POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Approximation Algorithms for Uniform Parallel Machine Scheduling through Mathematical Programming

Relatori
prof. Federico Della Croce di Dojola
prof. Rosario Scatamacchia
firma dei relatori

. . . . . .
. . . . . .

Candidato
Luca Savant Aira

firma del candidato

. . . . . .

Anno Accademico 2021-2022

Summary

Making active and context-aware decisions is a fundamental part of human intellect. Operations Research is the engineered version of this process.

An Operations Research problem is usually translated into the mathematical language of Optimization, where one is requested to maximize a given *objective*, expressed as a function of *decision variables*, while satisfying given *constraints*. If the decision variables are continuous and both the objective and the constraints are convex functions, the optimization problem is called a *convex optimization problem* and there are efficient algorithms to find the optimal solution. When some decision variables are discrete or the convexity requirement is not satisfied, the problem becomes much more difficult to be solved to optimality. Therefore, one settles for a suboptimal solution that can be found in a reasonable amount of time and computing resources, with case-specific algorithms, called *heuristics*.

This work studies one of the most important theoretical questions in Operations Research: to find the *approximation ratio* ρ, if it exists, of a given heuristic algorithm. In other words, one is interested in certifying that the ratio between the proposed algorithm solution value and the optimal solution value is bounded in a minimization (maximization) problem above (below) by ρ. Such algorithm has then a ρ-approximation ratio.

In this work, we develop new approximation algorithms for the scheduling problem of minimizing the maximum completion time on a set of uniform parallel machines. In the literature, there are examples of such algorithms. However, the proofs are often complex and difficult to be generalized, as they rely on proofs handling many exhaustive subcases or requiring algorithm-specific properties. We try, instead, to delineate a methodology that can be applied and possibly generalized to other algorithms and problems.

The methodology works as follows. First we prove a very general theorem that can be applied to all proposed algorithms and then, leveraging the power of commercial Mixed-Integer Non-Linear Program (MINLP) solvers, we solve an optimization problem for each proposed algorithm. Joining the analytical proofs with the optimality results, we formally prove the approximation ratios.

Acknowledgements

Il lavoro che state leggendo è il frutto del mio percorso universitario, durato 5 anni e che ha caratterizzato e caratterizzerà gran parte della mia vita. Durante questo percorso molte persone mi sono state vicine.

Vorrei ringraziare i miei genitori che da sempre mi consigliano e sostengono in qualsiasi decisione io prenda.

Vorrei ringraziare i miei nonni per tutti gli "imbocca al lupo" e preghiere che mi hanno sempre infuso coraggio e determinazione. Un grazie anche ai miei cugini, che considero fratelli, e a zia Pippi per essermi stati vicini ogni weekend.

Grazie a Vale che mi ha sopportato fin da prima che diventassi un ingegnere, e soprattutto durante il percorso per diventarlo.

Un ringraziamento anche ai miei amici e colleghi Ferox, Gabbo, i tre Matteo e Tib per aver condiviso le gioie e i dolori del Poli. In particolare vorrei ringraziare Marco, carissimo amico, per essere sempre stato un eccellente compagno di discussioni matematiche e non.

Non per ultimi, vorrei ringraziare i miei relatori e professori Federico Della Croce e Rosario Scatamacchia per avermi introdotto nel mondo della ricerca e per essere stati sempre diponibili a confrontarsi con me.

Infine, vorrei ringraziare i gruppi HPC@Polito e Ladispe@Polito per aver messo a disposizione le risorse computazionali usate in questo lavoro.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We consider the problem of scheduling a set of jobs on a set of uniform parallel machines with the aim of minimizing the maximum completion time, and we will denote it as $QM||C_{max}$.

Given a list of J jobs (also called tasks or processes), each characterized by a length p_j (also called duration), and a list of M machines (also called workers or processors), each characterized by a speed factor q_m, how do we assign the jobs to the machines (called scheduling) in a way that minimizes the maximum completion time?

This problem is an $\mathcal{NP}$-hard optimization problem. In other words, since one basically should evaluate all possible M^J assignments, optimally solving the problem is too time-consuming if the number of machines and jobs is big enough.

Hence, the need to develop algorithms that find suboptimal schedules in a reasonable time. Moreover, one could be interested in proving that the value of such a suboptimal schedule is at most ρ times worse than the optimal one. If it can be proven that ρ is always finite, even in the worst possible combination of jobs lengths and machines speeds, then the used algorithm is called a ρ-approximation algorithm. The Longest Processing Time (LPT) algorithm is an example of an approximation algorithm, as we will show later.

In this work, we develop new approximation algorithms for the uniform machine scheduling problem. In particular, we find valid proofs for approximation ratios by mathematical optimization and commercial solvers.

1.1 What is an optimization problem?

How can we design an efficient electricity grid? Which is the optimal price for this product? What is the shortest route from point A to point B? What are the solutions to this Sudoku puzzle? Which of these options is the best one according to this criterion?

All the questions above can be mathematically formulated as optimization problems. An optimization problem consists of an objective function, a domain, and several variables. The objective function is the mathematical formulation of the criterion to be optimized (maximized or minimized), for example, a measure of the efficiency of an electricity grid or the number of errors in a Sudoku puzzle. The domain is the mathematical formulation of the constraints that must be satisfied, for example, the start and end points of a route or the already placed clues in a Sudoku puzzle. The variables are the mathematical objects used to encode a possible solution, for example, the yet unknown numbers in a Sudoku puzzle. Let f be the objective function, Ω be the domain, and x be the vector variable of the problem. The problem can then be mathematically modeled as:

$$\max_{x \in \Omega} f(x)$$

Three clarifications are necessary. Firstly, note that by applying the operators max and min, we get a value $f^* = f(x^*)$ for a point x^* in the domain and not the point x^* itself. If one is interested in the optimal variable x^* itself, one should use the operators argmax and argmin. These operators describe the subset of the domain formed by all the points in the domain where f is optimized. From now on, as the computational implementations of max and argmax operators return both the optimum and the point of optimum, we will use the two forms interchangeably. Secondly, note that we can use the maximization form without loss of generality, as it holds that $-\max_{x \in \Omega} f(x) = \min_{x \in \Omega} -f(x)$. Thirdly, with an abuse of notation, in this work we will use the operator max even when the operator sup should be used.

Now that the general structure of an optimization problem is defined, we focus on how to solve an optimization problem.

If Ω is a finite set with a sufficiently low number of elements, the optimization problem can be solved simply by evaluating the objective function f over each element of Ω and selecting the ones that maximize f. Unfortunately, Ω is often an infinite subset of $\mathbb{R}^n$ that satisfies multiple constraints, for example $\Omega = \left\{ x \in \mathbb{R}^n : \vec{g}(x) \leq \vec{0} \right\}$ where $g : \mathbb{R}^n \to \mathbb{R}^m$ is a known function. In this case, a plethora of optimization problems families emerges, such as linear problems, quadratic problems, convex problems, concave problems, integer problems, and combinatorial problems, ...

The most important properties for categorizing an optimization problem are:

- **Continuity**: If all the variables are continuous, the problem is said to be continuous. On the other hand, if some variables are discrete, then the problem is called a mixed-integer problem.
- **Convexity**: If both the objective function and the domain are convex, then the problem can be solved to optimality by a gradient descent procedure.
  - If both the objective function and the domain are described by linear functions (hence they are also convex) the gradient descent procedure can be discarded

for more efficient algorithms, both in practice and in theory. Refer to Tadei and Della Croce [2010] for an in-depth explanation of the solution process of a linear problem with the *Simplex* algorithm.

In this work, we mostly deal with Mixed-Integer Non-Linear Problems (MINLP). These problems are characterized by the presence of both continuous and binary variables (discrete variables that can assume only the values 0 or 1) and also by the presence of non-linear constraints. In particular, all the non-linearizable non-linearities are non-positive definite quadratic non-linearities, i.e., some constraints contain products of two continuous variables.

1.2 What is a machine scheduling problem?

A machine scheduling problem is an optimization problem whose input are a set of jobs and a set of machines. The expected output is a schedule, which is the assignment of jobs to machines. Details of the problem, such as the objective function, scheduling constraints, and job characteristics, can be specified using a standard three-field notation that can be found in Graham et al. [1979]. The fields describe respectively the machine environment, the job characteristics and constraints, and the objective function.

- Examples for the first field are P, Q, and R. P means that the machines are parallel and identical. Q means that the machines are parallel machines, but each one is characterized by a speed factor. For example, if machine A is twice as fast as machine B, then each job on machine A will be completed twice as fast as the same job on machine B. R means that the machines are parallel but unrelated, i.e., each job has an unrelated duration on each machine. These letters (P, Q, R) can be followed by the number of available machines. If the number of machines is present, for example, $P2$ or $Q3$, then it is a fixed parameter.
- Examples for the second field are $prec$, d_j or fix_j. $prec$ means that a given precedence relation must be respected while scheduling the jobs, i.e., job A must be finished before starting job B (as in a production chain). d_j represents the due date of each job, i.e., job A must be finished before a given time otherwise a penalty must be paid. fix_j means that each job has a known subset of machines and needs all of them for its execution.
- Examples for the third field are C_{max}, C_{mean} or F_{mean}. C_{max}, also called *makespan*, is the maximum completion time. In other words, it is the time at which the last running job has finished. C_{mean} is the mean completion time, i.e., the mean of all machines completion times. F_{mean} is the mean flow time, i.e., the mean difference between completion times and release times (the earliest time when a job can start execution).

In this work, we will focus on the $QM||C_{max}$ problem. There are M machines, each characterized by a speed factor q_m. Each job is characterized only by its length p_j. The schedule must minimize the makespan, i.e., when the last running job on the last running

machine has finished. Note that the second field is empty, hence no further jobs constraints are present.

1.3 What is a computationally impractical problem?

Some scheduling problems, for example, $P2|fix_j|C_{max}$ or $QM||F_{mean}$, can be efficiently solved to optimality using specific algorithms, as done in Hoogeveen et al. [1994], Horowitz and Sahni [1976]. Other problems are *computationally impractical*, hence they cannot be solved efficiently. Here, *computationally impractical* means that, if the input is big enough, the expected running time quickly becomes comparable with the life of the universe (see Table 1.1).

This is closely linked to the very famous $\mathcal{P}$-vs-$\mathcal{NP}$ problem, a fundamental question in theoretical computer science and one of the Millennium Problems[1]. Loosely speaking, this question asks if any problem, whose solution can be efficiently checked, admits an algorithm that can solve it efficiently or if there are problems that are intrinsically hard to be solved.

If an algorithm has an execution time that can be bound with a function f of the size n of its input, it is common to say that the algorithm has a complexity of $\mathcal{O}(f(n))$. Table 1.1 shows some algorithm complexities and examples of their expected running times.

Algorithm complexity	n=10	n=20	n=30	...	n=100
$f(n) \in \mathcal{O}(n)$	0.1 sec	0.2 sec	0.3 sec	...	1.0 sec
$f(n) \in \mathcal{O}(2^n)$	0.0001 sec	1.0 sec	17.1 min	...	3.8×10^{16} years

Table 1.1. Algorithm Complexity with some examples of expected running times

To clarify the concept of running time complexity, let us consider as an example a list of n real numbers that must be sorted in ascending order.

Firstly, it is easy to check that a solution to this problem, i.e., a candidate sorting, can be checked in linear time $\mathcal{O}(n)$. It suffices to iterate over each element once and check that the current element is smaller than the next one. Hence, the number of operations necessaries to check a candidate solution is $n - 1 = \mathcal{O}(n)$.

One very inefficient algorithm to sort the list is to iteratively check each permutation until a sorted one is found. As the number of permutations of n objects is $n!$, and checking if a list of n objects is sorted requires $n - 1$ comparisons, the total number of operations needed to sort the list in this way is $\mathcal{O}(n! \cdot n)$. As the number of operations required by an algorithm is proportional to its running time, this algorithm is said to have a non-polynomial run-time complexity. Surely, a human tasked with sorting a list will never try

[1] https://www.claymath.org/millennium-problems

each permutation and check if it is sorted.

A more natural approach is the *Insertion Sort* algorithm. This algorithm iteratively takes an item from the input list and puts it in the output list in the correct position, keeping the output list sorted. This algorithm has a run-time complexity of $\mathcal{O}(n^2)$, hence a polynomial one.

Note that there are more efficient algorithms, for example the *Heapsort* algorithm, that have a run-time complexity of $\mathcal{O}(n \log n)$. As it can be proved that a lower complexity cannot be reached, it is guaranteed that the problem of sorting a list has a complexity of $\mathcal{O}(n \log n)$. Refer to Crescenzi et al. [2012] for a deeper explanation.

Now that we have matured an intuition about this subject, we can give the following definitions:

- A **Decision Problem** is a problem whose solution is binary: yes or no. Note that any optimization problem can be translated into a series of decision problems. For example, *"How long is the shortest path from A to B?"* may become *"Does it exist a path from A to B shorter than* 3 *units?"*. Also note that sorting a list is not a decision problem, but one could rephrase it as *"Is list α the sorted version of list β?"*. Formally, there is a very elegant definition for a decision problem. Let us represent an input to a decision problem as a string of symbols, for example a binary string. Then we identify the decision problem itself as the possibly infinite set of binary strings $\{0,1\}^*$ whose answer is "yes". In this representation, a decision problem is simply the set of binary strings that, interpreted as input, satisfy a certain property. From now on, x will refer to an (encoded) input, n will refer to the length of x and $\Pi \subseteq \{0,1\}^*$ will refer to a decision problem.
- An **Algorithm** $\mathcal{A}$ is, without being too technical, a computational representation of a function from a binary string x to another binary string y. Given a decision problem Π, $\mathcal{A}$ solves Π if $\mathcal{A}^{-1}(1) = \Pi$, i.e. the preimage of $y = 1$ through the algorithm $\mathcal{A}$ is Π. The computational aspect is very important: an algorithm follows a finite sequence of operations, one at a time, to transform the string x to the string y.
- The **Complexity** of an algorithm $\mathcal{A}$ is $f(n)$, where $f(n)$ is the maximum total number of operations needed to transform the input string x to the output string y. Note that the complexity is a function of the length of the input string x, not of the string x itself.
- A decision problem Π belongs to the **class** $\mathcal{P}$ if it exists at least one algorithm $\mathcal{A}$, whose complexity $f(n)$ is polynomial, that solves it. In other words, problems in $\mathcal{P}$ can be solved in polynomial time.
- A decision problem Π belongs to the **class** $\mathcal{NP}$ if it exists at least one algorithm $\mathcal{V}$ with polynomial complexity and a polynomial p such that:
  - $x \in \Pi \implies \exists y \in \{0,1\}^{p(n)} : \mathcal{V}(x, y) = 1$
  - $x \notin \Pi \implies \forall y \in \{0,1\}^{p(n)} : \mathcal{V}(x, y) = 0$

The algorithm $\mathcal{V}$ is called verifier, and y is called **yes-certificate** as it should represent a proof that $x \in \Pi$. The first condition states that the verifier must accept all valid proofs. The second condition states that the verifier must reject all invalid proofs. Nothing is said about how we found a valid yes-certificate y for a given input x. From a theoretical point of view, we can brute-force every possible binary string y and test if it is a valid yes-certificate. However, the number of possible y strings is exponential in n: $2^{p(n)}$. In other words, problems in $\mathcal{NP}$ can be checked in polynomial time.

Going back to our machine scheduling problem $QM||C_{max}$, we can show that its decision version is in $\mathcal{NP}$. In fact, also the $P2||C_{max}$ problem is in $\mathcal{NP}$, as shown in Lawler et al. [1993]. Note that an instance of $QM||C_{max}$ can be reduced to an instance of $P2||C_{max}$ by simply fixing the number of machines to 2 and fixing the speed of the machines to 1. Hence, the decision version of $QM||C_{max}$ is at least as difficult as the $P2||C_{max}$ counterpart. So, the decision version of $QM||C_{max}$ is in $\mathcal{NP}$. Note that this means that also the optimization version of $QM||C_{max}$ is difficult to solve, otherwise it should suffice to solve the optimization version to solve the decision version.

From now on, with a slight abuse of notation, we state that an optimization problem is in $\mathcal{NP}$ if its decision version is in $\mathcal{NP}$.

1.4 What is an approximation algorithm?

If an $\mathcal{NP}$ optimization problem is sufficiently large, then finding the optimal solution is *computationally impractical* and, even if supplied, it is *computationally impractical* to prove its optimality. We are thus forced to drop the requirements of certified optimality and settle for a humbler target: finding efficiently suboptimal solutions.

There are two main kinds of algorithms developed for this goal: heuristics and approximation algorithms. This duality reflects the one introduced at the beginning of this chapter: finding a solution vs proving its optimality.

Heuristics are algorithms created to be effective in real-life scenarios. It means that, while no solid theoretical foundations are present, solutions reached by heuristics are normally satisfactory. Heuristics are often developed by combining sensible strategies to provide solutions that are as good as possible in the majority of real-life cases. But, in the remote possibility of a pathological instance forged with the intent of tricking the heuristic, the solution can be arbitrarily worse than the optimal one.

Some famous heuristics are the following:

- **Greedy algorithms** is a general term that defines any iterative algorithm that performs the most profitable action at each step, without any kind of future foresight. For example, playing a turn in chess by just looking at the current state of the board, without thinking of future moves.
- **Beam Search** can be considered an evolution of a greedy algorithm, as it takes into account a "fixed amount of foresight". Going back to the chess example, a beam

search algorithm could be one that explores all different scenarios up to three moves in the future.

- **Local Search** heuristics typically improve a given starting solution by sequentially applying "small" modifications to it. This means that a "geometry" is constructed in the solution space, and all the solutions nearby the current one are considered in search of an improvement. In other words, this resembles a discrete version of the famous gradient descent algorithm.
- **Genetic Algorithms** are heuristics that imitate the natural evolutionary processes. Overall, the algorithm considers a pool of candidate solutions. Each solution is encoded in a problem-dependent way, to resemble the DNA. Each step of the algorithm contains a reproduction phase and a selection phase. In the reproduction phase, new solutions are generated by combining already present solutions. In the selection phase, each solution is scored with a *fitness function* and the lowest solutions are discarded.
- **Matheuristics** can be described as local search heuristics that perform the local search step by solving an optimization problem using, for example, a MILP solver. Matheuristics are typically superior to classical local search procedures as they are able to explore efficiently a bigger neighborhood.

As we will use local search procedures, let us study them more deeply. These heuristics are used to solve optimization problem by iteratively exploring the solution space. The exploration produces a sequence of feasible solutions x_i. The central idea of the local search procedures is that these solutions should be pairwise near to each other. Indeed, to use a local search heuristic a *neighborhood* must be defined for each feasible solution. Typically, some sort of similarity measure is used, for example the *Hamming distance*. Procedure 1.1 summarizes the main steps of a text-book local search procedure.

Procedure 1.1: Generic Local Search Procedure

1. **Initialization**: selection of an initial (feasible) solution x_1 as current solution and computation of its objective function value $f(x_1)$;
2. **Neighborhood generation**: construction of a (totally feasible) neighborhood $\mathcal{N}(x_i)$ of the current solution x_i and selection of a candidate solution $\tilde{x} \in \mathcal{N}(x_i)$;
3. **Acceptance test**: check whether solution $\tilde{x}$ should be accepted to replace x_i. In the negative case the current solution remains $x_{i+1} = x_i$. In the positive case the current solution must be updated: $x_{i+1} = \tilde{x}$. Typically, this step is based on randomness to "escape local optima" ;
4. **Stopping test**: if the test is positive stop the local search algorithm and return the best solution encountered, else go to the Neighborhood generation step

If the best solution in the neighborhood is selected at each step, the local search heuristic is said to follow the *steepest descent* strategy. On the other hand, if the first solution

that improves the current one is selected, the local search heuristic is said to follow a *first improvement* strategy.

