

Scheduling on single and parallel machines with
constrained resources: batching or shared memory
environments

Alessandro Druetto

September 18, 2023

Università degli Studi di Torino



**UNIVERSITÀ
DI TORINO**

Dipartimento di Informatica

PhD in Computer Science

UniTO Doctoral School

Doctoral Thesis

**Scheduling on single and parallel machines
with constrained resources: batching or shared
memory environments**

Alessandro Druetto

Supervisors **Andrea Cesare Grosso**
Università degli Studi di Torino

Enrico Bini
Università degli Studi di Torino

Reviewers **Alessandro Agnetis**
Università degli Studi di Siena

Jean-Charles Billaut
Ecole Polytechnique de l'Université de Tours

September 18, 2023

Alessandro Druetto

Scheduling on single and parallel machines with constrained resources: batching or shared memory environments

Doctoral Thesis – September 18, 2023

Reviewers: Alessandro Agnetis and Jean-Charles Billaut

Committee: Pontus Ekberg, Luca Roversi and Vincent T'kindt

Supervisors: Andrea Cesare Grosso and Enrico Bini

Università degli Studi di Torino

UniTO Doctoral School – PhD in Computer Science

Dipartimento di Informatica

Corso Svizzera, 185

10149 – Torino (Italy)

Acknowledgments

Firstly, the biggest thanks goes to my two PhD supervisors, Andrea Cesare Grosso and Enrico Bini, who supported and guided me during the entirety of my PhD, looking forward to what my career will be after these three years. I really appreciate their efforts in stimulating my curiosity, always proposing new challenging problems trusting my ideas and my work, and always deeply analyzing my methods to aim for the best possible results. My time spent doing research with both of them was excellent, because of their high scientific and human value; they succeeded in sharing with me their passion for research.

Among the people I have met in the Computer Science department, I want to thank Roberto Aringhieri, Rossella Cancelliere and Davide Cavagnino in particular. They got me involved in research projects in which I was able to contribute in parallel to my PhD, in the fields of vehicle routing, machine learning and computer vision.

Along this journey, I was exposed by the Huawei Pisa Research Center to a problem that is very central in the embedded systems design: the joint mapping of tasks to cores and variables to memory. I learned a lot during this collaboration and would like to thank Marco Di Natale, Stefano Puri and Silvio Bacci of the Modeling Team in Pisa.

I would thank the reviewers Alessandro Agnetis and Jean-Charles Billaut for the precious time they spent on my doctoral thesis and for their remarks that allowed me to improve my work and suggested new developments.

Huge thanks to all my family as well, for all their essential support, and for their neverending trust in my aspirations; even when I decided to abandon my old job to go back to University and continue my studies. Last but not least, special thanks goes to Anna, my significant other (and her parents), who supports me everyday and in every possible way. With her, every moment is brighter.

Alessandro Druetto

Torino – September 18, 2023

Contents

1	Introduction	1
1.1	Research motivations	2
1.1.1	Flow models for parallel batching	2
1.1.2	Process scheduling in embedded systems	3
1.2	Thesis outline	4
2	Literature review	7
2.1	Flow models for parallel batching	7
2.1.1	Polynomially solvable problems	9
2.1.2	Minimizing the maximum completion time	10
2.1.3	Minimizing the maximum lateness	11
2.1.4	Minimizing the (weighted) total completion time	12
2.1.5	Minimizing the (weighted) number of tardy jobs	13
2.1.6	Minimizing the (weighted) total tardiness	13
2.1.7	Other objective functions	14
2.1.8	Contribution of this work	14
2.2	Process scheduling in embedded systems	15
2.2.1	Metaheuristic techniques	15
2.2.2	Direct Acyclic Graphs	16
2.2.3	Specific environments	17
2.2.4	Generalization of memory hierarchy	18
2.2.5	Related patents	19
2.2.6	Contribution of this work	20

