- [28] ———, *Table for fundamentals of series: Part I: Basic properties of series and products*. published electronically at <http://www.math.wvu.edu/~gould/Vol.1.1.PDF>, 2011.
- [29] R. L. GRAHAM, D. E. KNUTH, AND O. PATASHNIK, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd ed., 1994.

- [30] 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 Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, P. L. Hammer, E. L. Johnson, and B. H. Korte, eds., vol. 5 of Annals of Discrete Mathematics, Elsevier, 1979, pp. 287–326.
- [31] T. C. HU, *Parallel sequencing and assembly line problems*, Operations Research, 9 (1961), pp. 841–848.
- [32] N. JOHNSON, A. KEMP, AND S. KOTZ, *Univariate Discrete Distributions*, Wiley Series in Probability and Statistics, Wiley, 2005.
- [33] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, eds., Boston, MA, 1972, Springer US, pp. 85–103.
- [34] D. E. KNUTH, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.
- [35] M. R. KROM, *The decision problem for a class of first-order formulas in which all disjunctions are binary*, Mathematical Logic Quarterly, 13 (1967), pp. 15–20.
- [36] E. L. LAWLER, *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, Annals of Discrete Mathematics, 2 (1978), pp. 75–90.
- [37] E. L. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS, *Sequencing and scheduling: Algorithms and complexity*, Handbooks in Operations Research and Management Science, 4 (1993), pp. 445–522.
- [38] J. K. LENSTRA AND A. H. G. RINNOOY KAN, *Complexity of scheduling under precedence constraints*, Oper. Res., 26 (1978), pp. 22–35.
- [39] J. K. LENSTRA, A. H. G. RINNOOY KAN, AND P. BRUCKER, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics, 1 (1977), pp. 343–362.
- [40] L. A. LEVIN, *Universal sequential search problems*, Problemy Peredachi Informatsii, 9 (1973), pp. 115–116.
- [41] M. MAASS, *Scheduling independent and identically distributed tasks with in-tree constraints on three machines in parallel*, Diplomarbeit, Technische Universität München, October 2001.

- [42] E. MOKOTOFF, *Parallel machine scheduling problems: a survey*, Asia-Pacific Journal of Operational Research, 18 (2001), pp. 193–242.
- [43] P. MÜLLER, *Investigation of stochastic scheduling problems*, Master's Thesis, Technische Universität München, November 2013.
- [44] K. NAKAJIMA, J. T. Y. LEUNG, AND S. L. HAKIMI, *Optimal two processor scheduling of tree precedence constrained tasks with two execution times*, Performance Evaluation, 1 (1981), pp. 320–330.
- [45] C. H. PAPADIMITRIOU AND J. N. TSITSIKLIS, *On stochastic scheduling with in-tree precedence constraints*, SIAM Journal on Computing, 16 (1987), pp. 1–6.
- [46] M. PETKOVSEK, H. S. WILF, AND D. ZEILBERGER, *A = B*, Ak Peters Series, Taylor & Francis, 1996.
- [47] M. L. PINEDO, *Scheduling: Theory, Algorithms, and Systems*, Springer Publishing Company, Incorporated, 4th ed., 2011.
- [48] X. S. RAYMOND, *Elementary Introduction to the Theory of Pseudodifferential Operators*, Studies in Advanced Mathematics, Taylor & Francis, 1991.
- [49] P. F. REYNOLDS AND K. M. CHANDY, *Scheduling partially ordered tasks with exponentially distributed times*, tech. rep., Austin, TX, USA, 1979.
- [50] T. SCHICKINGER AND A. STEGER, *Diskrete Strukturen 2: Wahrscheinlichkeitstheorie und Statistik*, Springer-Lehrbuch, Springer Berlin Heidelberg, 2013.
- [51] N. J. A. SLOANE, *The on-line encyclopedia of integer sequences*. published electronically at <http://oeis.org>, 2016.
- [52] R. P. STANLEY, *Enumerative Combinatorics*, Cambridge Studies in Advanced Mathematics, Wadsworth Publ. Co., Belmont, CA, USA, 2011.
- [53] R. P. STANLEY AND S. FOMIN, *Enumerative Combinatorics, Volume 2*, Cambridge Studies in Advanced Mathematics, Cambridge University Press, Cambridge, New York, 1999. Errata et addenda: pp. 583-585.
- [54] W. SZPANKOWSKI, *Average Case Analysis of Algorithms on Sequences*, Wiley Series in Discrete Mathematics and Optimization, Wiley, 2001.
- [55] J. D. ULLMAN, *NP-complete scheduling problems*, Journal of Computer and System Sciences, 10 (1975), pp. 384–393.
- [56] ———, *Complexity of sequencing problems*, Computer and Job-Shop Scheduling Theory, (1976), pp. 139–164.

Index

- antiderivative, 24
- binomial coefficient, 20, 58
- configuration, 14, 37, 123
- configuration graph, 48, 58, 78, 132, 137
- decision point, 7, 14
- derivative, 24
- Erlang distribution, 115, 192
 - minimum, 117
- expected makespan, 11, 15, 31, 37, 44, 92, 105, 119, 154, 180
 - continuous, 32
 - discrete, 31
- exponential distribution, 35, 38, 45, 131, 180
 - minimum, 36
- flatness, 40, 122
- Gamma function, 116
- Gantt chart, 8, 132
- generating function, 21, 60, 71
- geometric distribution, 89, 181
 - minimum, 90
- Hasse diagram, 9, 48, 52, 55
- HLF*, 13, 38, 94, 111, 122, 137, 163
 - level-oriented, 14
- hypergeometric function, 23, 60
- integral, 24
- intree decomposition, 82
- makespan, 10
- memorylessness, 35, 89
- multinomial coefficients, 66
- optimal strategy, 11, 39, 94
- optimality, 11, 13
- optimality criterion, 11
- precedences, 9
 - chains, 57, 180
 - dag*, 9, 131
 - inforest, 9
 - intree, 9, 39, 45, 163
 - Psi-graph, 81
 - Y-graph, 80, 168
- probability, 25, 47, 58, 84, 91
 - conditional, 26, 100
 - independent, 25
- probability distribution, 25
 - continuous, 26
 - discrete, 26
- profile, 15, 39, 51
 - join level, *see also* intree decomposition
- random variable, 26
 - continuous, 26
 - discrete, 26
 - expected value, 27
- schedule, 8
- scheduling notation, 12
- scheduling strategy, 7
 - dynamic, 139
 - static, 139
- series, 19
 - convolution, 22, 63
 - geometric, 19, 23, 63
 - hypergeometric, 23
 - power series, 21, 60
- stair function, 73, 85
- sum, 19
 - geometric, 19
- task, 7
 - active, 7, 15

- ancestor, 9, 140
- available, *see also* source
- descendant, 9, 141, 143
- expected processing time, 11
- level, 13, 39, 45, 81, 139, 143
- predecessor, 9
- processing time, 7
- source, 15
- successor, 9

uniform distribution, 99

- continuous, 102
- discrete, 99, 188
- minimum, 99