Approximation algorithms, on the other hand, are created in order to address worst cases. It can be proved that their solutions are always "almost good as" the optimal one. The concept of approximation ratio is used to quantify by how much their solutions differ from the optimal one. The approximation ratio of an algorithm used to solve an optimization problem is defined as the worst possible ratio between the suboptimal value returned by the algorithm and the optimum of the optimization problem. If this ratio exists and is finite, then the algorithm is called an approximation algorithm.

Schematically, let $\mathcal{I}$ be any instance of an optimization problem and $\mathcal{H}$ be a candidate approximation algorithm to solve the optimization problem. We can define the approximation ratio ρ of the algorithm $\mathcal{H}$ as:

$$\rho = \max_{\mathcal{I}} \quad \frac{\text{Value of the solution provided by } \mathcal{H} \text{ applied to } \mathcal{I}}{\text{Value of the optimal solution of } \mathcal{I}} \tag{1.1}$$

This optimization problem is usually solved indirectly by proving problem-specific properties that connect the value of the algorithm solution with the optimal one. In this work, we try to do the opposite: we will add constraints to this optimization problem to encode the approximation algorithm logic and the optimal solution value. Finally, exploiting the power of commercial solvers, we will solve the optimization problem and get the approximation ratios.

For some optimization problems, there are approximation algorithms having the ability to "control" the approximation ratio, getting it arbitrarily close to 1. They are called *Approximation Schemes*. There are two kinds of approximation schemes: *Polynomial Time Approximation Scheme* (PTAS) and *Fully Polynomial Time Approximation Scheme* (FPTAS).

A PTAS is an approximation algorithm that offers a controllable approximation ratio. Indeed, a PTAS takes as input an arbitrary parameter $\epsilon > 0$ and an instance of the optimization problem and yields a solution with an approximation ratio $\rho < 1 + \epsilon$. Moreover, a PTAS must have a runtime complexity polynomial in the input instance size.

A FPTAS is a PTAS whose runtime complexity is polynomial both in the input instance size and in $\frac{1}{\epsilon}$.

1.5 What are LPT and LS?

As discussed, the problem $QM||C_{max}$ is in $\mathcal{NP}$. In other words, no known algorithm can solve it efficiently. Hence, the need for developing efficient algorithms that find suboptimal solutions. Surely, two famous algorithms are the List Scheduling (LS) rule and the Longest Processing Time (LPT) rule.

List Scheduling is an iterative algorithm that starts from an empty schedule and, given a fixed list of jobs, updates the schedule at each step by assigning the job to the machine that would finish it first given the current schedule. List Scheduling can also be applied to other problems, such as $P2||C_{max}$ or even $RM||C_{max}$. We give here a pseudocode implementation of the List Scheduling algorithm for $P2||C_{max}$ because it is probably the easiest machine scheduling problem.

Procedure 1.2: List Scheduling for $P2||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
Output: The value C_{max} of the List Scheduling schedule
// Initialize an empty schedule
// T_1, T_2 are sums of jobs lengths assigned to each machine
$T_1 \leftarrow 0$;
$T_2 \leftarrow 0$;
// Now start building the schedule iteratively
for $j = 1, \dots, J$ **do**
  // Assign the job to the machine that would finish it first
  if $T_1 + p_j \leq T2 + p_j$ **then**
    // Assign job j to machine 1 and increment T_1
    $T_1 \leftarrow T_1 + p_j$;
  end
  else
    // Assign job j to machine 2 and increment T_2
    $T_1 \leftarrow T_1 + p_j$;
  end
end
// Now compute the schedule value by C_{max} rule
$C_{max} \leftarrow \max\{T_1, T2\}$;

The List Scheduling algorithm is an example of a greedy heuristic. It is greedy, as each job is placed without caring for the next ones. It is also a (very simple) heuristic, as it is sensible to place each job to minimize, at each step, the maximum total running time.

The List Scheduling algorithm is easily extendable to the $QM||C_{max}$ problem, by introducing in the input also the speeds of the machines and considering them during the assignment process:

Procedure 1.3: List Scheduling for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The value C_{max} of the List Scheduling schedule
// Initialize an empty schedule
// T_m is sums of jobs lengths assigned to machine m
1 $T_1, \dots, T_M \leftarrow 0$;
// Now start building the schedule iteratively
2 **for** $j = 1, \dots, J$ **do**
  // First compute all possible partial completion time
3 $F_m \leftarrow (T_m + p_j)q_m \quad \forall m = 1, \dots, M$;
  // Then find the machine $\tilde{m}$ that would finish it first (minimum partial completion time)
4 $\tilde{m} = \operatorname{argmin}\{F_1, \dots, F_M\}$;
  // Then assign job j to machine $\tilde{m}$ and increment $T_{\tilde{m}}$
5 $T_{\tilde{m}} \leftarrow T_{\tilde{m}} + p_j$;
6 **end**
// Now compute the schedule value by C_{max} rule
7 $C_{max} \leftarrow \max\{q_1 T_1, \dots, q_M T_M\}$;

Even if applied to $QM||C_{max}$ this algorithm is very fast: it is linear in the number of jobs J and in the number of machines M. In other words, its complexity is $\mathcal{O}(JM)$. Let us try to deduce it from the pseudocode:

1. The first row performs exactly M operations, hence it has a complexity of $\mathcal{O}(M)$.
2. The second row is a fixed `for` loop over the jobs, so each further row complexity should be multiplied by J.
3. The third row performs M times the same two operations: addition and multiplication. Hence, we could say that inside the `for` loop has a complexity of $\mathcal{O}(2MJ) = \mathcal{O}(MJ)$.
4. The fourth row has to find the minimum in a list of M numbers. Hence, inside the `for` loop, has a complexity of $\mathcal{O}(MJ)$. In fact, to find a minimum of a list, we just need to scan the list once keeping a pointer to the minimum element found so far.
5. The fifth row is a single operation that requires $\mathcal{O}(1)$. So inside the loop, it brings a complexity of $\mathcal{O}(J)$.

By summing the complexity of each row we get:

$$\mathcal{O}(M) + \mathcal{O}(MJ) + \mathcal{O}(MJ) + \mathcal{O}(J) = \mathcal{O}(MJ)$$

If the number of machines M is fixed than the overall complexity is just $\mathcal{O}(J)$.

The LPT algorithm is just LS applied to the sorted list of jobs in decreasing job length order.

Procedure 1.4: LPT for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The value C_{max} of the LPT schedule
`// Sort the jobs by decreasing length`
1 $(p_j)_{j=1,\dots,J} \leftarrow \text{sort}((p_j)_{j=1,\dots,J})$;
`// Apply LS and return its schedule`
2 ListScheduling$((p_j)_{j=1,\dots,J},(q_m)_{m=1,\dots,M})$;

Let us try again to derive the complexity of this algorithm directly from the pseudocode:

1. The first row sorts a list of J numbers. To do that, we can apply the *Heapsort* algorithm that has a complexity of $\mathcal{O}(J \log J)$.
2. The second row is just List Scheduling, hence it has a complexity of $\mathcal{O}(MJ)$.

Supposing the number of machines fixed, by summing the complexity of each row we get:

$$\mathcal{O}(J \log J) + \mathcal{O}(MJ) = \mathcal{O}(J \log J) + \mathcal{O}(J) = \mathcal{O}(J \log J)$$

Hence, the LPT algorithm has a log-linear complexity, only due to the initial sorting of the jobs. This means that the LPT algorithm is not simply slower than the List Scheduling, but it has greater complexity. Surely, there is a benefit for this added complexity. For $PM||C_{max}$, both algorithms are approximation algorithms. LPT has a tight approximation ratio of $\frac{4}{3} - \frac{1}{3M}$ and List Scheduling has an approximation ratio of $2 - \frac{1}{M}$. For $QM||C_{max}$, the LPT algorithm has an approximation ratio $\rho \leq 2$, as we will discuss later. The List Scheduling algorithm is not even an approximation algorithm (see Lenstra and Shmoys [2019]).

Figure 1.1 shows the LPT algorithm at work. There are 5 jobs and 2 machines. The jobs lengths are {5,3,2,2,1}. For the sake of simplicity, the machines are identical and they have the same speed factor of 1. The final makespan is 7.

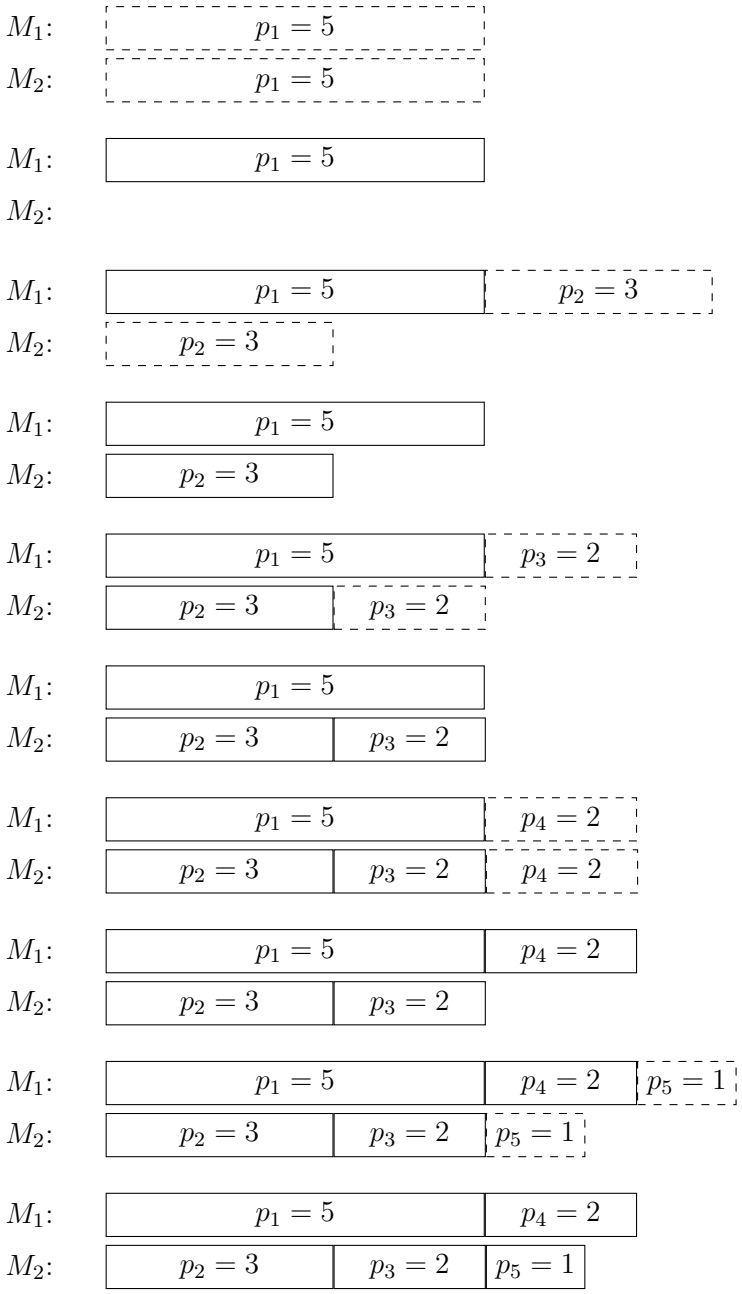

Figure 1.1. An example of LPT schedule building process.

Chapter 2

Related Work

The seminal work of Graham [1969] is fundamental as he was the first to analyze the LPT algorithm for $PM||C_{max}$ as an approximation algorithm. In that work, he first proved that the approximation ratio of LPT is $\frac{4}{3} - \frac{1}{3M}$, where M is the number of machines.

The scheduling problem on uniform machines, i.e., the $QM||C_{max}$ problem, was introduced by Horowitz and Sahni [1976]. They developed two PTASs, one for $Q2||C_{max}$ and one for $R2||C_{max}$

A series of results focus on bounding the approximation ratio ρ of LPT applied to the problem $Q||C_{max}$, i.e., when the number of machines M is not fixed. The first result was by Gonzalez et al. [1977]. They proved that $\frac{3}{2} \leq \rho \leq 2$. The lower bound was proved by explicitly giving a family of instances such that, in the limit $M \to \infty$, attains an approximation ratio of $\frac{3}{2}$. Afterward, Dobson [1984] gave an instance where the approximation ratio is ≈ 1.512, hence improving the lower bound. He also improved the upper bound to $\frac{19}{12} \approx 1.583$ by making a connection to a bin packing problem. Then Friesen [1987] gave an instance where the approximation ratio is ≈ 1.52, hence improving the lower bound. He also proved, independently of Dobson, a worse upper bound: $\frac{5}{3} \approx 1.666$. Finally, Kovács [2010] tightened the bounds to $1.54 \leq \rho \leq 1 + \frac{\sqrt{3}}{3} \approx 1.5773$.

Regarding the LPT algorithm applied to the $QM||C_{max}$ problem, in recent years Mitsunobu et al. [2022] found the approximation ratio when the number of processors M is 2,3,4,5: $\rho_2 \approx 1.28, \rho_3 \approx 1.38, \rho_4 \approx 1.43, \rho_5 \approx 1.46$. The best previous result was by Gonzalez et al. [1977] who showed that $\rho \leq \frac{2M}{M+1} \quad \forall M$.

Note how all these results are independent of the machines speeds. A beautiful parametric analysis of the approximation ratio of the LPT algorithm applied to $Q2||C_{max}$ was done by Mireault et al. [1997]. They designed a function $f(q_1, q_2)$ such that

$$\rho(q_1, q_2) \leq f(q_1, q_2) \quad \forall q_1, q_2 \in \mathbb{R}^+$$

Then they produced examples to prove that the inequality is tight for all machines speeds. To get this beautiful result, they hinted at the usage of the optimization problem method

to prove approximation ratios. In particular, they explicitly wrote some optimization problems, but they did not solve them directly, probably for a lack of resources. Instead, they manually analyzed some constraints and proved the approximation ratios in a standard analytic fashion.

Regarding the LPT algorithm applied to $Q2||C_{max}$, Massabò et al. [2016] found some interesting posterior bounds. In this context, posterior bound refers to information that is available only after the application of the LPT algorithm, for example, the index of the machines where the makespan will take place or the index of its last inserted job. They developed a posterior worst-case performance ratio bound and, through examples, showed that the ratio is tight.

Moving on from the LPT algorithm applied to $Q2||C_{max}$, Koulamas and Kyparisis [2009] proposed a modified version of the LPT algorithm and showed that their version is an approximation algorithm for $Q2||C_{max}$ with a ratio of $\sqrt{\frac{3}{2}} \approx 1.2247$. Note that this approximation ratio is an improvement over the LPT approximation ratio for $Q2||C_{max}$, which is $\rho_2 \approx 1.28$. The main difference in their algorithm is an initial "brute-force" phase on a fixed number of large jobs.

In recent years, Della Croce et al. [2019] developed an approximation algorithm for $P2||C_{max}$ with an approximation ratio of $\frac{13}{12}$. Their algorithm was essentially the LPT algorithm followed by a step of a local search procedure. Focusing on $PM||C_{max}$, Della Croce and Scatamacchia [2020] developed a modification of the LPT algorithm that improves Graham [1969] bound from $\frac{4}{3} - \frac{1}{3M}$ to $\frac{4}{3} - \frac{1}{3(M-1)}$ for $M \geq 3$ and from $\frac{7}{6}$ to $\frac{9}{8}$ for $M = 2$.

As one can imagine, there is a myriad of variations of machines scheduling problems. Refer to "Elements of Scheduling" by Lenstra and Shmoys [2019] or to "Scheduling" by Pinedo [2012] as a starting point for further references.

Chapter 3

Proposed approach

3.1 Description of the Problem

We focus on the problem known in literature as $QM||C_{max}$, that is: given a set of jobs, each characterized by its length, and a set of machines, each characterized by its speed, find the assignment of the jobs to the machines that minimize the maximum completion time.

To properly state the $QM||C_{max}$ problem, we use the following notation:

- $M \in \mathbb{N}$ is the number of machines and $J \in \mathbb{N}$ is the number of jobs.
- An instance is the pair $\mathcal{I} = \left(\{q_m\}_{m=1,\dots,M}, \{p_j\}_{j=1,\dots,J}\right)$ where q_m is the speed factor of the machine m and p_j is the length of job j.
- A schedule is defined as a function $\mathcal{S} : \{1, \dots, J\} \to \{1, \dots, M\}$.
- The makespan of the m-th machine of an instance $\mathcal{I}$ with respect to a schedule $\mathcal{S}$ is defined as:
$$\mathcal{C}_{m,\mathcal{I},\mathcal{S}} = \sum_{j \in \mathcal{S}^{-1}(m)} q_m p_j$$
Note that if a speed factor q_m is $q_m \geq 1$, it plays the role of a "slowness" factor.
- Finally, the makespan of an instance $\mathcal{I}$ with respect to a schedule $\mathcal{S}$ is defined as:
$$\mathcal{C}_{\mathcal{I},\mathcal{S}} = \max_{m \in \{1,\dots,M\}} \mathcal{C}_{m,\mathcal{I},\mathcal{S}}$$

So the problem $QM||C_{max}$ is: given an instance $\mathcal{I}$ find an optimal schedule $\mathcal{S}^*$ such that the makespan $\mathcal{C}_{\mathcal{I},\mathcal{S}}$ is minimized.

We can try to solve the problem to optimality using a commercial Mixed-Integer Linear Programming (MILP) solver as the problem admits a MILP formulation:

Optimization Model 3.1: MILP model for solving $QM||C_{max}$

$$
\begin{aligned}
C^* = \min_{C, x_{j,m}} \quad & C \\
\text{s.t.} \quad & C \geq q_m \sum_{j=1}^{J} x_{m,j} p_j && (\forall m \in \{1, \ldots, M\}) \\
& \sum_{m} x_{m,j} = 1 && (\forall j \in \{1, \ldots, J\}) \\
& x_{m,j} \in \{0,1\} && (\forall m \in \{1, \ldots, M\}, \forall j \in \{1, \ldots, J\}) \\
& C \geq 0
\end{aligned}
$$

Where for all $m \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, J\}$, $x_{m,j}$ is a binary variable whose value is 1 if job j is assigned to machine m, i.e. $\mathcal{S}^*(j) = m$, and 0 otherwise. Note that the above optimization problem is a Mixed-Integer Linear Program (MILP), as its variables are C and $\{x_{j,m}\}$, and it is "parametric" in $\{q_m\}$ and $\{p_j\}$.

Alas, while solving a continuous linear problem is a polynomial problem, for example using the *ellipsoid* method from Karmarkar [1984], solving a mixed-integer linear program is $\mathcal{NP}$. The school-book solution process is typically the *Branch and Bound* algorithm (see Tadei and Della Croce [2010]). This algorithm iteratively performs a tree-like search procedure, to find the optimal integer variable values. At each step of the algorithm, a leaf is selected, where an integer variable is fixed with a constraint and a new continuous linear program is solved. Then the algorithm may perform a pruning step where all the leafs that can be proved to be suboptimal are discarded and no more explored.