I	Flow models for parallel batching	21
3	Problem description	23
3.1	Unweighted total completion time: exact approach and parallel machines	24
3.2	Weighted total completion time: heuristics for the single machine case	24
3.3	Unweighted total completion time: heuristics for multi-size jobs and incompatibility families	25
3.4	Unweighted total completion time: analysis of two polynomial-size models	25
4	Unweighted total completion time: exact approach and parallel machines	27
4.1	Single-machine models	28
4.1.1	A new problem formulation	29
4.1.2	Continuous relaxation for the new graph-based formulation: Column Generation	33
4.1.3	Heuristic procedure: Price-and-Branch	36
4.1.4	Exact approach: Branch-and-Price	36
4.2	Parallel-machines models	38
4.3	Computational results	43
4.3.1	Evaluation of the heuristic algorithms	44
4.3.2	Evaluation of the exact approach	50
4.4	Final remarks	54
5	Weighted total completion time: single machine and heuristics	57
5.1	Column Generation models	58
5.1.1	The graph-based model	58
5.1.2	An arc-based flow model	62
5.1.3	A path-based model	67
5.2	Upper bounding: heuristics	70
5.2.1	Variable Rounding Upper Bound	70
5.2.2	Early Rounding Upper Bound	71
5.3	Computational results	72
5.3.1	Performance of basic models	73
5.3.2	Generating feasible solutions by rounding	74
5.4	Final remarks	78
6	Unweighted total completion time: heuristics for multi-size jobs and incompatibility families	81
6.1	Multiple sizes and incompatible families	82
6.2	Column Generation-based heuristics	83
6.2.1	The CG-LB Column Generation algorithm	83
6.2.2	The CG-UB and VR-UB heuristic procedures	86

6.3	Computational results	87
6.3.1	Standard instances ($b_i = 10$)	88
6.3.2	Extra instances ($b_i = 50$)	93
6.4	Final remarks	95

7	Unweighted total completion time: analysis of two polynomial-size models	97
7.1	Models description	97
7.1.1	Arc-flow models	98
7.1.2	A polynomial-size flow-based model	98
7.1.3	A stronger model	99
7.2	Variable Rounding heuristic	100
7.2.1	Variable Rounding for the 7.1.3 model	101
7.2.2	Variable Rounding for the 7.1.2 model	101
7.3	Computational results	102
7.3.1	Testing environment	102
7.3.2	Results for the 7.1.2 model	103
7.3.3	Results for the 7.1.3 model	104
7.4	Final remarks	106

II Process scheduling in embedded systems 109

8 Process scheduling and memory mapping: multi-step optimization approach 111

- 8.1 CPUs and memories in NUMA 113
- 8.2 System model 114
 - 8.2.1 Hardware model 114
 - 8.2.2 Software model 115
- 8.3 Problem description 116
- 8.4 Binding labels to runnables 118
 - 8.4.1 Polynomial-time algorithms 120
- 8.5 Mapping runnables to CPUs 121
 - 8.5.1 Hierarchical Clustering 125
- 8.6 Aggregation of runnables into tasks 127
- 8.7 Assigning priorities to tasks 128
- 8.8 Experiments 130
 - 8.8.1 The use case 130
 - 8.8.2 A Simulated Annealing approach 132
 - 8.8.3 Setup for our approach 133
 - 8.8.4 Computational results 134
- 8.9 Final remarks 136

9 Conclusions	137
9.1 Flow models for parallel batching	137
9.1.1 Time-indexed formulation for exponential-size models	137
9.1.2 Application of Column Generation to polynomial-size models	138
9.1.3 Flow formulations for Common Server problems	139
9.2 Process scheduling in embedded systems	139
9.2.1 Handle modifications over runnables	139
9.2.2 Application to different architectures	140
Bibliography	141
List of Publications	151
List of Acronyms (Part I)	153
List of Acronyms (Part II)	155
List of Figures	157
List of Tables	159

” *There cannot be a crisis next week.
My schedule is already full.*

— **Henry Alfred Kissinger**
(1973 Nobel Peace Prize)



SCHEDULING is everywhere. From business management to industrial machinery, passing through multi-processor computers, finding the optimal sequence of actions to be undertaken in order to maximize the profit (or minimize the cost) under resource constraints is an always appealing and often difficult goal to achieve.