Hence, if a given instance has too many jobs, even the best MILP solver cannot solve it to optimality. So we apply a scheduling algorithm $\mathcal{H}$ and get a suboptimal result. After getting the schedule provided by algorithm $\mathcal{H}$, one could ask how good or bad is this schedule with respect to the unobtainable optimal one. This question can be addressed by considering the approximation ratio ρ. Recall equation (1.1):

$$
\rho = \max_{\mathcal{I}} \quad \frac{\text{Value of the solution provided by } \mathcal{H} \text{ applied to } \mathcal{I}}{\text{Value of the optimal solution of } \mathcal{I}} \qquad \text{((1.1) revisited)}
$$

If we solve this optimization problem and find a finite ρ than $\mathcal{H}$ is an approximation algorithm with approximation ratio equal to ρ. This means that, even in the worst possible case, the algorithm $\mathcal{H}$ schedule value will be at most ρ times worse than the optimum schedule value.

The above optimization problem is too "high level" to be solved in practice. We need to encode into constraints all three elements of the problem: any instance $\mathcal{I}$, the value of the solution provided by $\mathcal{H}$ applied to $\mathcal{I}$ and the value of the optimal schedule of $\mathcal{I}$.

In the next chapter we enter into details on how we solve all these issues for the LPT algorithm and also for new algorithms.

3.2 Methodology

In this section, we delineate a methodology that we will use with all the algorithms.

It is clear that it is impossible to study any conceivable algorithm in a unified manner and elegantly. So we need to delineate a particular family of algorithms we are interested in. Taking inspiration from the work of Della Croce et al. [2019], we will consider algorithms that produce a partial schedule using only a sub-list of the jobs and then complete that schedule with List Scheduling. This approach is also favorable from a complexity point of view.

As we discussed, the LPT complexity is $\mathcal{O}(J \log J)$, only due to the required sorting of all the jobs. One could wonder if it is possible to devise an algorithm that could both reduce the complexity (by ordering a small fixed number of jobs) and improve the approximation ratio. So we focus on the following algorithm structure, denoted hereafter as "meta-algorithm":

Procedure 3.1: Meta-Algorithm for $QM||C_{max}$

Parameters: A number $L \in \mathbb{N}$
A scheduling algorithm `H` for $QM||C_{max}$
Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: C

1 Find the LM-th longest job in $\{p_j\}_{j=1,\dots,J}$
2 Let $\mathcal{J}^+$,$\mathcal{J}^-$ be, respectively, the list containing all the jobs larger than the LM-th one and the list containing the remaining jobs (if any).
3 Sort only the list $\mathcal{J}^+$.
4 Apply algorithm `H` to the sorted list $\mathcal{J}^+$ and obtain a (partial) schedule $\mathcal{S}$.
5 Complete the schedule $\mathcal{S}$ using list scheduling with $\mathcal{J}^-$.

Below we assess the complexity of the proposed Meta-Algorithm:

1. As L and M are fixed numbers, the complexity of the first row is $\mathcal{O}(J)$. In fact, finding the k-th largest element in an unsorted list of length can be done in linear time $\mathcal{O}(LMJ)$ using a multiple scans approach.
2. The second row can be implemented by simply scanning each job once and placing it in either one of the lists. Hence, it has a complexity of $\mathcal{O}(J)$.
3. By construction, the list $\mathcal{J}^+$ has exactly LM elements. Hence, the complexity of the third row is $\mathcal{O}(LM \log(LM))$, using for example the *Heapsort* algorithm.
4. The fourth row complexity depends on the complexity of the algorithm `H`. As `H` accepts as input the list $\mathcal{J}^+$ (length LM) and the list of machines (length M), `H` complexity has the form $\mathcal{O}(f(LM, M))$.

5. The fifth row is just an application of the List Scheduling algorithm, whose complexity is the product of the number of machines times the length of the list of the input jobs: $\mathcal{O}\left(M(J - ML)^+\right)$.

Hence, the overall complexity of the Meta-Algorithm is linear in J when M and L are fixed:

$$\mathcal{O}\left(LMJ + LM\log(LM) + f(LM, M) + M(J - ML)^+\right) = \mathcal{O}\left(LMJ\right) = \mathcal{O}\left(J\right) \tag{3.1}$$

It is important to note that the overall complexity of Meta-Algorithm is independent of the complexity of the algorithm `H` used. Moreover, its complexity is $\mathcal{O}\left(J\right)$, hence it is an improvement over the complexity of LPT, that is $\mathcal{O}\left(J\log J\right)$.

We want to study the approximation ratio of multiple algorithms that follow the structure of the Meta-Algorithm. Hence, we need to fix a scheduling algorithm `H`, applicable to the $QM||C_{max}$ problem, and a number $L \in \mathbb{N}$, that will limit the number of jobs in the input of `H`. Let C^H be the value of the suboptimal schedule produced by Meta-Algorithm instantiated with `H` and L. Let C^* be the value of the optimal schedule. Then, we have to solve the following optimization problem:

$$\rho = \max_{\text{instance } \mathcal{I}} \frac{C^H}{C^*} \tag{3.2}$$

We remark that the J jobs lengths and the M machines speeds should be variables of problem (3.2). While M can be considered fixed, for example $M = 2$ if we are interested in finding approximation ratios for $Q2||C_{max}$, surely the number of jobs J is not fixed as any instance $\mathcal{I}$ should be taken into consideration, regardless of its length. An elegant solution could be to add an infinite number of variables, one for each possible job, and treat the resulting problem with variational methods. Alas, the commercial solvers are able to represent only finite dimensional problems, hence a different strategy must be followed.

We propose the following approach to solve (3.2). We split the optimization problem domain into two parts, according to the index of the *critical job* K. The critical job is the smallest job whose removal causes the makespan to strictly decrease. For example, looking back at Figure 1.1, the critical job index is $K = 4$. We obtain the following:

$$\rho = \max\left\{ \max_{\substack{\text{instance } \mathcal{I} \text{ with} \\ K \leq LM}} \frac{C^H}{C^*}, \max_{\substack{\text{instance } \mathcal{I} \text{ with} \\ K \geq LM+1}} \frac{C^H}{C^*} \right\}$$

Note that choosing LM as the "threshold" for K, we actually separate the algorithm `H` from the List Scheduling algorithm. In fact, if $K \geq LM + 1$, the critical job is scheduled according to the List Scheduling algorithm. On the other hand, if $K \leq LM$, the critical job is scheduled by the algorithm `H`.

Moreover, the first sub-problem can be simplified from an infinite dimensional problem to a finite dimensional one, using the following proposition:

Proposition 3.2.1. *It holds that:*

$$\max_{\text{instance } \mathcal{I} \text{ with } K \leq LM} \frac{C^H}{C^*} \leq \max_{\text{instance } \mathcal{I} \text{ with } J = LM} \frac{C^H}{C^*}$$

Proof. Recall the structure of the Meta-Algorithm. If the critical job index is K and $K \leq LM$, then the critical job has been scheduled by algorithm `H` and not by the List Scheduling part. Moreover, by the definition of critical job, any job scheduled by List Scheduling does not increase the makespan. Otherwise, if such a job existed, then it would be the new critical job, contradicting the fact that $K \leq LM$. So, we can remove the jobs in $\mathcal{J}^-$ without changing the performance of the overall algorithm.
On the other hand, removing the jobs in $\mathcal{J}^-$ can only reduce the makespan of the optimal schedule, i.e., reducing C^*.
In other words, for any instance $\mathcal{I}$ with $K \leq LM$, there is an instance $\mathcal{I}'$ with $J \leq LM$ such that the makespan of the overall algorithm cannot decrease and the makespan of the optimal schedule cannot increase. □

Applying the above proposition, we get the following inequality:

$$\rho \leq \max \left\{ \max_{\substack{\text{instance } \mathcal{I} \text{ with} \\ J = LM}} \frac{C^H}{C^*}, \max_{\substack{\text{instance } \mathcal{I} \text{ with} \\ K \geq LM+1}} \frac{C^H}{C^*} \right\} \tag{3.3}$$

The first sub-problem is algorithm dependent, and it is tackled with commercial solvers. The latter sub-problem is instead more general, and it will be tackled in the next section.

Chapter 4

Main Propositions

The goal of this section is to devise a bound for the Meta-Algorithm if the critical job is in $\mathcal{J}^-$. To simplify the notation, given the machines speed factors, we define the following quantity:

$$\mathcal{Q} = \sum_{m=1,\dots,M} \frac{1}{q_m}$$

The following proposition provides a bound for the optimal value of an instance (no reference to any approximation algorithm).

Proposition 4.0.1. *Let C^* be the value of an optimal schedule $\mathcal{S}^* = \{\mathcal{S}_1^*, \dots, \mathcal{S}_M^*\}$ with respect to an instance $\mathcal{I} = (\{q_m\}, \{p_j\})$. Then:*

$$\sum_{j=1,\dots,J} p_j \leq C^* \mathcal{Q}$$

Proof. By the definition of C^*:

$$\begin{aligned} C^* &= \max_{m=1,\dots,M} \left\{ q_m \sum_{j \in \mathcal{S}_m^*} p_j \right\} \\ &\geq \sum_{m=1,\dots,M} \left(\alpha_m q_m \sum_{j \in \mathcal{S}_m^*} p_j \right) \quad \forall \alpha_1, \dots, \alpha_M > 0 : \sum_{m=1,\dots,M} \alpha_m = 1 \end{aligned} \tag{4.1}$$

The inequality follows from the fact that the maximum of a set of scalars is greater than or equal to any convex combination of such scalars. Hence, α_m are free coefficients of an arbitrary convex combination. We choose the following:

$$\alpha_m = \frac{1}{q_m \mathcal{Q}} \quad \forall m = 1, \dots, M \tag{4.2}$$

This choice is valid as the resulting coefficients are positive and sum up to 1:

$$\sum_{m=1,\dots,M} \alpha_m = \sum_{m=1,\dots,M} \frac{1}{q_m \mathcal{Q}} = \frac{1}{\mathcal{Q}} \sum_{m=1,\dots,M} \frac{1}{q_m} = \frac{1}{\mathcal{Q}} \mathcal{Q} = 1$$

By substituting (4.2) in (4.1) we get the result:

$$\begin{aligned} C^* &\geq \sum_{m=1,\dots,M} \left(\alpha_m q_m \sum_{j \in \mathcal{S}_m} p_j \right) \\ &= \sum_{m=1,\dots,M} \left(\frac{1}{q_m \mathcal{Q}} q_m \sum_{j \in \mathcal{S}_m} p_j \right) \\ &= \sum_{m=1,\dots,M} \left(\frac{1}{\mathcal{Q}} \sum_{j \in \mathcal{S}_m} p_j \right) \\ &= \frac{1}{\mathcal{Q}} \sum_{m=1,\dots,M} \left(\sum_{j \in \mathcal{S}_m} p_j \right) \\ &= \frac{1}{\mathcal{Q}} \sum_{j=1,\dots,J} p_j \end{aligned}$$

□

The next proposition provides a bound to the approximation ratio of each scheduling algorithm using list scheduling.

Proposition 4.0.2. *Consider a scheduling algorithm H and let $\mathcal{I}$ be an instance such that the critical job K has been scheduled according to the list scheduling rule. Then the following inequality holds:*

$$\frac{C^H}{C^*} \leq 1 + \frac{(M-1)p_K}{C^* \mathcal{Q}}$$

Proof. Let $t_1, \dots, t_M$ be the sums of the jobs scheduled respectively on machines $m = 1, \dots, M$ before the K-th job, i.e., the critical one.
The makespan C^H is equal to the completion time of the critical job p_K. Moreover, due to list scheduling, the critical job is scheduled on the machine that completes it first. Hence, the following holds:

$$C^H = \min_{\tilde{m}=1,\dots,M} q_{\tilde{m}} \left(t_{\tilde{m}} + p_K\right) \leq q_m \left(t_m + p_K\right) \quad \forall m = 1, \dots, M$$

Dividing by q_m and summing on m we get:

$$\begin{aligned}
&\implies \sum_{m=1,\dots,M} \frac{C^H}{q_m} \leq \sum_{m=1,\dots,M} \left(t_m + p_K\right) \\
&\iff C^H \sum_{m=1,\dots,M} \frac{1}{q_m} \leq \sum_{m=1,\dots,M} \left(t_m + p_K\right) \\
&\iff C^H \mathcal{Q} \leq \sum_{m=1,\dots,M} \left(t_m + p_K\right) \\
&\iff C^H \mathcal{Q} - (M-1)p_K \leq p_K + \sum_{m=1,\dots,M} t_m
\end{aligned}$$

Note that we can apply Proposition 4.0.1, because:

$$p_K + \sum_{m=1,\dots,M} t_m = \sum_{j=1,\dots,K} p_j \leq \sum_{j=1,\dots,J} p_j \leq C^* \mathcal{Q}$$

Correspondingly, we get the following inequality:

$$C^H \mathcal{Q} - (M-1)p_K \leq C^* \mathcal{Q}$$

□

Proposition 4.0.3. *Under the same hypotheses of Proposition 4.0.2, let additionally suppose that there are X jobs greater than p_K. It holds that:*

$$\frac{C^H}{C^*} \leq 1 + \frac{M-1}{X+1}$$

Proof.

$$C^* \mathcal{Q} \geq \sum_{j=1,\dots,J} p_j \geq \left(\sum_{\text{jobs greater than } p_K} p_j \right) + p_K \geq X p_K + p_K = (X+1) p_K$$

That is:

$$\frac{p_K}{C^* \mathcal{Q}} \leq \frac{1}{X+1}$$

By plugging this last inequality in Proposition 4.0.2 we get the result. □

As a corollary of Proposition 4.0.3, the next proposition gives a bound for every algorithm that follows the framework of the Meta-Algorithm. This bound depends only on the

number of machines M and the parameter L.

Proposition 4.0.4. *Suppose that the Meta-Algorithm is applied to an instance $\mathcal{I}$ such that $K \geq LM + 1$. Then it holds:*

$$\frac{C^H}{C^*} \leq 1 + \frac{M-1}{LM+1}$$

Proof. As $K \geq LM + 1$, the Meta-Algorithm puts $p_K \in \mathcal{J}^-$, so the critical job K will be scheduled according to list scheduling. Moreover, any job in $\mathcal{J}^+$ is greater than p_K. By remembering that $|\mathcal{J}^+| = LM$ we have that at least LM jobs are greater than p_K. Applying Proposition 4.0.3 we get the result. □

Note that if we consider problem $PM||C_{max}$, i.e. identical processors, a better bound can be proved:

$$\frac{C^H}{C^*} \leq 1 + \frac{M-1}{(L+1)M}$$

Indeed, the following proposition holds.

Proposition 4.0.5. *Suppose that the Meta-Algorithm is applied to an instance $\mathcal{I}$ such that $K \geq LM + 1$. Then:*

$$\frac{C^H}{C^*} \leq 1 + \frac{M-1}{\mathcal{Q}(L+1)\min\limits_{m=1,\dots,M} q_m}$$

Proof. As $K \geq ML + 1$ then, for the pigeonhole principle, in the optimal schedule it exists a machine with at least $(I + 1)$ jobs. Moreover, as $K \geq ML + 1$ then $p_K \geq p_j, \forall j = 1, \dots, LM + 1$. So the following inequality holds:

$$C^* \geq (L+1)p_K \min_{m=1,\dots,M} q_m$$

Hence:

$$\frac{p_K}{C^*} \leq \frac{1}{(L+1)\min\limits_{m=1,\dots,M} q_m}$$

We recover the thesis applying this last inequality to Proposition 4.0.2 □

Chapter 5

Developed Algorithms - Examples

In this section we finally define the new algorithms. Remember that each of them is created from Meta-Algorithm by specifying an algorithm `H` and a positive integer L. In fact, the algorithm `H` is applied only to the sorted list containing the LM largest jobs. Afterwards, List Scheduling is applied to the remaining jobs.

Consequently, a new approximation ratio must be proved for each choice of `H` and L, as the initial part of the resulting algorithm is different. To study these approximation ratios, we will use the following inequality (obtained by plugging Proposition 4.0.4 into (3.3)):

$$\rho \leq \max \left\{ \max_{\substack{\text{instance } \mathcal{I} \text{ with} \\ J = LM}} \frac{C^H}{C^*}, 1 + \frac{M-1}{LM+1} \right\} \tag{5.1}$$

Note that only the first sub-problem involves algorithm `H` while List Scheduling is not involved. This issue will become very handy later.

In this section, for each new algorithm `H`, we give its pseudocode implementation. Afterwards we derive equality and inequality constraints. These constraints represent the logic and the rules of the algorithm `H`. With these constraints we will construct optimization problems following the first sub-problem scheme. To solve the optimization problems we use the commercial solver *Gurobi v9.2*.

Beyond `H`, also L is a free parameter of the Meta-Algorithm. Hence, also L needs to be taken into consideration when solving (5.1) for each algorithm `H`. Choosing L as small as possible has two benefits. First, according to equation (3.1), a smaller L implies faster runtimes, as only the largest LM jobs need to be "filtered". Secondly, the first sub-problem optimum increases (non strictly) as L increases. In fact, any instance with $L_1 M$ jobs is also an instance with $(L_1+1)M$ jobs by simply adding an infinitesimal job, without

altering C^H or C^*. On the other hand, the second term in (5.1) decreases as L increases. Hence, for each algorithm `H`, we will choose the smallest L such that the maximum in (1.1) is given by the first sub-problem.

In the following, several low-complexity polynomial-time algorithms are presented with their relevant approximation ratios.

5.1 LPT Algorithm

5.1.1 The Algorithm

The first algorithm `H` that we test is the classic LPT. In this way, a somehow direct comparison with Mitsunobu et al. [2022] and Mireault et al. [1997] can be made.

A succinct pseudocode implementation of the LPT algorithm for $QM||C_{max}$ problem is:

Procedure 5.1: LPT for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The LPT schedule $\mathcal{S}$ and its value C

1 Sort the jobs in decreasing order, i.e.
$i \leq j \implies p_i \geq p_j, \quad \forall i, j \in \{1, \dots, J\}$;
2 $T_m \leftarrow 0 \quad \forall m = 1, \dots, M$;
3 $\mathcal{S}_m \leftarrow \{\} \quad \forall m = 1, \dots, M$;
4 **for** $j = 1, \dots, J$ **do**
`// Assign the job to the machine that would finish it first`
`// Break eventual ties at random`
5 $m \leftarrow \underset{\tilde{m}=1,\dots,M}{\operatorname{argmin}} q_{\tilde{m}}(T_{\tilde{m}} + p_j)$;
`// Update partial schedule and partial completing times`
6 $T_m \leftarrow T_m + p_j$;
7 $S_m \leftarrow S_m \cup \{j\}$;
8 **end**
9 $C = \max_{m=1,\dots,M} q_m T_m$;

This algorithm requires in input the list of machines speed factors $\{q_m\}_{m=1,\dots,M}$ and the list of jobs lengths , $\{p_j\}_{j=1,\dots,J}$.

1. In the first row the jobs are sorted from the longest to the shortest.
2. In the second row, M variables are initialized to zero. The m-th variable, T_m, will hold the current total jobs lengths assigned to machine m. At the start, no jobs are assigned, hence all the totals are zero.