Furthermore, having a good scheduling plan makes every system more robust towards unexpected events that could arise. If all jobs to be manufactured by machines in some interconnected industrial process are perfectly scheduled to meet their delivery dates with a good margin, if something goes wrong with a job or on a particular machine one can often recover from the issue without compromising the production chain. When all tasks to be run in a complicated software system over multiple processors are tightly scheduled, that leaves room for eventual new and urgent tasks that could spawn and require computational power without having to stop some of the tasks.

During my PhD in Computer Science I decided to dedicate myself to the study of hard scheduling problems in contexts with constrained resources. In some industrial processes the jobs need to be processed in batches with limited size, the processing time required by a batch depends on the individual processing times of contained jobs, and having each one of the batches to end their processing as soon as possible is difficult to achieve. In embedded multi-processor systems the tasks have to be partitioned between all processors aiming at maximizing the free computational time still available on them. However, high-bandwidth memory cannot typically host all variables; it is then necessary to store some of them in the global low-bandwidth memory, with a negative impact over computational times. This fact could result in situations of unfeasibility.

The principal trend of my research has been, in fact, the following: *scheduling with constrained resources*. In particular the investigation has been focused on two lines:

- the study of a complicate family of parallel batch scheduling problems and the development of efficient algorithms for several variants of the problem;

- the development and subsequent application of ad-hoc assignment and scheduling algorithms for specific problems that arise in automotive embedded systems.

Apart from this principal trend, other problems have been addressed during my PhD, but are out of the scope of this thesis and will not be discussed here.

1.1 Research motivations

Next two sections report a brief introduction over the two lines of investigations I decided to pursue during my PhD:

- Section 1.1.1 is about the analysis done over a family of parallel batch scheduling problems;
- Section 1.1.2 relates to specific problems that arise in automotive embedded systems.

1.1.1 Flow models for parallel batching

In manufacturing system management, capacity is a key factor to have supply matching demand, that is, to have a system able to produce what is needed to satisfy customer demand.

Several are the factors negatively impacting the system capacity. The most studied ones are those related to system balancing and to part batching when setup times are present, as severe bottlenecks and/or small batches can substantially reduce the system capacity, thus leading to the incapacity of the manufacturing system to timely respond to the market demand (Cachon and Terwiesch [14]).

Batches induced by setup times are called serial batches and, although they are very important in manufacturing systems, they are not the only type of batches that can be present in the shop floor. Transfer batches and parallel batches can also be found in manufacturing systems, the first being related to the capacity of the material handling resources and the second, as the serial ones, to the capacity of the machines.

Although both serial and parallel batches are related to and affect machine capacity, their nature is very different. Serial batches are due to the presence of setup times, while parallel batches stem from the ability of machines to accommodate and manufacture several jobs at the same time. They are less studied than serial and transfer batches, because they are less frequent; however, they are not less important.

Specifically, parallel batches can be found in many manufacturing processes where heating operations are necessary, such as in mould manufacturing (Liu et al. [44]) and semiconductor industry (Mönch et al. [51]), or when there are sterilization phases (Ozturk et al. [57]), just to cite a few examples.

In all these cases, operations take a quite long time and the machines usually are batch machines that can accommodate several parts and process them simultaneously, exactly to virtually share the long processing time among all the parts processed at the same time. Each part has an individual size and batch machines (that is, batch oven for heating treatments or autoclaves for sterilization operations) have a limited capacity; therefore, the number of parts that can be in a single batch is limited.

Due to the limited capacity of the batch machine, and then to the limited number of parts that can be accommodated in it, when several jobs have to be processed on the batch machine, they have to be partitioned in several batches. When batches have been created, their processing has to be scheduled on the machine, and this decision is obviously intertwined with batch creation. Moreover, the two decisions (how to create batches and how to sequence them on the batch machines) strictly depend on the objective the shop floor manager aims at (that is, minimizing the number of tardy jobs, minimizing the maximum delay, reducing the total flow time, maximizing the machine utilization, etc).

1.1.2 Process scheduling in embedded systems

In many embedded systems, including most of automotive real-time controls, the fundamental problems for designers are:

- the definition of the task model from the set of functions that need to be executed;
- the allocation of those tasks to all Central Processing Units (CPUs);
- the mapping of data onto memory, including all variables shared among functions.

In automotive systems, the application definition is in most cases formalized by the AUTomotive Open System ARchitecture (AUTOSAR) standard (Fürst et al. [93]), in which a system is defined as a collection of components with a well defined data and operation interface. The internal behavior of each component consists of a set of *runnables* (functions) activated in response to events. The situation is the same in control applications developed according to other paradigms, such as the code generated by Simulink.

For example, when generating code for Simulink subsystems, the code generator creates from two to four functions for each subsystem, one of which needs to be

called at system initialization, one (optional) at termination, and one or two usually periodically with the same rate of the subsystem. These functions are in every means equivalent to the AUTOSAR runnables, and their task implementation and mapping on CPUs is formulated in the same way.

In both use cases, and also when the application is coded manually, the problem input consists of a set of runnables to be executed according to some specified event (periodic, upon the completion of another one, upon receiving some input or call request). A set of *tasks* needs to be defined to execute these runnables. These tasks need to have an activation pattern that is consistent with the ones of the functions executed by them, need to be allocated to a CPU, and the data (including the program and communication variables) needs to be allocated in memory.

Solving this problem is not easy and is key to achieve good performance of the application. It is the concern of most designers how Non-Uniform Memory Access (NUMA) architectures (Lameter [102]) can result in a large variation in the execution times if the memory allocation is not carefully managed. Similarly, with the advent of platforms with more CPUs, the problem becomes highly combinatorial and unlikely to be solved in an effective way by a human designer without automation support.

The problem has been investigated along several lines by the research community. However, the delivered performance and insights of current methodologies accounting for the memory allocation are not fully satisfactory.

1.2 Thesis outline

This thesis is divided in two parts, that corresponds to the two principal lines of investigation I followed during my PhD. Those two topics were already very briefly described in Section 1.1.1 and Section 1.1.2.

First, in Chapter 2 is found a broad and (to the author's knowledge) as complete as possible literature review over relevant works, for both parts (Section 2.1 and Section 2.2) in which the thesis is divided.


Part I contains the analysis over a family of parallel batch scheduling problems (Chapter 3) with complicating constraints, that resulted in the development of new state-of-the-art bounds, efficient heuristics and exact methods. Chapter 4 defines the innovative approach for this problem family while providing both very strict bounds and an exact approach. Chapter 5 studies the weighted variant of the problem and introduces new heuristics that are both fast and of high quality. Chapter 6 considers variants with more strict constraints (that is, multiple sizes and incompatibilities) combining bounds and heuristics described in the previous chapter with good results. Chapter 7 defines another approach for this problem family, giving

two modeling strategies able to produce relatively small-sized models that can be efficiently processed by a commercial solver.

Part II contains a specific case study over a memory mapping, task scheduling, and priority assignment problem that arises in embedded systems for automotive applications, that resulted in the development of an efficient and adaptable ad-hoc approach. Chapter 8 describes in detail this multi-step optimization approach in application to the considered use case with excellent results, that resulted in a patent application.

Finally, in Chapter 9 is found a brief conclusion with the description of future works and research directions that are currently being undertaken, both relevant to parallel batching (Section 9.1) and to process scheduling (Section 9.2).

Literature review

 HIS chapter illustrates a broad review upon related works about the topics discussed in my PhD thesis, subdivided in two sections as the entire thesis is also split in two parts. Section 2.1 contains the literature review pertinent to Part I, while in Section 2.2 it is present the literature review relevant for Part II.

2.1 Flow models for parallel batching

The batch scheduling problem, addressed in the first part of the thesis, has been studied for a few years by many researchers (Ikura and Gimple [33]) because of its many fields of application.