3. The third row initializes an empty schedule. In principle, there are multiple ways to represent a schedule. Here, the variable $\mathcal{S}_m$ will hold the indexes of the jobs assigned to machine m.
4. The fourth row is a `for` loop over the (sorted) jobs.
5. The fifth row is the core of LPT algorithm. Here we select the machine where job j will be assigned. The criterion is to assign the job to the machines that would finish it first. Note that $q_{\tilde{m}}T_{\tilde{m}}$ is the current completion time of machine $\tilde{m}$. If we assign the job to machine $\tilde{m}$, its completion time would be $q_{\tilde{m}}T_{\tilde{m}} + q_{\tilde{m}}p_j$, as we need to consider also the speed factor.
6. The sixth row updates the chosen machine completion time.
7. The seventh row registers the assignment in the schedule.

5.1.2 From the Algorithm to the Model

Now we want to encode the run-time logic of the algorithm in the first sub-problem of (5.1).

The main difficulty is how to encode the "LPT logic". Intuitively, let us suppose that LPT had assigned job j to machine m. This means that, at that step of the `for` loop, the machine m is such that:

$$m \in \underset{\tilde{m}=1,\dots,M}{\operatorname{argmin}} q_{\tilde{m}}(T_{\tilde{m}} + p_j)$$

Instead of dealing with the argmin operator, we can equivalently reformulate it using only inequalities. At that step of the `for` loop, it holds that:

$$q_m(T_m + p_j) \leq q_{\tilde{m}}(T_{\tilde{m}} + p_j), \quad \forall \tilde{m}$$

Note that the variables $T_{\tilde{m}}$ can be substituted with the sum of jobs lengths assigned to machine m at the proper `for` loop step.

Now we are ready to introduce the corresponding optimization problem and relevant mathematical programming formulation for LPT.

For the sake of brevity the "loop variable" j always ranges in $\{1,\dots,J\}$ and $m \in \{1,\dots,M\}$. Note that the constraint $J = LM$ is implicit, as the number of jobs in the instance is directly related to the number of variables in the model. The variables for the following mathematical programming model are:

- $q_m \in \mathbb{R}^+$ is the speed factors of the m-th machine
- $p_j \in \mathbb{R}^+$ is the length of the j-th job
- $x^{LPT}_{m,j} \in \{0,1\}$ is the indicator variable if LPT schedules job j on machine m.
- $x^*_{m,j} \in \{0,1\}$ is the indicator variable if the optimal schedule schedules job j on machine m.

- $C^{LPT} \in \mathbb{R}^+$ is the value of the LPT schedule
- $C^* \in \mathbb{R}^+$ is the value of an optimal schedule

Optimization Model 5.1: LPT

$$\max \quad \frac{C^{LPT}}{C^*}$$

$$\text{s.t.} \quad C^{LPT} = \max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right)$$

$$C^* = \max_m \left(q_m \sum_j x^*_{m,j} p_j \right)$$

$$\sum_m x^{LPT}_{m,j} = 1 \qquad (\forall j)$$

$$\sum_m x^*_{m,j} = 1 \qquad (\forall j)$$

$$x^{LPT}_{m,j} = 1 \implies \left[q_m \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{m,i} p_i \right) \leq q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{\tilde{m},i} p_i \right), \forall \tilde{m} \right] \qquad (\forall m, j)$$

$$p_1 \geq p_2 \geq \cdots \geq p_J$$

The first two constraints simply calculate the values of the LPT schedule and the optimal one, respectively. As the objective function should be maximized, and the two schedules share all jobs lengths and machines speed factors, the problem is pushed towards the worst instance with respect to the approximation ratio. The third and fourth constraints say that each job should be assigned to one and only one machine (both in LPT and optimal schedules). The fifth constraint represents the list scheduling rule. Indeed, if job j has been assigned by LPT to machine m, i.e. $x^{LPT}_{m,j} = 1$, then the machine m should be the machine that, respecting the actual loads, would finish the j-th job first. Finally, the last constraint simply fixes the order of the jobs, as LPT sorts them before applying the list scheduling rule.

Note that Optimization Model 5.1 is highly non-linear as the objective function is expressed as a ratio of variables. Further, the following proposition holds.

Proposition 5.1.1. *The objective function C_{max} is independent of the units of measure / normalization constants of the machines speeds and the jobs lengths.*

Proof. Let $A, B \in \mathbb{R}^+$ be, respectively, the units of measure / normalization constants of the machines speeds and the jobs lengths. Then it holds that:

$$\frac{\max_m \left(\frac{q_m}{A} \sum_j x^{LPT}_{m,j} \frac{p_j}{B} \right)}{\max_m \left(\frac{q_m}{A} \sum_j x^{*}_{m,j} \frac{p_j}{B} \right)} = \frac{\frac{1}{AB} \max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right)}{\frac{1}{AB} \max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right)} =$$

$$= \frac{\max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right)}{\max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right)} = \frac{C^{LPT}}{C^*}$$

□

We apply Proposition 5.1.1 and correspondingly these constraints:

- The unit of measure of the machines speeds is not fixed, and, without loss of generality, the machines can be sorted w.r.t. their speed factors. Note that the LPT algorithm does not sort the machines in any way, hence they are "equivalent". However, it is convenient to sort them by speed in the optimization model: in this way we restrict the domain and remove symmetric solutions. So we can impose the additional constraints:
$$1 = q_1 \leq q_2 \leq \cdots \leq q_M$$
- The unit of measure of the jobs lengths is not fixed. Correspondingly, the value of the optimal schedule can be arbitrarily fixed to a constant value, e.g. it can be fixed to 1. In this way, we get rid of the denominator in the objective function. So we can impose the following constraint:
$$C^* = \max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right) = 1$$
- As the speed factors are greater than 1 and the optimal schedule value is fixed to 1, it means that each individual job length must lay in [0,1].

Moreover, the implication constraints, modeling the core of LPT logic, can be "linearized" following a big-M approach (see Tadei and Della Croce [2010] for a deeper explanation of logic constraints modeling). Let B be a positive constant. If B is big enough, the implication constraints are equivalent to the following:

$$q_m \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{\tilde{m},i} p_i \right) \leq B(1 - x^{LPT}_{m,j}) \qquad (\forall m, \tilde{m}, j)$$

In fact when $x_{m,j}^{LPT} = 1$ the left-hand side becomes 0 and the original constraint is recovered. We choose B to be greater than the upper bound of the right-hand side:

$$\begin{aligned} RHS &= q_m \left(p_j + \sum_{i=1}^{j-1} x_{m,i}^{LPT} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x_{\tilde{m},i}^{LPT} p_i \right) \\ &\leq q_m \left(p_j + \sum_{i=1}^{j-1} z_{m,i}^{LPT} \right) \\ &\leq q_m \sum_{j=1}^{J} p_j \\ &\leq q_{max} \sum_{j=1}^{J} p_j \\ &\leq J q_{max} \\ &\leq B \end{aligned}$$

5.1.3 The model

So we end up with the following mixed-integer non-convex quadratic problem, that a commercial solver is capable of solving to optimality (see Achterberg and Towle [2020] for a deeper explanation of the solution process).

Optimization Model 5.2: LPT

$$\begin{aligned} \max \quad & C^{LPT} \\ \text{s.t.} \quad & C^{LPT} = \max_m \left(q_m \sum_j x_{m,j}^{LPT} p_j \right) \\ & \max_m \left(q_m \sum_j x_{m,j}^{*} p_j \right) = 1 \\ & \sum_m x_{m,j}^{LPT} = 1 \qquad (\forall j) \\ & \sum_m x_{m,j}^{*} = 1 \qquad (\forall j) \\ & q_m \left(p_j + \sum_{i=1}^{j-1} x_{m,i}^{LPT} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x_{\tilde{m},i}^{LPT} p_i \right) \leq B(1 - x_{m,j}^{LPT}) \quad (\forall m, \tilde{m}, j) \\ & 1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0 \\ & 1 = q_1 \leq q_2 \leq \cdots \leq q_M \end{aligned}$$

There are still some non-linearities: the products $q_m x_{m,j}^{LPT} p_j$, the products $q_m x_{m,j}^{*} p_j$, and

the maximum operators inside constraints. All these issues can be managed by the commercial solver, and some of them can even be linearized:

- To address the product of variables q_m, $x_{m,j}^{LPT}$ and p_j we do the following. First we introduce continuous variables $\left\{z_{m,j}^{LPT}\right\}$ and add constraints $z_{m,j}^{LPT} = x_{m,j}^{LPT} p_j$. In this way any product of three variables is decomposed into two products of two variables. Note that any product that contains the binary variable $x_{m,j}^{LPT}$ can be "linearized":

$$\begin{cases} z_{m,j}^{LPT} = x_{m,j}^{LPT} p_j \\ p_j \in [0,1] \\ x_{m,j}^{LPT} \in \{0,1\} \end{cases} \iff \begin{cases} z_{m,j}^{LPT} \leq x_{m,j}^{LPT} \\ z_{m,j}^{LPT} \leq p_j \\ z_{m,j}^{LPT} \geq p_j + (x_{m,j}^{LPT} - 1) \\ z_{m,j}^{LPT} \in [0,1] \\ p_j \in [0,1] \\ x_{m,j}^{LPT} \in \{0,1\} \end{cases}$$

  A similar transformation is also used with $z_{m,j}^{*} = x_{m,j}^{*} p_j$.

- Also, the maximum constraints can be linearized. For example, introducing a binary variable y and a big enough positive constant B, it holds that:

$$X = \max\{x_1, x_2\} \iff \begin{cases} X \geq x_1 \\ X \geq x_2 \\ x_1 - x_2 \leq By \\ x_2 - x_1 \leq B(1-y) \\ X \leq x_1 + B(1-y) \\ X \leq x_2 + By \\ y \in \{0,1\} \end{cases}$$

5.1.4 The Approximation Ratio

We can now use the developed model optimum to study the approximation ratio of the Meta-Algorithm that uses LPT as algorithm H. In particular, the developed model is a "specific setting" of the first sub-problem in (5.1), by fixing $J = LM$. The following table shows different approximation ratios by varying L:

L	$\frac{C^{LPT}}{C^*}$	$1 + \frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.2808	1.2000	1.2808
3	1.2808	1.1429	1.2808

Table 5.1. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using LPT as `H` and varying L.

Besides the approximation ratio itself, the optimization problem solution also includes a schedule with $J = LM = 4$ jobs that numerically achieves that ratio. One of them is a zero-length job. By removing it, we deduce the following instance with $M = 2$ machines and $J = 3$ jobs, which achieves the approximation ratio:

$$\mathcal{I}^{LPT} = (\{q_1 = 1, q_2 = 1.2808\}, \{p_1 = 0.7808, p_2 = p_3 = 0.5000\})$$

We can deduce an algebraic form of $\mathcal{I}^{LPT}$ from its numeric values, by solving the following system of equations:

$$\begin{cases} q_1(p_1 + p_3) = q_2(p_2 + p_3) \\ q_1(p_2 + p_3) = q_2(p_1) = 1 \\ p_2 = p_3 = \frac{1}{2} \\ q_1 = 1 \end{cases}$$

In fact:

- The first equation is justified by direct inspection of Figure 5.1, where the LPT algorithm is shown building the schedule for the instance $\mathcal{I}^{LPT}$. The last job could be assigned to any machine without changing the final makespan.
- The second row represents the value of the optimal schedule $x^*_{m,j}$, given in the solution of the optimization model. In this case, the optimal schedule assigns jobs p_2 and p_3 to machine q_1 and job p_1 to machine q_2.
- The fourth equation is directly justified from the numeric values of $\mathcal{I}^{LPT}$.
- The last equation is just the constraint from the optimization problem regarding the normalization constant of the machines speeds.

Solving the above system of equations, we get the algebraic form of $\mathcal{I}^{LPT}$:

$$\mathcal{I}^{LPT} = \left(\left\{ q_1 = 1, q_2 = \frac{1 + \sqrt{17}}{4} \right\}, \left\{ p_1 = \frac{4}{1 + \sqrt{17}}, p_2 = p_3 = \frac{1}{2} \right\} \right)$$

Hence, the Meta-Algorithm, fixing $L = 2$ and LPT as `H`, has an approximation ratio of $\frac{1+\sqrt{17}}{4} \approx 1.2808$ for the problem $Q2||C_{max}$. This is actually the same value found in Mitsunobu et al. [2022] for the "full" LPT algorithm.

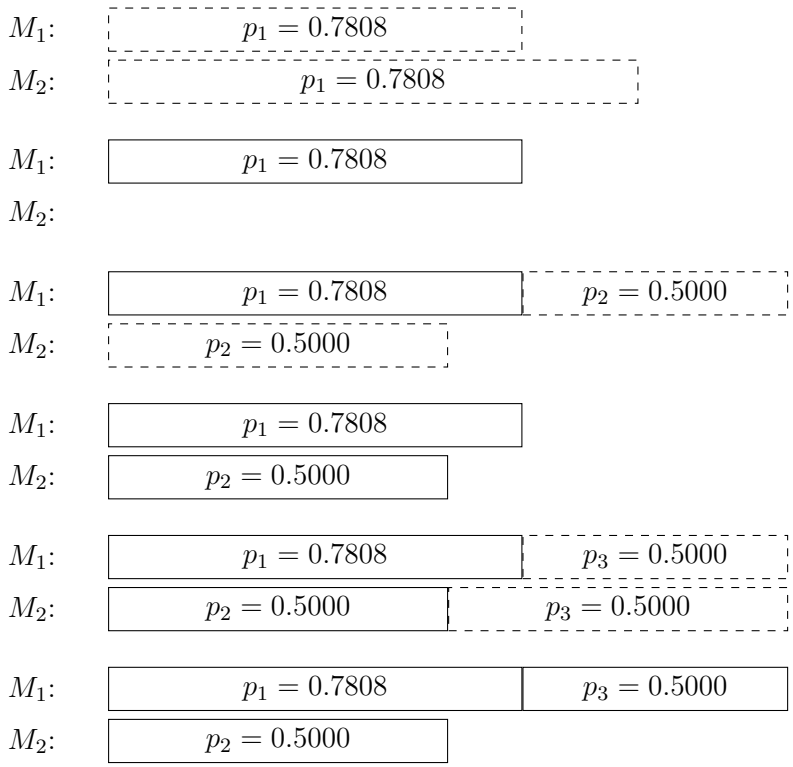

Figure 5.1. The LPT algorithm building the schedule of the instance $\mathcal{I}^{LPT}$.

5.1.5 Three Machines and some Parametric Results

If we set $M = 3$ and re-solve the model, we get the approximation ratio for $Q3||C_{max}$: 1.3837. The corresponding instance is:

$$\mathcal{I}^{LPT}_{M=3} = \{\, \{q_1 = 1, q_2 = 1.1316, q_3 = 1.3837\}\,, \\ \{p_1 = 0.8837, p_2 = 0.7227, p_3 = p_4 = 0.5000\}\, \}$$

It should be clear that by simply increasing time and computation power resources we can compute the bound for any number of machines M, without relying on long and complex analytical proofs, such those in Mitsunobu et al. [2022]. See Appendix A for deeper explanation.

It is interesting to consider the speed factors $\{q_m\}$ as parameters. Then, the same model becomes a parametric mixed-integer linear program (parametric MILP). Figure 5.3 shows the graph for $M = 2$, which is the result obtained by the complex and long analytical

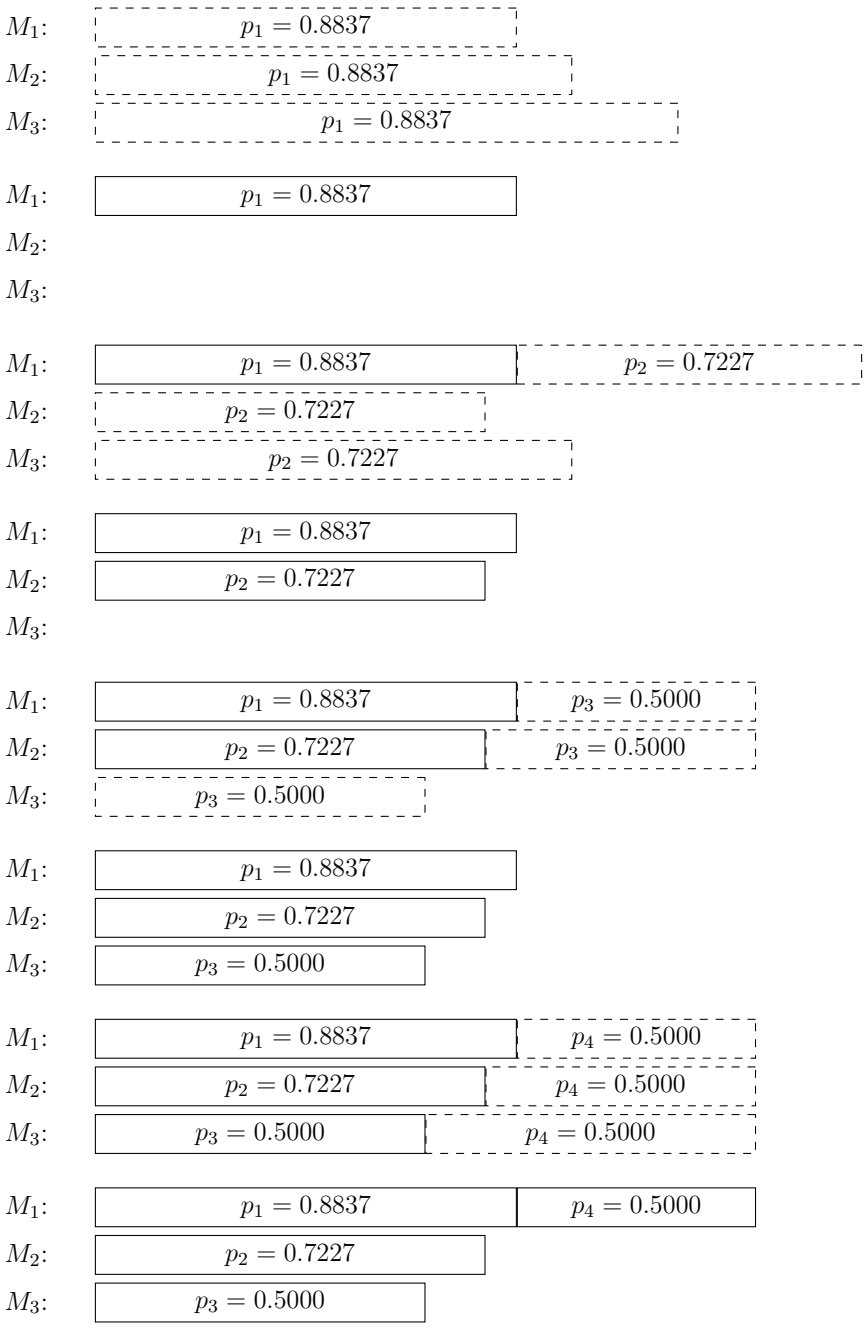

Figure 5.2. The LPT algorithm building the schedule of the instance $\mathcal{I}_{M=3}^{LPT}$.

proof in Mireault et al. [1997]. Moreover, we can also plot the graph for $M = 3$, yielding a completely novel result, as shown in Figure 5.4.

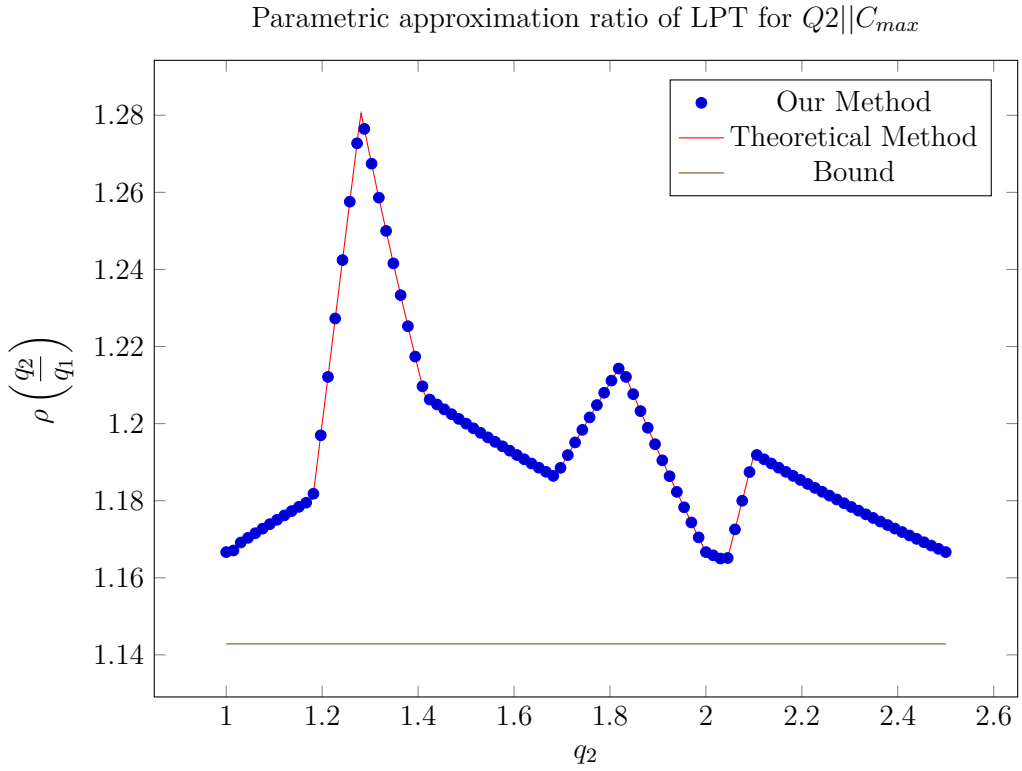

Figure 5.3. Parametric analysis of the approximation ratio of the LPT algorithm for the $Q2||C_{max}$ problem. As done in Mireault et al. [1997] we fix $q_1 = 1$ and let q_2 be free. The red solid line is the theoretical approximation ratio. Each blue dot is the optimum of the parametric MILP. The brown solid line is the $1 + \frac{M-1}{LM+1}$ bound, with $M = 2$ and $L = 3$.

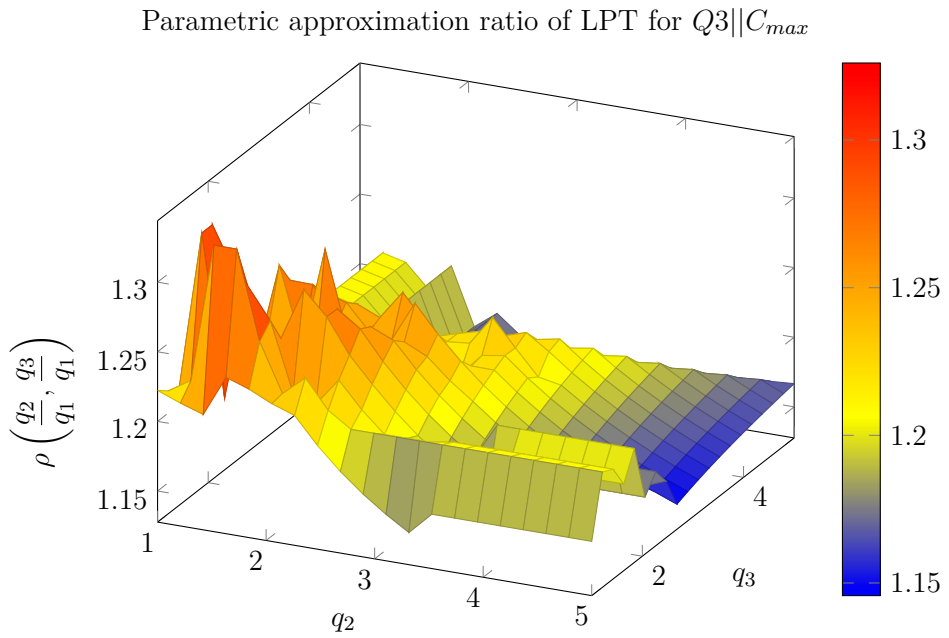

Figure 5.4. Parametric analysis of the approximation ratio of the LPT algorithm for the $Q3||C_{max}$ problem. As done in Figure 5.3, we fix the fastest machine speed $q_1 = 1$ and let q_2, q_3 be free. The surface represents $\rho\left(\frac{q_2}{q_1}, \frac{q_3}{q_1}\right)$. It was obtained by fixing different speeds pairs in Optimization Model 5.2.

5.2 Koulamas and Kyparisis Algorithm

We can also replicate the results of the algorithm proposed by Koulamas and Kyparisis [2009]. We will call it the Algorithm-$\kappa\kappa$. The Algorithm-$\kappa\kappa$ uses a positive integer parameter R and is divided into three phases. The first phase sorts the jobs by decreasing length, as LPT. The second phase, core of the algorithm innovation, is a "brute-force" phase. Here, the largest R jobs are assigned to the machines in the best possible way. In other words, every possible schedule of the first R jobs is generated, and, among these, the schedule minimizing the makespan of the R jobs is selected. In the third and last phase, the selected schedule, currently accounting only the largest R jobs, is completed using the standard LPT logic.

Procedure 5.2: Algorithm-$\kappa\kappa$ for $QM||C_{max}$

Parameters: $R \in \mathbb{N}$
Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The algorithm-$\kappa\kappa$ schedule $\mathcal{S}$ and its value C

`// First Phase`

Sort the jobs in decreasing order, i.e.
$i \leq j \implies p_i \geq p_j, \quad \forall i, j \in \{1, \dots, J\}$;

`// Second Phase`

$v_{best} \leftarrow +\infty$;
$\mathcal{S} \leftarrow \{\}$;
`// Iterate over all possible partial schedules with only` R `jobs`
for $\tilde{\mathcal{S}}$ *schedule of* R *jobs on* M *machines* **do**
  `// Compute makespan after` R `jobs`
  $v \leftarrow \max_{m=1,\dots,M} \sum_{j \in \tilde{\mathcal{S}}^{-1}(m)} q_m p_j$;
  `// Update, if needed, the best partial schedule`
  if $v < v_{best}$ **then**
    $v_{best} \leftarrow v$;
    $\mathcal{S} \leftarrow \tilde{\mathcal{S}}$;
  end
end
$T_m \leftarrow \sum_{j \in \mathcal{S}^{-1}(m)} q_m p_j \quad \forall m = 1, \dots, M$;

`// Third Phase`
`// The LPT loop starts from the` $R+1$`-th job`
for $j = R+1, \dots, J$ **do**
  $m \leftarrow \operatorname{argmin}_{\tilde{m}=1,\dots,M} q_{\tilde{m}}(T_{\tilde{m}} + p_j)$;
  $T_m \leftarrow T_m + p_j$;
  $S_m \leftarrow S_m \cup \{j\}$;
end
$C = \max_{m=1,\dots,M} q_m T_m$;

Similarly to what done with the LPT algorithm in the previous section, we now want to develop a mathematical optimization model to study the approximation ratio of the algorithm-$\kappa\kappa$. As the first and third phase of the algorithm resemble very closely the LPT algorithm, we will use Optimization Model 5.2 as a backbone model: only few constraints must be modified, added or removed. These alterations must reflect how the algorithm-$\kappa\kappa$ works. We will rename the LPT variables as $\kappa\kappa$ variables.

- The `for` loop in row 12 involves only the jobs after the R-th job. Hence, the constraints in Optimization Model 5.2 that encoded the full LPT logic:

$$q_m \left(p_j + \sum_{i=1}^{j-1} z^{\kappa\kappa}_{m,i} \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} z^{\kappa\kappa}_{\tilde{m},i} \right) \leq B(1 - x^{\kappa\kappa}_{m,j}) \quad (\forall m, \tilde{m}, \forall j = 1, \ldots, J)$$

  must be modified as:

$$q_m \left(p_j + \sum_{i=1}^{j-1} z^{\kappa\kappa}_{m,i} \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} z^{\kappa\kappa}_{\tilde{m},i} \right) \leq B(1 - x^{\kappa\kappa}_{m,j}) \quad (\forall m, \tilde{m}, \quad \forall j = R+1, \ldots, J)$$

- We need to encode the logic that selects the best partial schedule for the longest R jobs. Note that the whole schedule that will be selected is encoded into the $z^{\kappa\kappa}$ variables. Hence, we need to add constraints to the first part of this schedule that reflect the logic of the `for` loop at row 4:

$$\max_{m=1,\ldots,M} \left(q_m \sum_{j=1,\ldots,R} z^{\kappa\kappa}_{m,j} \right) \leq \max_{m=1,\ldots,M} \left(q_m \sum_{\tilde{\mathcal{S}}^{-1}(m)} p_j \right) \quad (\forall \text{ schedule } \tilde{\mathcal{S}} \text{ of } R \text{ jobs})$$

  The quantifier "$\forall$ schedule $\tilde{\mathcal{S}}$ of R jobs" means that the constraint must be repeated for every possible schedule of the longest R jobs on the M machines. In other words, we are adding M^R constraints, and each of them should be linearized as it contains maximum operators.

Hence, the model for the algorithm-$\kappa\kappa$ is:

Optimization Model 5.3: Algorithm-$\kappa\kappa$

$$\max \quad C^{\kappa\kappa}$$

$$\text{s.t.} \quad C^{\kappa\kappa} = \max_m \left(q_m \sum_j x^{\kappa\kappa}_{m,j} p_j \right)$$

$$\max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right) = 1$$

$$\sum_m x^{\kappa\kappa}_{m,j} = 1 \qquad (\forall j)$$

$$\sum_m x^{*}_{m,j} = 1 \qquad (\forall j)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x^{\kappa\kappa}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{\kappa\kappa}_{\tilde{m},i} p_i \right) \leq B(1 - x^{\kappa\kappa}_{m,j})$$

$$(\forall m, \tilde{m}, \quad \forall j = R+1, \ldots, J)$$

$$\max_m \left(q_m \sum_{j=1,\ldots,R} x^{\kappa\kappa}_{m,j} p_j \right) \leq \max_m \left(q_m \sum_{\tilde{\mathcal{S}}^{-1}(m)} p_j \right) \quad (\forall \text{ schedule } \tilde{\mathcal{S}} \text{ of } R \text{ jobs})$$

$$1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0$$

$$1 = q_1 \leq q_2 \leq \cdots \leq q_M$$

We choose M and R as Koulamas and Kyparisis [2009] did: $M = 2$ and $R = 3$. We get:

L	$\frac{C^{\kappa\kappa}}{C^*}$	$1 + \frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.2247	1.2000	1.2247
3	1.2247	1.1429	1.2247

Table 5.2. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using algorithm-$\kappa\kappa$ as `H` and varying L.

Note that in the pathological case $R = 3$, $M = 2$ and $L = 1$, it holds that $J = LM = 2 < R$. In this case, the pure algorithm-$\kappa\kappa$ is reduced to brute-force.

An instance for $L = 2$ that achieves the approximation ratio is:

$$\mathcal{I}^{\kappa\kappa_{R=3}} = (\{q_1 = 1, q_2 = 1.2247\}, \{p_1 = 0.5918, p_2 = p_3 = p_4 = 0.4082\})$$

As done with LPT, we can deduce an algebraic form of $\mathcal{I}^{\kappa\kappa_{R=3}}$ by solving the following

system of equations:

$$\begin{cases} q_1(p_2+p_3+p_4) = q_2(p_1+p_2) \\ q_1(p_1+p_2) = q_2(p_3+p_4) = 1 \\ p_2 = p_3 = p_4 \\ q_1 = 1 \end{cases}$$

Giving:

$$\mathcal{I}^{\kappa\kappa_{R=3}} = \left(\left\{ q_1 = 1, q_2 = \sqrt{\frac{3}{2}} \right\}, \left\{ p_1 = 1 - \frac{1}{\sqrt{6}}, p_2 = p_3 = p_4 = \frac{1}{\sqrt{6}} \right\} \right)$$

Therefore, the Meta-Algorithm, fixing $L = 2$ and using algorithm-$\kappa\kappa$ as `H`, has an approximation ratio of $\sqrt{\frac{3}{2}} \approx 1.2247$ for the problem $Q2||C_{max}$. This is actually the same value found in Koulamas and Kyparisis [2009] for the "full" algorithm-$\kappa\kappa$.

In the same work by Koulamas and Kyparisis [2009], two modification are suggested: take $R = 4$ or $R = 5$. They provide only the following bounds: $1.167 \leq \rho_{R=4} \leq 1.2$ and $1.143 \leq \rho_{R=5} \leq 1.167$, as obtaining the actual approximation ratios would require significant modification to the proofs. With our model, we simply need to set a different value of R and let the solver do the rest. We get $\rho_{R=4} = 1.1861$ and $\rho_{R=5} = 1.1583$.

The two instances are:

$$\mathcal{I}^{\kappa\kappa_{R=4}} = (\{q_1 = 1, q_2 = 1.6861, \}, \{p_1 = 0.4069, p_2 = p_3 = p_4 = 0.2965, \})$$

$$\mathcal{I}^{\kappa\kappa_{R=5}} = (\{q_1 = 1, q_2 = 2.1583, \}, \{p_1 = 0.3050, p_2 = \cdots = p_6 = 0.2317\})$$

What about successive values of R? Is there a closed formula for $\mathcal{I}^{\kappa\kappa_R}$?

To answer the first question, we could simply re-run the model with other values of R. Alas, the second question cannot be directly answered by the model based approach, as R must be fixed in each run of the model. However, we can try to deduce a formula from the data. Comparing the instances $\mathcal{I}^{\kappa\kappa_{R=3}}, \mathcal{I}^{\kappa\kappa_{R=4}}, \mathcal{I}^{\kappa\kappa_{R=5}}$, we can "naturally" generalize the system of equations that holds for $\mathcal{I}^{\kappa\kappa_{R=3}}$ to:

$$\begin{cases} q_1(p_2+\cdots+p_J) = q_2(p_1+p_2) \\ q_1(p_1+p_2+\cdots+p_{J-2}) = q_2(p_{J-1}+p_J) = 1 \\ p_2 = \cdots = p_J \\ q_1 = 1 \end{cases}$$

That simplifies to:

$$\begin{cases} p_1 + (R-2)p_2 = 1 \\ 2p_2 q_2 = 1 \\ Rp_2 = (p_1 + p_2)q_2 \end{cases}$$

Hence, we may conjecture that the worst instance for a generic R is:

$$\begin{cases} p_1 & = \frac{\left(1-R+\sqrt{R^2+2R+9}\right)\left(3-R+\sqrt{R^2+2R+9}\right)}{8R} \\ p_j & = \frac{3-R+\sqrt{R^2+2R+9}}{4R} \quad \forall j = 2, \ldots, J \\ q_1 & = 1 \\ q_2 & = \frac{1}{4}\left(R-3+\sqrt{R^2+2R+9}\right) \\ \implies \rho_R^{\kappa\kappa} & = \frac{1}{4}\left(3-R+\sqrt{R^2+2R+9}\right) \end{cases} \tag{5.2}$$

The asymptotic behavior of $\rho_R^{\kappa\kappa}$, shown in Figure 5.5, is coherent with the algorithm-$\kappa\kappa$. As R grows, more and more jobs are scheduled optimally. In the limit $R \to \infty$, all jobs will be scheduled optimally, hence the approximation ratio should approach 1. Indeed, the conjecture satisfies this limit:

$$\lim_{R\to\infty} \rho_R^{\kappa\kappa} = \lim_{R\to\infty} \frac{1}{4}\left(3-R+\sqrt{R^2+2R+9}\right) = 1$$

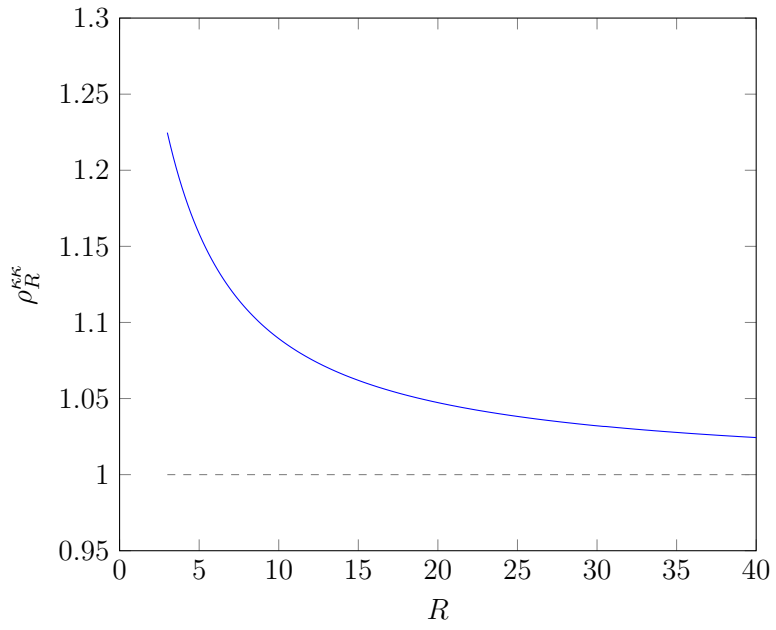

Figure 5.5. Asymptotic behavior of $\rho_R^{\kappa\kappa}$.

It should be noted that the algorithm-$\kappa\kappa$ approach (brute-force on the largest R jobs) cannot be extended indefinitely: as R increases, the brute-force phase becomes quickly computationally infeasible (it is an $\mathcal{NP}$ problem).

5.3 Local search V1 Algorithm

Inspired by Della Croce et al. [2019], we introduce a lightweight modification to LPT by appending to it a simple local search procedure. This procedure, that we will call from now on algorithm-σ_1, will modify the LPT schedule and find, in the neighborhood of the LPT schedule, a better schedule. We need to formally define what the neighborhood of a schedule $\mathcal{S}$ is. As intuitively as possible, a schedule $\tilde{\mathcal{S}}$ is in the neighborhood of the schedule $\mathcal{S}$ if one can obtain $\tilde{\mathcal{S}}$ from $\mathcal{S}$ by pairwise exchanging any two jobs.