We consider batching problems where a set $N = \{1, 2, \dots, n\}$ of n jobs have to be scheduled in several batches. Each batch has a capacity which limits the maximum number of jobs that can be contained. We define a capacity b , typically $< n$, which can be measured as the maximum number of jobs that can be batched together or to the size of the batch. The second case is a generalization and reduces to the first when all jobs have equal size.

A solution to the problem is made by a batch schedule, that is a sequence of batches $S = (B_1, \dots, B_k)$. This schedule must be *feasible* with respect to the maximum batch capacity allowed for the specific instance.

In a parallel batch scheduling problem, all jobs in a batch are processed simultaneously and the processing time p_B of a batch equals the longest processing time of all contained jobs. Also, a job is completed when all other jobs in the same batch are completed, hence the completion time C_j of each job j equals the completion time of the batch that contains it.

At the time of writing, the broadest survey available for problems with batching, from an algorithmic point of view, is still the one by Potts and Kovalyov [59], although followed by Mathirajan and Sivakumar [46], Mönch et al. [50] and Fowler and Mönch [28]. This absolutely does not mean that there is no recent literature about batching problems; on the contrary the big picture about batching problems seems to have become extremely varied and complex, where each work focuses on very specific technological constraints, that often change the mathematics of the models.

The literature review is structured as follows. After a brief look at polynomially solvable problems, the subsequent sections give a panoramic view over complexity and heuristics for a selection of relevant problem families, grouped by the different types of considered objective function.

Graham notation

Reviewed problems are referred to following the classical three-field standard notation $\alpha|\beta|\gamma$ by Graham et al. [31], to distinguish between different machine scheduling problems, in which a set of tasks has to be sequenced and assigned to one or more machines.

The α field represents the machine environment, where a value specifies the number of available machines and a letter indicates the processing category in which those machines belong. The β field indicates whether the problem has some specific characteristics or constraints. Finally, the γ field specifies the objective function to be optimized.

In Tab. 2.1 is listed a brief, and non-exhaustive, summary of various options that can be used in Graham's three-field notation to characterize a specific problem.

Tab. 2.1: Graham three-field notation: explanation of various fields.

α	description
1	a single machine
Pm	environment with m identical parallel machines
Rm	environment with m unrelated parallel machines
Om	open shop with m machines
Fm	flow shop with m machines
Jm	job shop with m machines
β	description
p -batch	all jobs in a batch are processed in parallel
batch $\{AM\}$	batches are processed by an Additive Manufacturing machine
b, b_m	the batch capacity, eventually different on each machine m
r_j	each job j has a release date
σ_j	each job j has a size
$p_j = p$	all jobs have equal processing times
$d_j = d$	all jobs have equal due dates
<i>incomp</i>	incompatibilities exist between jobs from different families
<i>prec</i>	precedence relations exist between jobs
s	each batch has a constant setup time
s_f	each batch with jobs from family f has a different setup time
γ	description
C_{\max}	minimize the maximum completion time among jobs
$\sum(w_j)C_j$	minimize the (weighted) total completion time
L_{\max}	minimize the maximum lateness among jobs
$\sum(w_j)T_j$	minimize the (weighted) total tardiness
$\sum(w_j)U_j$	minimize the (weighted) number of tardy jobs

Jobs can have several parameters, such as:

- *weight*, a penalty factor that impacts the objective function;
- *release date*, the point in time on which a job is available for processing;
- *size*, a physical dimension that have effect on several constraints;
- *processing time*, the time required for a job to be successfully processed;
- *due date*, the point in time before which a job must have been processed;
- there can be *incompatibilities* between jobs, that is, jobs that cannot be processed in the same batch;
- there can be *precedence relations* between jobs, that is, jobs that must end their processing before the start of other jobs.

With regard to the objective function, several job properties can be considered as the optimization focus, such as:

- C_j is the *completion time* of job j , the point in time on which a job ends its processing;
- L_j is the *lateness* of job j , defined as $L_j = C_j - d_j$, the difference between completion time and due date;
- T_j is the *tardiness* of job j , defined as $\max\{L_j, 0\}$;
- U_j indicates if job j is *tardy* or not, its value is 1 if $L_j > 0$ and 0 otherwise.

2.1.1 Polynomially solvable problems

There are few problems that are known to be solvable by polynomial or pseudo-polynomial algorithms.

Single machine

A first case is the restricted batch size problem $1|p\text{-batch}, b|\sum C_j$, with b , the maximum number of jobs per batch, fixed; Brucker et al. [11] developed a polynomial Dynamic Programming (DP) algorithm to solve this problem. The variant of $1|p\text{-batch}, b|\sum C_j$ where jobs have q distinct processing times was solved in the most efficient way by Brucker et al. [11] in $O(b^2 q^2 2^q)$ time.

Lee et al. [40] considered the maximum lateness and the sum of tardy jobs as performance measures for problems taking in account jobs with agreeable release dates and due dates. Agreeable release dates were considered such that if $d_i \leq d_j$, then $r_i \leq r_j$; agreeable due dates were considered such that if $p_i \leq p_j$, then $d_i \leq d_j$. From the work of Lageweg et al. [38], they deduced the polynomial time complexity

of the following problems: $1|p\text{-batch}, b, r_j, p_j = p|L_{\max}$ with agreeable release dates, $1|p\text{-batch}, b|L_{\max}$ with agreeable due dates, $1|p\text{-batch}, b, r_j, p_j = p|\sum U_j$ with agreeable release dates and $1|p\text{-batch}, b|\sum U_j$ with agreeable due dates.

Baptiste [7] extended complexity results on polynomial solvable problems for the case $1|p\text{-batch}, b, r_j, p_j = p|F$, where $F \in \{\sum w_j U_j, \sum w_j C_j, \sum T_j, L_{\max}\}$.

Multiple machines

For identical parallel machines, results on the unrestricted batch size problem are generalized by Brucker et al. [11] and show that algorithms can be adapted to the case of m identical parallel machines returning optimal solutions in pseudo-polynomial time.

Complexity of open shops, flow shops and job shops were considered by Potts and Kovalyov [59] with the makespan minimization as objective function. The unrestricted case in all scenarios is shown to be polynomially solvable. Also, the $F2|p\text{-batch}, b_1 = 1|C_{\max}$ and the $J2|p\text{-batch}, b_1 = 1|C_{\max}$ problems are solved by them with polynomial algorithms.

2.1.2 Minimizing the maximum completion time

Single machine

Large part of the literature on parallel batching is devoted to the minimization of the maximum completion time, also called *makespan*, criterion: see Dupont and Dhaenens-Flipo [24], Damodaran et al. [15], Rafiee Parsa et al. [60], Li [42] and Muter [54].

Uzsoy et al. [70] first proved the NP-hardness of the single batching machine scheduling problem with batch capacity constraint and jobs with different sizes. The problem $1|p\text{-batch}, b, \sigma_j|C_{\max}$ is shown to be NP-hard even in the case with equal processing times, hence for the problem $1|p\text{-batch}, b, \sigma_j, p_j = p|C_{\max}$. They formulated some heuristics and a lower bound to run computational experiments. Melouk et al. [48] formulated a Mixed-Integer Program (MIP) and developed a Simulated Annealing algorithm for the problem. Rafiee Parsa et al. [60] later formulated the problem using Dantzig-Wolfe decomposition as a set partitioning problem to obtain tighter lower bounds through Column Generation (CG) and to develop a Branch-and-Price (B&P) algorithm. Muter [54] presented a cut and CG method, which integrates the batch generation and machine schedule generation in a single pricing subproblem. An original arc-flow MIP is proposed by Trindade et al. [67] for makespan minimization, leading to excellent computational results on very large instances.

Boudhar [10] considered the problem $1|p\text{-batch}, b, \text{incomp}|C_{\max}$ with unit size jobs, where incompatibilities between jobs exist; such relations are represented through

a compatibility graph, which is split to minimize the makespan graph. While the problem for $b = 2$ is polynomial, the authors show the NP-hardness for greater batch capacity. Online algorithms for the problem are considered by Bellanger et al. [8] and Fu et al. [29].