Procedure 5.3: Algorithm-σ_1 for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The algorithm-σ_1 schedule $\mathcal{S}$ and its value C

```
// Apply LPT. Inside LPT the jobs will be sorted
```
$\mathcal{S}, C \leftarrow \texttt{LPT}\left(\{q_m\}, \{p_j\}\right)$;
$T_m \leftarrow \sum_{j \in \mathcal{S}^{-1}(m)} q_m p_j \quad \forall m = 1, \dots, M$;
```
// Now search for the best one-job-to-one-job swap
// The variable C^σ1 will hold the value of the schedule found
   after the local search step V1
```
$C^{\sigma_1} \leftarrow C$;
$i^{\sigma_1} \leftarrow -1$;
$k^{\sigma_1} \leftarrow -1$;
```
// There are smarter ways to implement this search.  This is
   the simplest way
// Cycle through each pair of jobs on different machines
```
for $i = 1, \dots, J$ **do**
  for $k = 1, \dots, J$ **do**
    // Get which machines jobs i and j are assigned to
    $m_i \leftarrow \mathcal{S}(i)$;
    $m_k \leftarrow \mathcal{S}(k)$;
    if $m_i \neq m_k$ **then**
      $c \leftarrow \max\{q_1 T_1, \dots, q_{m_i}(T_{m_i} - p_i + p_k), \dots, q_{m_k}(T_{m_k} + p_i - p_k), \dots, q_M T_M\}$;
      if $c < C^{\sigma_1}$ **then**
        $i^{\sigma_1} \leftarrow i$;
        $k^{\sigma_1} \leftarrow k$;
        $C^{\sigma_1} \leftarrow c$;
      end
    end
  end
end
```
// Update the schedule S, if needed, by exchanging job i^σ1
   with job k^σ1.
```
if $i^{\sigma_1} \neq -1$ **then**
  $m_i \leftarrow \mathcal{S}(i)$;
  $m_k \leftarrow \mathcal{S}(k)$;
  $\mathcal{S}(i) \leftarrow m_k$;
  $\mathcal{S}(k) \leftarrow m_i$;
end

For the sake of simplicity, we introduce the model of the algorithm-σ_1 only for the problem $Q2||C_{max}$, i.e., fixing the number of machines to 2.

As done with the algorithm-$\kappa\kappa$, we use Optimization Model 5.2 as a backbone model, introducing the following modifications:

- The algorithm-σ_1 has two phases. The first is simply LPT, hence all the constraints in Optimization Model 5.2 are valid. The second phase is the single local search step, where a new schedule is developed, whose value is encoded in the new variable C. Then the best between these two schedules is returned. To reflect this duality, we propose the following structure:

$$\begin{aligned} \max \quad & C^{\sigma_1} \\ \text{s.t.} \quad & C^{LPT} = \max_m \left(q_m \sum_j x_{m,j}^{LPT} p_j \right) \\ & C = \text{ schedule value after local search step} \\ & C^{\sigma_1} \leq C^{LPT} \\ & C^{\sigma_1} \leq C \\ & \vdots \end{aligned}$$

  In this way, C^{LPT} and the whole LPT schedule are not affected by the local search step "until the objective function".

- How to constrain the new variable C? Suppose that $M = 2$:

$$\begin{aligned} C \leq \max \{ & q_1 \left(p_k - p_i + \sum_j x_{1,j}^{LPT} p_j \right), \\ & q_2 \left(p_i - p_k + \sum_j x_{1,j}^{LPT} p_j \right) \\ & \} + B(2 - x_{1,i}^{LPT} - x_{2,k}^{LPT}) \qquad (\forall i, k = 1, \ldots, J \quad i \neq k) \end{aligned}$$

  In this way, when job i is assigned by LPT to machine 1 and job k is assigned by LPT to machine 2, the term with the big constant B cancels. Hence, the variable C is truly constrained, by varying i and k among $1, \ldots, J$. We remind that all maximum constraints are handled as presented in Section 5.1.3.

Therefore, the model of algorithm-σ_1 for $Q2||C_{max}$ is:

Optimization Model 5.4: Algorithm-σ_1 for $Q2||C_{max}$

$$\max \quad C^{\sigma_1}$$

$$\text{s.t.} \quad C^{LPT} = \max_m \left(q_m \sum_j x_{m,j}^{LPT} p_j \right)$$

$$\max_m \left(q_m \sum_j x_{m,j}^{*} p_j \right) = 1$$

$$\sum_m x_{m,j}^{LPT} = 1 \qquad (\forall j)$$

$$\sum_m x_{m,j}^{*} = 1 \qquad (\forall j)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x_{m,i}^{LPT} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x_{\tilde{m},i}^{LPT} p_i \right) \leq B(1 - x_{m,j}^{LPT}) \quad (\forall m, \tilde{m}, j)$$

$$C \leq \max \left\{ q_1 \left(p_k - p_i + \sum_j x_{1,j}^{LPT} p_j \right), q_2 \left(p_i - p_k + \sum_j x_{2,j}^{LPT} p_j \right) \right\} +$$
$$+ B(2 - x_{1,i}^{LPT} - x_{2,k}^{LPT}) \qquad (\forall i, k = 1, \ldots, J \quad i \neq k)$$

$$C^{\sigma_1} \leq C^{LPT}$$

$$C^{\sigma_1} \leq C$$

$$1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0$$

$$1 = q_1 \leq q_2 \leq \cdots \leq q_M$$

Solving the model, we get the same results as the LPT case:

L	$\frac{C^{\sigma_1}}{C^*}$	$1 + \frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.2808	1.2000	1.2808
3	1.2808	1.1429	1.2808

Table 5.3. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using algorithm-σ_1 as `H` and varying L.

Indeed, the Meta-Algorithm, fixing $L = 2$ and algorithm-σ_1 as `H`, has an approximation ratio of 1.2808 for the problem $Q2||C_{max}$. Its approximation ratio is the same as the LPT algorithm. Also, the corresponding worst instance is the same:

$$\mathcal{I}^{\sigma_1} = (\{q_1 = 1, q_2 = 1.2808\}, \{p_1 = 0.7808, p_2 = p_3 = 0.5000\}) = \mathcal{I}^{LPT}$$

The above result indicates that the local search step could not improve the LPT schedule. So, even a variation of algorithm-σ_1, where a sequence of local search steps is performed, would have the same approximation ratio. We need to improve the quality of the local search step by enlarging its neighborhood.

However, the algorithm-σ_1 is an improvement over LPT for the problem $P2||C_{max}$. Indeed, we can run the model fixing both speed factors to 1, reducing the $Q2||C_{max}$ problem to the $P2||C_{max}$ problem. Using Proposition 4.0.5 as bound, we get:

L	$\frac{C^{\sigma_1}}{C^*}$	$1 + \frac{M-1}{M(L+1)}$	ρ
1	1.0000	1.2500	1.2500
2	1.0000	1.1667	1.1667
3	1.1250	1.1250	1.1250
4	1.1250	1.1000	1.1250

Table 5.4. Table of model optima, theoretical bounds (using Proposition 4.0.5) and approximation ratios of the Meta-Algorithm using algorithm-σ_1 as `H` and varying L for the $P2||C_{max}$ problem.

Hence, fixing $L = 3$, the algorithm-σ_1 is an approximation algorithm for the problem $P2||C_{max}$ with an approximation ratio of $1.1250 \approx \frac{9}{8}$. The LPT algorithm has an approximation ratio of $\frac{7}{6} \approx 1.1667$ for the problem $P2||C_{max}$.

5.4 Local search V2 Algorithm

As the algorithm-σ_1 has the same approximation ratio as LPT for $Q2||C_{max}$, a wider local search neighborhood must be considered. We enlarge the old neighborhood to the new neighborhood by including also other two kinds of swapping. Indeed, in algorithm-σ_1, only one-to-one job swappings were considered. In this algorithm, introduced in Della Croce et al. [2019], also two-to-one (and one-to-two) job swappings are considered.

Given a schedule $\mathcal{S}$, any schedule that is "reachable" by performing only one of the following actions is considered in the neighborhood of $\mathcal{S}$:

- Swap one job on a machine with one job on another machine (also done in algorithm-σ_1).
- Swap two jobs on a machine with one job on another machine.
- Swap one job on a machine with two jobs on another machine.

We will call this algorithm algorithm-σ_2.

As the algorithm-σ_2 is very similar to algorithm-σ_1 (only the definition of the neighborhood is changed), the optimization model for algorithm-σ_2 is based on the one for algorithm-σ_1. For the sake of simplicity, we fix $M = 2$; as such the model is:

Optimization Model 5.5: Algorithm-σ_2

$$\max \quad C^{\sigma_2}$$

$$\text{s.t.} \quad C^{LPT} = \max_m \left(q_m \sum_j x_{m,j}^{LPT} p_j \right)$$

$$\max_m \left(q_m \sum_j x_{m,j}^{*} p_j \right) = 1$$

$$\sum_m x_{m,j}^{LPT} = 1 \qquad (\forall j)$$

$$\sum_m x_{m,j}^{*} = 1 \qquad (\forall j)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x_{m,i}^{LPT} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x_{\tilde{m},i}^{LPT} p_i \right) \leq B(1 - x_{m,j}^{LPT}) \quad (\forall m, \tilde{m}, j)$$

$$C \leq \max \left\{ q_1 \left(p_k - p_i + \sum_j x_{1,j}^{LPT} p_j \right), q_2 \left(p_i - p_k + \sum_j x_{2,j}^{LPT} p_j \right) \right\} + B(2 - x_{1,i}^{LPT} - x_{2,k}^{LPT}) \qquad (\forall i, k = 1, \ldots, J \quad i \neq k)$$

$$C \leq \max \left\{ q_1 \left(p_k - p_i - p_l + \sum_j x_{1,j}^{LPT} p_j \right), q_2 \left(p_i + p_l - p_k + \sum_j x_{2,j}^{LPT} p_j \right) \right\} + B(3 - x_{1,i}^{LPT} - x_{2,k}^{LPT} - x_{1,l}^{LPT}) \qquad (\forall i, k, l = 1, \ldots, J \quad i \neq k, i \neq l, k \neq l)$$

$$C \leq \max \left\{ q_1 \left(p_k + p_l - p_i + \sum_j x_{1,j}^{LPT} p_j \right), q_2 \left(p_i - p_k - p_l + \sum_j x_{2,j}^{LPT} p_j \right) \right\} + B(3 - x_{1,i}^{LPT} - x_{2,k}^{LPT} - x_{2,l}^{LPT}) \qquad (\forall i, k, l = 1, \ldots, J \quad i \neq k, i \neq l, k \neq l)$$

$$C^{\sigma_2} \leq C^{LPT}$$

$$C^{\sigma_2} \leq C$$

$$1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0$$

$$1 = q_1 \leq q_2 \leq \cdots \leq q_M$$

Solving the model we get:

L	$\frac{C^{\sigma_2}}{C^*}$	$1+\frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.1805	1.2000	1.2000
3	1.1805	1.1429	1.1805

Table 5.5. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using algorithm-σ_2 as `H` and varying L.

The instance corresponding to $L = 3$ is:

$$\mathcal{I}^{\sigma_2} = (\{q_1 = 1, q_2 = 1.1805\}, \{p_1 = 0.8471, p_2 = p_3 = p_4 = 0.3333\})$$

As done with LPT and algorithm-$\kappa\kappa$, we can get the following algebraic form:

$$\mathcal{I}^{\sigma_2} = \left(\left\{q_1 = 1, q_2 = \frac{6}{-1+\sqrt{37}}\right\}, \left\{p_1 = \frac{-1+\sqrt{37}}{6}, p_2 = p_3 = p_4 = \frac{1}{3}\right\}\right)$$

Hence, the approximation ratio of the meta-algorithm applied with the algorithm-σ_2 and $L = 3$ to the problem $Q2||C_{max}$ is $\frac{6}{-1+\sqrt{37}} \approx 1.1805$.

It is interesting to note that the new expanded neighborhood is large enough to improve on the LPT approximation ratio. However, the LPT algorithm applied to this instance produces the same schedule as algorithm-σ_2. It means that, as in algorithm-σ_1, the worst instance of algorithm-σ_2 does not benefit from the local search step. Hence, even a variation of algorithm-σ_2, where a sequence of local search steps is performed, would have the same approximation ratio as algorithm-σ_2.

As done with algorithm-σ_1, we can re-run Optimization Model 5.5, fixing $q_1 = q_2 = 1$, hence reducing again the $Q2||C_{max}$ problem to the $P2||C_{max}$ problem. Using Proposition 4.0.5 as bound, we get:

L	$\frac{C^{\sigma_2}}{C^*}$	$1+\frac{M-1}{M(L+1)}$	ρ
1	1.0000	1.2500	1.2500
2	1.0000	1.1667	1.1667
3	1.0000	1.1250	1.1250
4	1.0833	1.1000	1.1000
5	1.0833	1.0833	1.0833
6	1.0833	1.0714	1.0833

Table 5.6. Table of model optima, theoretical bounds (using Proposition 4.0.5) and approximation ratios of the Meta-Algorithm using algorithm-σ_2 as `H` and varying L for the $P2||C_{max}$ problem.

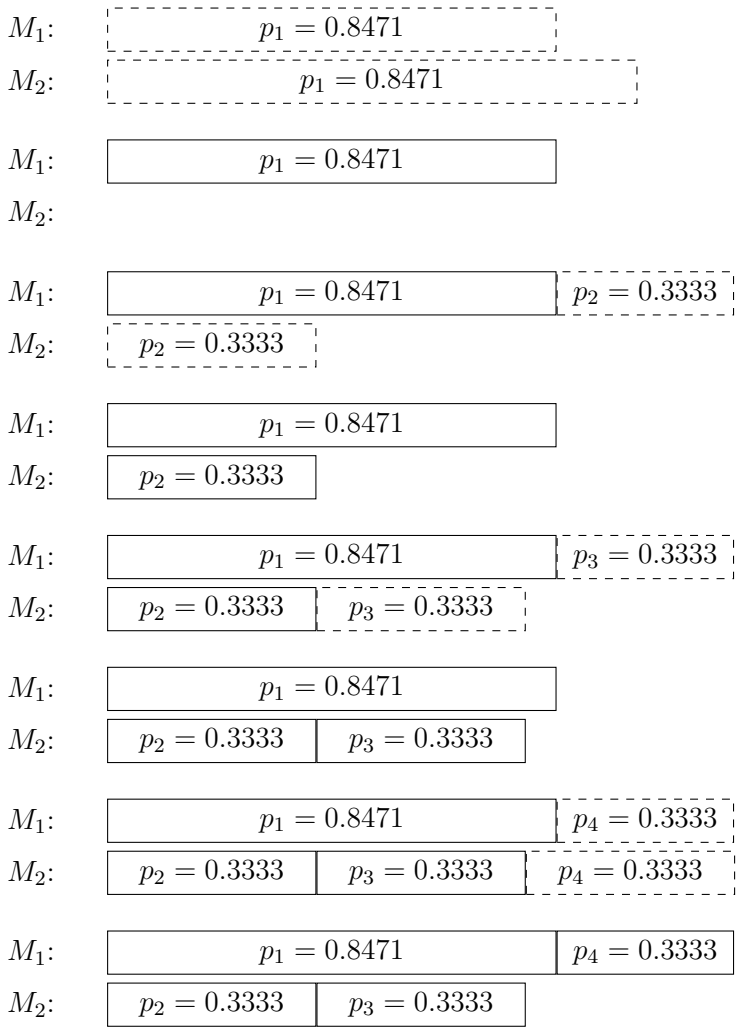

Figure 5.6. The LPT algorithm building the schedule of the instance $\mathcal{I}^{\sigma_2}$. Remember that eventual ties are broken at random, and we must always check for the worst case.

Fixing $L = 5$, the algorithm-σ_2 is an approximation algorithm for the problem $P2||C_{max}$ with an approximation ratio of $1.0833 \approx \frac{13}{12}$. This is a complete confirmation of the result found by Della Croce et al. [2019].

5.5 Alternative start V1 Algorithm

We could keep improving the approximation ratio by enlarging the neighborhood considered in the local search step, but this entails much longer run-times. Instead, somewhat inspired by Koulamas and Kyparisis [2009], we introduce the algorithm-α_1. This algorithm considers two schedules: the one generated by LPT and a "revised" one.

During the first step of the LPT algorithm, the first job (the longest one) will always be assigned to the fastest machine. By doing so, the LPT algorithm, the algorithm-σ_1 and the algorithm-σ_2 all misplace the first job in their worst case. Indeed, in all cases, the corresponding optimal schedules assign the longest job to the slowest machine.

As such, we propose to consider another schedule, where the first job (the longest one) is always fixed on the slowest machine. Then the LPT rule is followed onwards.

Finally, algorithm-α_1 returns the best schedule between the pure LPT one and the "revised" one.

Procedure 5.4: Algorithm-α_1 for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The algorithm-α_1 schedule $\mathcal{S}$ and its value C
`// First Step: order the jobs`
Sort the jobs in decreasing order, i.e.
$i \leq j \implies p_i \geq p_j, \quad \forall i,j \in \{1,\dots,J\}$;
`// Second Step: apply LPT`
$\mathcal{S}^{LPT}, C^{LPT} \leftarrow \texttt{LPT}(\{q_m\},\{p_j\})$;
`// Third Step: consider a start-corrected schedule and then apply LPT`
$T_m \leftarrow 0 \quad \forall m = 1,\dots,M$;
$\mathcal{S}_m \leftarrow \{\} \quad \forall m = 1,\dots,M$;
`// Assign job 1 to the slowest machine`
$T_M \leftarrow p_1$;
$\mathcal{S}_M \leftarrow \{1\}$;
`// From job 2, start with LPT logic assignments`
for $j = 2,\dots,J$ **do**
  $m \leftarrow \underset{\tilde{m}=1,\dots,M}{\operatorname{argmin}} q_{\tilde{m}}(T_{\tilde{m}} + p_j)$;
  $T_m \leftarrow T_m + p_j$;
  $S_m \leftarrow S_m \cup \{j\}$;
end
$C = \max_{m=1,\dots,M} q_m T_m$;

As the algorithm-α_1 is basically two full LPT applications, the model will encode two schedules: $x^{LPT}_{m,j}$ and $x^{REV}_{m,j}$, standing for revised schedule. The model will obviously

encode also the (single) optimal schedule $x^*_{m,j}$.

Optimization Model 5.2 is used as backbone and it is modified as following.

- As done in algorithm-σ_1 and σ_2, we introduce the following modification to account for returning the best of two schedules:

$$\begin{aligned} \max \quad & C^{\alpha_1} \\ \text{s.t.} \quad & C^{LPT} = \max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right) \\ & C^{REV} = \max_m \left(q_m \sum_j x^{REV}_{m,j} p_j \right) \\ & C^{\sigma_1} \leq C^{LPT} \\ & C^{\sigma_1} \leq C^{REV} \\ & \vdots \end{aligned}$$

  Where the new variable C^{REV} will encode the value of the revised schedule.

- The constraints for $x^{REV}_{m,j}$ are almost the same as for $x^{LPT}_{m,j}$, but the first job is fixed on the second machine:

$$x^{REV}_{M,1} = 1$$

Hence, the model for algorithm-α_1 is:

Optimization Model 5.6: Algorithm-α_1

$$
\begin{aligned}
\max \quad & C^{\alpha_1} \\
\text{s.t.} \quad & C^{\alpha_1} \leq C^{LPT} \\
& C^{\alpha_1} \leq C^{REV} \\
& C^{LPT} = \max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right) \\
& C^{REV} = \max_m \left(q_m \sum_j x^{REV}_{m,j} p_j \right) \\
& \max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right) = 1 \\
& \sum_m x^{LPT}_{m,j} = 1 \qquad (\forall j) \\
& \sum_m x^{REV}_{m,j} = 1 \qquad (\forall j) \\
& \sum_m x^{*}_{m,j} = 1 \qquad (\forall j) \\
& q_m \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{\tilde{m},i} p_i \right) \leq B(1 - x^{LPT}_{m,j}) \quad (\forall m, \tilde{m}, j) \\
& q_m \left(p_j + \sum_{i=1}^{j-1} x^{REV}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{REV}_{\tilde{m},i} p_i \right) \leq B(1 - x^{REV}_{m,j}) \\
& \qquad\qquad (\forall m, \tilde{m}, \quad \forall j = 2, \ldots, J) \\
& x^{REV}_{M,1} = 1 \\
& 1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0 \\
& 1 = q_1 \leq q_2 \leq \cdots \leq q_M
\end{aligned}
$$

For simplicity, fixing $M = 2$ we get:

L	$\frac{C^{\alpha_1}}{C^*}$	$1 + \frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.2071	1.2000	1.2071
3	1.2071	1.1429	1.2071

Table 5.7. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using algorithm-α_1 as `H` and varying L.

The instance with $L = 2$ is:

$$\mathcal{I}^{\alpha_1} = (\{q_1 = 1, q_2 = 1.4142\}, \{p_1 = p_2 = 0.5000, p_3 = p_4 = 0.3536\})$$

Its algebraic form is:

$$\mathcal{I}^{\alpha_1} = \left(\left\{q_1 = 1, q_2 = \sqrt{2}\right\}, \left\{p_1 = p_2 = \frac{1}{2}, p_3 = p_4 = \frac{1}{2\sqrt{2}}\right\}\right)$$

Hence, fixing $L = 2$ and algorithm-α_1 as `H`, the Meta-Algorithm has an approximation ratio of $\frac{1}{2} + \frac{1}{\sqrt{2}} \approx 1.2071$ for the problem $Q2||C_{max}$. The corresponding instance is shown in Figure 5.7.

Algorithm-α_1 was inspired by algorithm-$\kappa\kappa$. There are two main differences. Firstly, α_1 does not check for every possible starting schedule, but only for two of them, in a fixed manner. Hence, α_1 should be faster than $\kappa\kappa$. Secondly, α_1 decides which schedule to return after placing all LM jobs. On the contrary, $\kappa\kappa$ confronts partial schedules containing only R jobs. Moreover, the worst instance of algorithm-$\kappa\kappa$ can be solved to optimality with LPT. Hence, algorithm α_1 "mixes together" LPT and some important starting schedules from algorithm-$\kappa\kappa$.

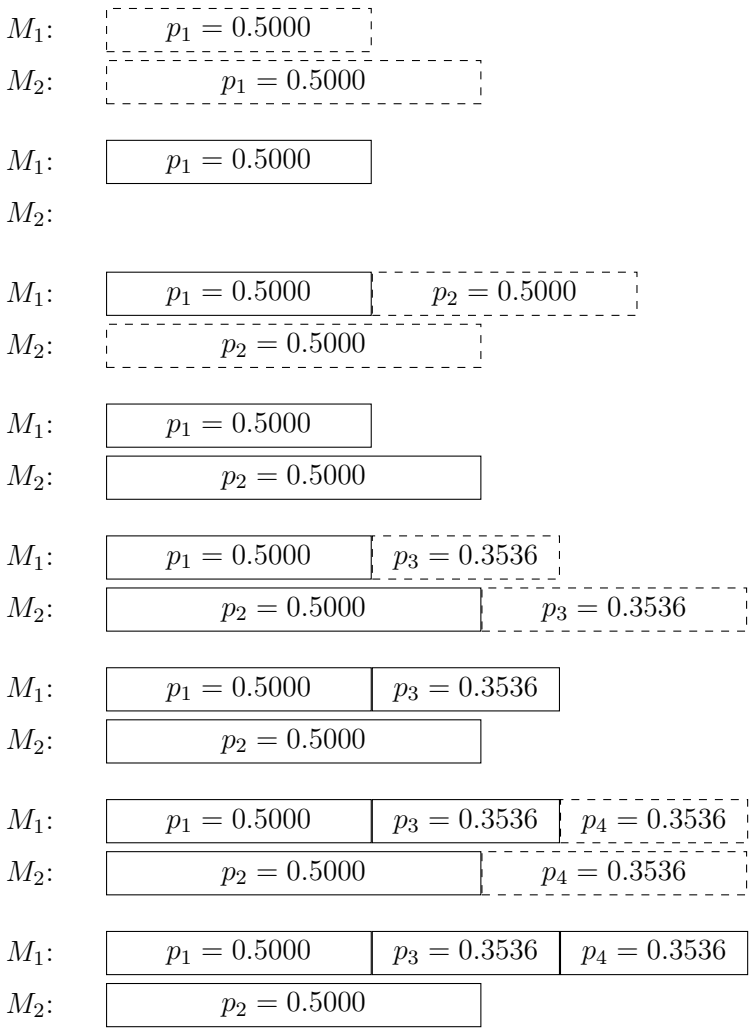

Figure 5.7. The LPT algorithm building the schedule of the instance $\mathcal{I}^{\alpha_1}$.

5.6 Alternative start V2 Algorithm

This algorithm, called algorithm-α_2, improves upon the algorithm-α_1 by considering one more revised schedule. In particular, this new revised schedule is obtained from the optimal schedule of the algorithm-α_1 worst case.

Procedure 5.5: Algorithm-α_2 for $QM||C_{max}$

Input: A sequence of J jobs lengths $(p_j)_{j=1,\dots,J}$
A sequence of M machines speed factors $(q_m)_{m=1,\dots,M}$
Output: The algorithm-α_2 schedule $\mathcal{S}$ and its value C
`// First Step: order the jobs`
Sort the jobs in decreasing order, i.e.
$i \leq j \implies p_i \geq p_j, \quad \forall i,j \in \{1,\dots,J\}$;
`// Second Step: apply LPT and algorithm-`α_1
$\mathcal{S}^{LPT}, C^{LPT} \leftarrow$ `LPT` $(\{q_m\},\{p_j\})$;
$\mathcal{S}^{\alpha_1}, C^{\alpha_1} \leftarrow$ `Alpha1` $(\{q_m\},\{p_j\})$;
`// Third Step: consider a new start-corrected schedule and then apply LPT`
$T_m \leftarrow 0 \quad \forall m = 1,\dots,M$;
$\mathcal{S}_m \leftarrow \{\} \quad \forall m = 1,\dots,M$;
`// Assign jobs 1,2 to the fastest machine`
$T_1 \leftarrow p_1 + p_2$;
$\mathcal{S}_1 \leftarrow \{1,2\}$;
`// From job 3, start with LPT logic assignments`
for $j = 3,\dots,J$ **do**
  $m \leftarrow \underset{\tilde{m}=1,\dots,M}{\operatorname{argmin}} q_{\tilde{m}}(T_{\tilde{m}} + p_j)$;
  $T_m \leftarrow T_m + p_j$;
  $S_m \leftarrow S_m \cup \{j\}$;
end
$C = \max_{m=1,\dots,M} q_m T_m$;

The optimization model of algorithm-α_2 uses as backbone the one of algorithm-α_1. We simply introduce a new schedule $x^{REV2}_{m,j}$ and constrain it with $x^{REV2}_{1,1} = x^{REV2}_{1,2} = 1$. The value of the schedule $x^{REV2}_{m,j}$ is represented by the variable C^{REV2}. Hence, the model is:

Optimization Model 5.7: Algorithm-α_2

$$\max \quad C^{\alpha_2}$$

$$\text{s.t.} \quad C^{\alpha_2} \leq C^{LPT}$$

$$C^{\alpha_2} \leq C^{REV}$$

$$C^{\alpha_2} \leq C^{REV2}$$

$$C^{LPT} = \max_m \left(q_m \sum_j x^{LPT}_{m,j} p_j \right)$$

$$C^{REV} = \max_m \left(q_m \sum_j x^{REV}_{m,j} p_j \right)$$

$$C^{REV2} = \max_m \left(q_m \sum_j x^{REV2}_{m,j} p_j \right)$$

$$\max_m \left(q_m \sum_j x^{*}_{m,j} p_j \right) = 1$$

$$\sum_m x^{LPT}_{m,j} = 1 \qquad (\forall j)$$

$$\sum_m x^{REV}_{m,j} = 1 \qquad (\forall j)$$

$$\sum_m x^{REV2}_{m,j} = 1 \qquad (\forall j)$$

$$\sum_m x^{*}_{m,j} = 1 \qquad (\forall j)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{LPT}_{\tilde{m},i} p_i \right) \leq B(1 - x^{LPT}_{m,j}) \quad (\forall m, \tilde{m}, j)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x^{REV}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{REV}_{\tilde{m},i} p_i \right) \leq B(1 - x^{REV}_{m,j}) \qquad (\forall m, \tilde{m}, \quad \forall j = 2, \ldots, J)$$

$$q_m \left(p_j + \sum_{i=1}^{j-1} x^{REV2}_{m,i} p_i \right) - q_{\tilde{m}} \left(p_j + \sum_{i=1}^{j-1} x^{REV2}_{\tilde{m},i} p_i \right) \leq B(1 - x^{REV2}_{m,j}) \qquad (\forall m, \tilde{m}, \quad \forall j = 3, \ldots, J)$$

$$x^{REV}_{M,1} = 1$$

$$x^{REV2}_{1,1} = 1$$

$$x^{REV2}_{1,2} = 1$$

$$1 \geq p_1 \geq p_2 \geq \cdots \geq p_J \geq 0$$

$$1 = q_1 \leq q_2 \leq \cdots \leq q_M$$

Solving the model with $M = 2$ we get:

L	$\frac{C^{\alpha_2}}{C^*}$	$1 + \frac{M-1}{LM+1}$	ρ
1	1.0000	1.3333	1.3333
2	1.1754	1.2000	1.2000
3	1.1754	1.1429	1.1754

Table 5.8. Table of model optima, theoretical bounds and approximation ratios of the Meta-Algorithm using algorithm-α_2 as `H` and varying L.

The instance for $L = 3$ is:

$$\mathcal{I}^{\alpha_2} = (\{q_1 = 1, q_2 = 2.3507\}, \{p_1 = 0.5000, p_2 = 0.4254, p_3 = p_4 = 0.2500\})$$

and its algebraic form:

$$\mathcal{I}^{\alpha_2} = \left(\left\{ q_1 = 1, q_2 = \frac{1 + 16\sqrt{11}}{23} \right\}, \left\{ p_1 = \frac{1}{2}, p_2 = \frac{23}{1 + 16\sqrt{11}}, p_3 = p_4 = \frac{1}{4} \right\} \right)$$

Hence, Meta-Algorithm applied with algorithm-α_2 and $L = 3$ has an approximation ratio of $\frac{1+16\sqrt{11}}{46} \approx 1.1754$ for the problem $Q2||C_{max}$.

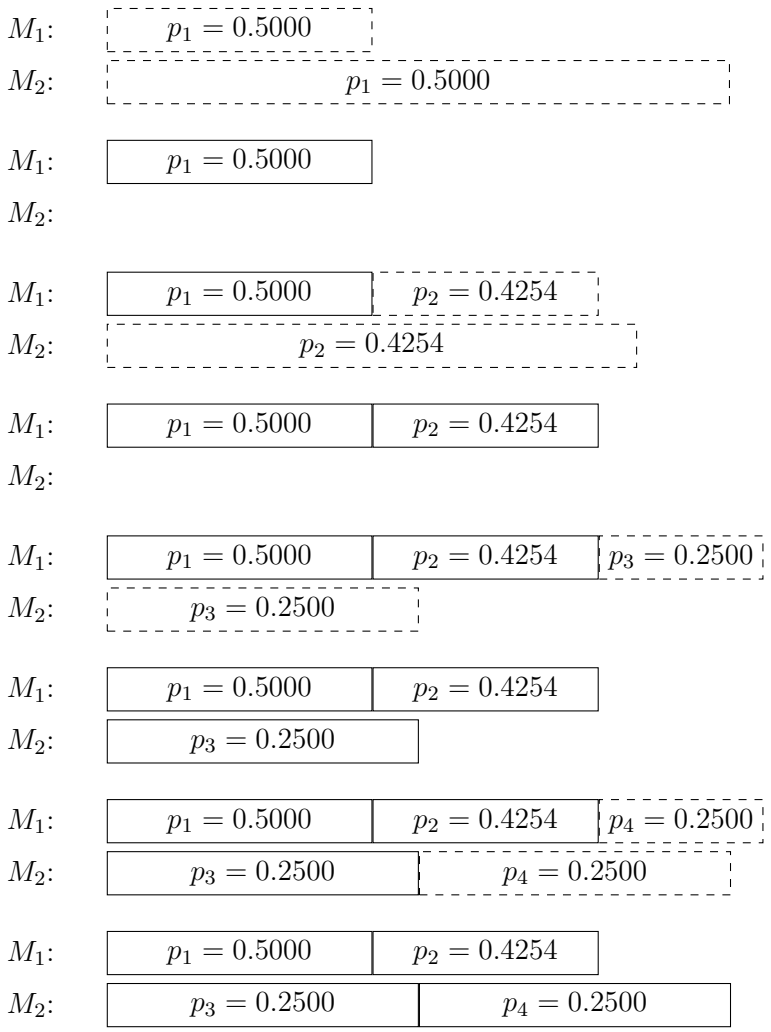

Figure 5.8. The LPT algorithm building the schedule of the instance $\mathcal{I}^{\alpha_2}$.

Chapter 6

Results

6.1 Approximation Results

Table 6.1 reports the approximation ratios of all the algorithms analyzed in the previous section, applied to the problem $Q2||C_{max}$. Recall that all results are valid concerning the framework of the Meta-Algorithm, where an algorithm H and an integer L must be chosen as parameters.

ρ for $Q2\|\|C_{max}$	$L=1$	$L=2$	$L=3$	$L=4$
H = LPT	1.3333	1.2808	1.2808	1.2808
H = $\kappa\kappa$ with $R=3$	1.3333	1.2247	1.2247	1.2247
H = $\kappa\kappa$ with $R=4$	1.3333	1.2000	1.1861	1.1861
H = $\kappa\kappa$ with $R=5$	1.3333	1.2000	1.1583	1.1583
H = σ_1	1.3333	1.2808	1.2808	1.2808
H = σ_2	1.3333	1.2000	1.1805	1.1805
H = α_1	1.3333	1.2071	1.2071	1.2071
H = α_2	1.3333	1.2000	1.1754	1.1754

Table 6.1. Recap approximation ratios for various algorithm applied to $Q2||C_{max}$

For each algorithm H, we choose the smallest L that results in the smallest approximation ratio. Hence, we propose as approximation algorithm the Meta-Algorithm with any of the following $(\mathtt{H}, L)$ pairs:

H	L	ρ	$\approx \rho$	q_2	Jobs Lengths
LPT	2	$\frac{1+\sqrt{17}}{4}$	1.2808	$\frac{1+\sqrt{17}}{4}$	$\left\{\frac{-1+\sqrt{17}}{4}, \frac{1}{2}, \frac{1}{2}\right\}$
$\kappa\kappa$ with $R=3$	2	$\sqrt{\frac{3}{2}}$	1.2247	$\sqrt{\frac{3}{2}}$	$\left\{1-\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}\right\}$
$\kappa\kappa$ with $R=4$	3	$\frac{-1+\sqrt{33}}{4}$	1.1861	$\frac{1+\sqrt{33}}{4}$	$\left\{\frac{9-\sqrt{33}}{8}, \frac{1-\sqrt{33}}{16}, \frac{1-\sqrt{33}}{16}, \frac{1-\sqrt{33}}{16}\right\}$
$\kappa\kappa$ with $R=5$	3	$\frac{-1+\sqrt{11}}{2}$	1.1583	$\frac{1+\sqrt{11}}{2}$	$\left\{\frac{13-3\sqrt{11}}{10}\right\} \cup \left\{\frac{-1+\sqrt{11}}{10}\right\} \times 5$
σ_1	2	$\frac{1+\sqrt{17}}{4}$	1.2808	$\frac{1+\sqrt{17}}{4}$	$\left\{\frac{-1+\sqrt{17}}{4}, \frac{1}{2}, \frac{1}{2}\right\}$
σ_2	3	$\frac{1+\sqrt{37}}{6}$	1.1805	$\frac{1+\sqrt{37}}{6}$	$\left\{\frac{-1+\sqrt{37}}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$
α_1	2	$\frac{1}{2}+\frac{1}{\sqrt{2}}$	1.2071	$\sqrt{2}$	$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}\right\}$
α_2	3	$\frac{1+16\sqrt{11}}{46}$	1.1754	$\frac{1+16\sqrt{11}}{23}$	$\left\{\frac{1}{2}, \frac{23}{1+16\sqrt{11}}, \frac{1}{4}, \frac{1}{4}\right\}$

Table 6.2. Proposed Approximation Algorithms for $Q2||C_{max}$ with their approximation ratios ρ and the instance achieving it.

We also run the models for the $P2||C_{max}$ problem by fixing the machines speeds to 1. Table 6.3 shows the corresponding approximation ratios, using Proposition 4.0.5 as bound.

H	L	ρ	Jobs Lengths
LPT	3	$\frac{7}{6}$	$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$
$\kappa\kappa$ with $R=3$	3	$\frac{7}{6}$	$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$
$\kappa\kappa$ with $R=4$	3	$\frac{7}{6}$	$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$
$\kappa\kappa$ with $R=5$	3	$\frac{8}{7}$	$\left\{\frac{3}{7}, \frac{3}{7}, \frac{2}{7}, \frac{2}{7}, \frac{2}{7}, \frac{2}{7}\right\}$
σ_1	3	$\frac{9}{8}$	$\left\{\frac{5}{8}, \frac{3}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\}$
σ_2	5	$\frac{13}{12}$	$\left\{\frac{7}{12}, \frac{5}{12}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right\}$
α_1	3	$\frac{7}{6}$	$\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$
α_2	3	$\frac{9}{8}$	$\left\{\frac{5}{8}, \frac{1}{2}, \frac{3}{8}, \frac{1}{4}, \frac{1}{4}\right\}$

Table 6.3. Proposed Approximation Algorithms for $P2||C_{max}$ with their approximation ratios ρ and the instance achieving it.

6.2 Heuristic Results

Up to now, all the results in this work are about algorithms performance guarantees in the worst case. We now briefly focus on the "average" performance of our algorithms and consider them as heuristics.

To this extent, we arbitrarily pick the following distribution:

$$M = 8 \qquad J = 2048 \qquad p_j \sim q_m \sim U\,[10,11,12,\ldots,100] \quad i.i.d \tag{6.1}$$

The chosen distribution is formed by instances with numerous jobs and machines, hence it is better suited for a practical average performance comparison. In fact, smaller instances, for example the instances in Table 6.2, can be solved optimally by direct inspection.

The heuristics are compared to the lower bound provided by Proposition 4.0.1, using the following Monte-Carlo procedure with 1024 samples:

1. Generate an instance $\mathcal{I}$ according to the chosen distribution.
2. Compute the "instance optimality constant": $IOC(\mathcal{I}) := \frac{\sum_{j=1,\dots,J} p_j}{\mathcal{Q}}$.
3. Run the algorithm H on instance $\mathcal{I}$ and get the schedule value $C^H(\mathcal{I})$
4. Compute $\frac{C^H(\mathcal{I})}{IOC(\mathcal{I})}$

We use the above procedure to estimate the average-case performance for the following 12 different algorithms:

- 5 approximation algorithms proposed in Table 6.2: (LPT,2), (σ_1,2), (σ_2,3), (α_1,2), (α_2,3). For example, as prescribed by the Meta-Algorithm, the algorithm (LPT,2) selects the biggest $LM = 2 \cdot 8 = 16$ jobs among all 2048 jobs. Then it applies the LPT algorithm (sorting included) only to the biggest ones. Finally, it completes the schedule using list scheduling with the remaining $2048 - 16 = 2032$ jobs.
- 5 heuristic algorithms: Full-LPT, Full-σ_1, Full-σ_2, Full-α_1, Full-α_2. Simply, the algorithm is applied to the whole instance. For example, the Full-LPT algorithm sorts all 2048 jobs and then applies the list scheduling logic to all 2048 jobs.
- A heuristic algorithm inspired by algorithm σ_1. This algorithm applies 10 steps of first-improvement local search, using the same neighborhood as algorithm-σ_1.
- Finally, we employ the commercial Solver *Gurobi* 9.5.1 with 1 second time-limit. We choose this time-limit in order to match the *Gurobi* performance with our proposed heuristic performance, and make a time-wise comparison between the two. Indeed, any MILP solver is in theory capable of solving the scheduling problem using Optimization Model 3.1. Alas, the problem $QM||C_{max}$ is in $\mathcal{NP}$, hence even the most sophisticated MILP solver cannot solve very large instances to optimality. We start solving the instance with the solver and interrupt it after 1 second, using then as output the best current schedule encountered by the solver.

Figure 6.1 shows all the 1024 realizations of the Monte-Carlo process to estimate the average-case performance ratio for all the 12 algorithms. In Figure 6.1, there are three well separated groups of algorithms, separated by different average-case performances. We list them in decreasing performance order.

The top-most group is composed by the approximation algorithms. They act on the 16 or 24 largest jobs only, then the list scheduling logic is followed. As they are designed having the $QM||C_{max}$ problem in mind, they are better suited for large scheduling instances with respect to the solver.

The second group is composed by the direct generalization of the approximation algorithms: they act on the whole set of jobs, hence a better average-case performance is to be expected with respect to the original approximation algorithms. However, we are not sure if they are still approximation algorithms, in the worst-case sense, or if they are just heuristics. In particular, it is difficult to imagine a proof strategy for the algorithms Full-σ_1 and Full-σ_2.

The last group is composed by the proposed heuristic and by the *Gurobi* algorithm with 1 second time-limit. Our proposed heuristic is inspired by the Full-σ_1, but with an average-case performance in mind, leaving behind the meticulousness required by an approximation algorithm.

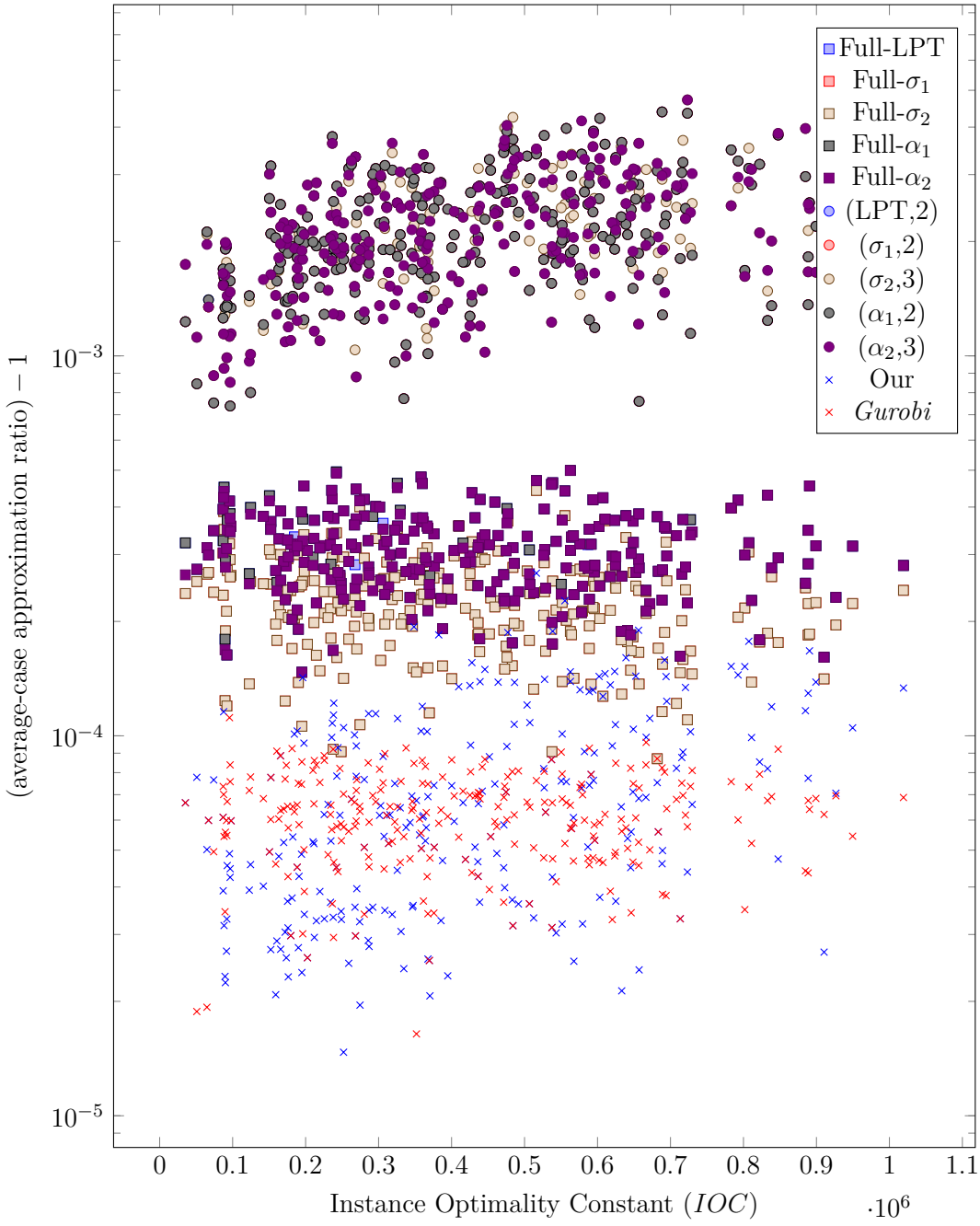

Figure 6.1. Heuristic Performance of some algorithms

In the practical average-case world, it could be interesting to compare algorithms also by their running times. Table 6.4 shows the average run times for each of the aforementioned algorithms.

As we can see, the algorithm with the highest runtime is Full-σ_2. This is well explained by the fact that this algorithm has a $\mathcal{O}(J^3)$ complexity, as basically all the possible combinations of three jobs should be considered during the local search step.

The time-limited *Gurobi* algorithm is the next slowest algorithm. Remember that we choose the 1 second time-limit for the *Gurobi* algorithm in order to match the performance of our proposed heuristic. Time-wise, the poor performance of *Gurobi* algorithm is not surprising. In fact, it is not designed with the scheduling problem in mind, but it is designed to be a general Mixed-Integer Non-Linear Program solver. This results in *Gurobi* being ≈ 1500 times slower than our proposed heuristic.

The Full-σ_2 algorithm is the next slowest algorithm, still one-hundred time faster than *Gurobi*.

Then there is our proposed heuristic. By reducing the number of first-improvement local search steps done in this algorithm, the run time can be reduced at the cost of average-case performance.

The remaining *Full*-style algorithms are faster than our heuristic, but they have no parameters to tune their performances. Finally, the approximation algorithms are the fastest, as they rely on the linear-time complexity List Scheduling algorithm for the majority of the jobs.

Algorithm Name	Time [μs]
Full-LPT	111.8936
Full-σ_1	1714.5967
Full-σ_2	1 392 090.5801
Full-α_1	147.5859
Full-α_2	176.0625
(LPT,2)	54.3008
(σ_1,2)	54.5342
(σ_2,3)	58.0332
(α_1,2)	54.3447
(α_2,3)	56.2080
Ours	746.5869
Gurobi	1 126 560.2827

Table 6.4. Average run time for an instance of $QM||C_{max}$ with $M = 8$ machines and $J = 2048$ jobs.

Chapter 7

Conclusions

In this work we addressed the question of approximating the uniform machines scheduling problem $QM||C_{max}$. In a nutshell, the $QM||C_{max}$ problem asks how to assign a set of jobs to a set of machines such that the total maximum running time is minimized, keeping in mind that each job is characterized by a duration and each machine by a speed of execution. This problem is $\mathcal{NP}$-hard hence we need to settle for approximation algorithms. The solution provided by a ρ-approximation algorithm is guaranteed to be at most ρ times worse than the optimal one. This finite constant ρ is called approximation ratio.

In this work we developed eight new approximation algorithms for the $QM||C_{max}$ problem. All the determined algorithms are low complexity polynomial time algorithms providing constant time approximation ratios. The best one has an approximation ratio $\rho = 1.1583 \approx \frac{-1+\sqrt{11}}{2}$.

Taking inspiration from Della Croce et al. [2019], all the developed algorithms share a common structure: a particular procedure is applied to a fixed number of long jobs, then the famous List Scheduling algorithm follows.

To reflect this common structure, in this work we establish a novel proof strategy, used with all the new approximation algorithms. Indeed, the quest to find approximation ratios is recast as mathematical programming problems, one for each proposed algorithm. Each of these optimization problems is split into two parts. One part, custom for each proposed algorithm, is solved by exploiting the power of a currently available commercial solver. The other part is dealt with very general propositions (that we developed) that can be applied to all proposed algorithms and many more.

The strength and beauty of this proof strategy is that, while it mixes both numerical and analytical results, it maintains complete formal validity. Hence, all efforts to prove approximation ratios are moved from developing custom proof strategies for each algorithm to recasting the problem as a mathematical program.

Surely, this work delineates a method for developing many more approximation algorithms, both for the $QM||C_{max}$ scheduling problem but also for other different problems. Indeed, as long as a candidate approximation algorithm can be recast as a mathematical programming model, the proof strategy explained and used in this work can be reused. In the world of scheduling problems, surely $PM||C_{max}$, $RM||C_{max}$ or some variants of $QM||C_{max}$ can be studied fruitfully with the same approach.

Moreover, it could be fascinating to develop an approximation algorithm for $QM||C_{max}$ with an approximation ratio of $\rho \leq \frac{3}{2}, \forall M \geq 2$ (improving any current known algorithm) following the same framework of the Meta-Algorithm. Indeed, using Proposition 4.0.4, only instances with M machines and at most $2M-1$ jobs must be taken into consideration.

Another future work could study more deeply the conjecture (5.2), from which a new PTAS for $QM||C_{max}$ may stem. From a cross-fertilization point of view, another interesting question that could be investigated is why all the approximation ratios obtained for $QM||C_{max}$ can be expressed as roots of some M degree polynomial.

Finally, as shown in Table 6.4, all the proposed approximation algorithms are very fast, as their complexity is linear in the number of jobs. This suggests that a more accurate study about heuristics that are generated from these approximation algorithms could be very prolific.

Bibliography

Tobias Achterberg and Eli Towle. Non convex quadratic optimization in gurobi 9.0. Technical report, Gurobi Optimization, 2020. URL https://www.gurobi.com/resource/non-convex-quadratic-optimization/.

Pierluigi Crescenzi, Giorgio Gambosi, Roberto Grossi, and Gianluca Rossi. *Strutture di dati e algoritmi*. Pearson Italia, 2 edition, 2012.

Federico Della Croce and Rosario Scatamacchia. The longest processing time rule for identical parallel machines revisited. *Journal of Scheduling*, 23(2):163–176, 04 2020.

Federico Della Croce, Rosario Scatamacchia, and Vincent T'kindt. A tight linear time $\frac{13}{12}$-approximation algorithm for the $p2||c_{max}$ problem. *Journal of Combinatorial Optimization*, 38(2):608–617, 08 2019.

Gregory Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.

Donald K. Friesen. Tighter bounds for lpt scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

Teofilo Gonzalez, Oscar H. Ibarra, and Sartaj Sahni. Bounds for lpt schedules on uniform processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

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

J.A. Hoogeveen, S.L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55 (3):259–272, 1994.

Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 04 1976.

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

Christos Koulamas and George Kyparisis. A modified lpt algorithm for the two uniform parallel machine makespan minimization problem. *European Journal of Operational Research*, 196:61–68, 07 2009.

Annamária Kovács. New approximation bounds for lpt scheduling. *Algorithmica*, 57(2): 413–433, 06 2010.

Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys. Chapter 9 sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. Elsevier, 1993.

Jan Karel Lenstra and David B. Shmoys. Elements of scheduling, 2019. URL https://elementsofscheduling.nl/.

Ivar Massabò, Giuseppe Paletta, and Alex J. Ruiz-Torres. A note on longest processing time algorithms for the two uniform parallel machine makespan minimization problem. *Journal of Scheduling*, 19(2):207–211, 04 2016.

Paul Mireault, James B. Orlin, and Rakesh V. Vohra. A parametric worst case analysis of the lpt heuristic for two uniform machines. *Operations Research*, 45(1):116–125, 1997.

Takuto Mitsunobu, Reiji Suda, and Vorapong Suppakitpaisarn. Worst-case analysis of lpt scheduling on small number of non-identical processors. Not yet published. Available on arXiv., 03 2022. URL https://arxiv.org/abs/2203.02724.

Michael L. Pinedo. *Scheduling*. Springer New York, NY, 2012.

Roberto Tadei and Federico Della Croce. *Elementi di ricerca operativa*. Società Editrice Esculapio, 2010.

Appendix A

The approximation ratios of LPT on a small number of uniform machines

In Section 5.1.4, using the framework of the Meta-Algorithm, we deduced an approximation algorithm called (LPT,2) for $Q2||C_{max}$. This approximation algorithm takes the $LM = 2 \cdot 2 = 4$ largest jobs, applies LPT to them, and completes the schedule applying list scheduling to the remaining jobs. As the LPT algorithm is just the application of list scheduling to the sorted jobs, we can give the following equivalent description of (LPT,2). The (LPT,2) algorithm takes as input the list of all the jobs and select the 4 largest ones. It sorts them and moves them at the beginning of the list. Finally, it applies list scheduling to the whole list.

Hence, the (LPT,2) algorithm and the LPT algorithm differ only by "how much" they sort the whole list of jobs. One could ask if it is possible to deduce the approximation ratio of LPT itself, dropping completely the Meta-Algorithm framework, using Optimization Model 5.2. Let recall that the optimum $\rho(M)$ of Optimization Model 5.2 is the approximation ratio of the LPT algorithm limited to an arbitrary fixed number of machines M and jobs J. Moreover, the LPT algorithm enjoys a similar property as the one proved in Proposition 3.2.1 for the Meta-Algorithm. In fact, all the jobs smaller than the critical one can be discarded without changing the approximation ratio. In other words, we can focus only on instances whose critical job is the smallest job.

We can include this last-mentioned interesting fact in the model. Moreover, we try to simplify the model as much as possible to explore higher and higher number of machines M. We introduce the following modification to Optimization Model 5.2.

1. Split the original model into a family of smaller models, indexed by the index K of the critical job p_K. In this way jobs after the critical one can be discarded.

2. As the index of the critical job is fixed in each of the smaller models, the constraint $h = \max_m \left(q_m \sum_j z^{LPT}_{m,j}\right)$ simplifies to $h \leq q_m \left(p_K + \sum_{j=1}^{K-1} z^{LPT}_{m,j}\right) \quad \forall m$ because of the LPT logic.
3. To reduce the number of nonlinear constraints, relax the constraints that impose the LPT logic by simply removing all of them but one (as it turned out to be necessary for $M = 7$). This constraint will be extremely simplified as we know that LPT will always assign the first job to the first machine.
4. To push the model "near" a LPT schedule, we introduce constraints that force initial jobs to be scheduled on the initial machines: $x^{LPT}_{m,j} = 0 \quad \forall m \gneq j$.
5. To further reduce the number of nonlinear constraints, use $s_m = \frac{1}{q_m}$ instead of q_m and simplify the denominators.
6. Introduce the positive variables $C_m = p_K + \sum_{j=1}^{K-1} z^{LPT}_{m,j} \quad \forall m$.

Hence, the model becomes $\tilde{\rho}(M) = \max_K \tilde{\rho}(M, K)$, where $\tilde{\rho}(M, K)$ is given by the following model:

Optimization Model A.1: LPT Orlin-Style

$$
\begin{aligned}
\tilde{\rho}(M,K) = \max \quad & h \\
\text{s.t.} \quad & C_m = p_K + \sum_{j=1}^{K-1} x^{LPT}_{m,j} p_j && (\forall m) \\
& h s_m \leq C_m && (\forall m) \\
& \sum_j x^*_{m,j} p_j \leq s_m && (\forall m) \\
& \sum_m x^{LPT}_{m,j} = 1 && (\forall j) \\
& \sum_m x^*_{m,j} = 1 && (\forall j) \\
& x^{LPT}_{m,j} = 0 && (\forall m \gneq j) \\
& C_m \geq 0 && (\forall m) \\
& \frac{1}{s_1}(p_1 + p_2) - \frac{1}{s_2}(p_2) \leq K(1 - x^{LPT}_{2,2}) \\
& 1 \geq p_1 \geq p_2 \geq \cdots \geq p_K \geq 0 \\
& 1 = s_1 \geq s_2 \geq \cdots \geq s_M \geq 0 \\
& h \geq 0, x^{LPT}_{m,j} \in \{0,1\}, x^*_{m,j} \in \{0,1\} && (\forall m, j)
\end{aligned}
$$

Note that the only remaining LPT constraint is further relaxed as:

$$\begin{aligned} & \frac{1}{s_1}(p_1+p_2)-\frac{1}{s_2}(p_2) \leq K(1-x_{2,2}^{LPT}) \\ \iff & s_2(p_1+p_2)-s_1(p_2) \leq s_1 s_2 K(1-x_{2,2}^{LPT}) \\ \implies & s_2(p_1+p_2)-s_1(p_2) \leq K(1-x_{2,2}^{LPT}) \end{aligned}$$

In order to save computational resources, we use the following workflow, exploiting Proposition 4.0.3:

Procedure A.1: Workflow for LPT bound

Input: The number of machines $M \geq 2$
Output: The approximation ratio ρ_M of LPT for $QM||C_{max}$

```
ρ_M ← 1;
K ← 3;
while ρ_M ≤ 1 + (M−1)/K do
    ρ_{M,K} ← solution of Optimization Model A.1;
    if ρ_M ≤ ρ_{M,K} then
        ρ_M ← ρ_{M,K};
    end
    K ← K + 1
end
```

We employ this workflow for $M = 2,\ldots,7$. The results, obtained with the commercial solved *Gurobi* in half an hour[1], are summarized in the following table:

M	ρ_M	K for which ρ_M is realized
2	1.2808	3
3	1.3837	4
4	1.4327	5
5	1.4591	6
6	1.4744	7
7	1.4837	8

Table A.1. Approximation ratios for LPT with a fixed number of machines $M = 2,\ldots,7$.

While results for $M = 2,3,4,5$ were already proved by Mitsunobu et al. [2022] using standard techniques, the results for $M = 6,7$ are completely new. These results make again

[1]Computational resources were provided by HPC@POLITO http://www.hpc.polito.it

clear the strength and the beauty of the proposed mathematical programming approach in deriving approximation results. Indeed, while Mitsunobu et al. [2022] were forced to prove increasingly difficult analytic propositions, we just let the algorithm run for half an hour in total, supplying it with different input parameters M.

The instances that realize these approximation ratios are the same proposed all the way back by Gonzalez et al. [1977]. We indeed have proved that these instances are the worst ones for $M \leq 7$.