A batch scheduling problem that specifically refers to the additive manufacturing context is addressed by Kucukkoc [37]: the problem $1|batch\{AM\}|C_{max}$, where jobs processing times are a function of jobs, material and process characteristics. Constant setups are also considered and included in the processing time of each batch. The author formulates a MIP, where the processing machine is treated as a batch and its area defines the constraint on the number of jobs that can be processed together.

Multiple machines

Lee et al. [40] define the $P|p\text{-batch}, b|C_{max}$ problem to be NP-hard by extending the complexity result of the $P||C_{max}$ problem obtained by Lageweg et al. [38]. They then derive the worst-case performance of some ordering scheduling algorithms for the batching problem.

Results and algorithms developed for the single machine environment are extended to the parallel machines problem $P|p\text{-batch}, \sigma_j|C_{max}$ by Melouk et al. [48] and Muter [54]. Referring to the additive manufacturing context and related both to identical and non-identical parallel machines, are the $P|batch\{AM\}|C_{max}$ and $R|batch\{AM\}|C_{max}$ problems, extended by Kucukkoc [37]. Trindade et al. [68] improved existing formulations of the problem and incorporated release times. Ozturk et al. [56] addressed a variation of this problem where all jobs exhibit unit sizes, developing a Branch-and-Bound (B&B) algorithm.

Unrelated parallel machines are tackled by Arroyo and Leung [4] that also consider different release dates; the problem is solved by means of some heuristics.

2.1.3 Minimizing the maximum lateness

Single machine

The single machine batching problem $1|p\text{-batch}, b|L_{max}$ that minimizes the maximum lateness was proved to be NP-hard by Brucker et al. [11]. As already mentioned, few exceptions of polynomially solvable are presented in Li and Lee [41], considering the special case of $1|p\text{-batch}, b|L_{max}$ with agreeable due dates, $d_i \leq d_j$ if $p_i \leq p_j$, and developing a polynomial algorithm. They also consider the case $1|p\text{-batch}, b, p_j = p, r_j|L_{max}$ with agreeable release times, $r_i \leq r_j$ if $d_i \leq d_j$, also showing its polynomial complexity.

Malapert et al. [45] addressed the problem $1|p\text{-batch}, b, \sigma_j|L_{\max}$ with non-identical job sizes by means of a constraint programming approach. Zhou et al. [72] incorporated release dates and developed for the $1|p\text{-batch}, b, r_j, \sigma_j|L_{\max}$ problem a modified particle swarm optimization. The algorithm is also shown to be competitive with others from the literature for the case without release dates.

Cabo et al. [13] develop in their work a split-merge neighborhood of exponential size that can be searched in polynomial time by DP for the $1|p\text{-batch}, b|L_{\max}$ problem. In Cabo et al. [12], in order to consider a bi-objective minimization of both the maximum lateness and the number of batches, authors introduce a MIP formulation that takes in account both objectives and apply the epsilon-constraint method and a biased random key genetic algorithm to solve the problem. Emde et al. [25] tackled in their work the more general case $1|p\text{-batch}, b, prec, incomp|L_{\max}$, that considers incompatibilities and precedence relations between jobs. A MIP is formulated and a logic-based Benders decomposition (Benders [9]) developed, which is shown to optimally solve even most of 100 jobs instances and to outperform existing exact methods for the specific case without incompatibilities and precedence relations.

2.1.4 Minimizing the (weighted) total completion time

Single machine

The total completion time problems, also called *total flow time*, have been less studied than the makespan ones: see Jolai Ghazvini and Dupont [35] and Rafiee Parsa et al. [61]. The work in Jolai Ghazvini and Dupont [35] and a modified version of the genetic algorithm presented by Damodaran et al. [15] have been used as benchmark procedures to the hybrid max-min ant system presented by Rafiee Parsa et al. [61].

Uzsoy [69] proved the NP-hardness for the $1|p\text{-batch}, b, \sigma_j|\sum C_j$ problem, where jobs have different sizes; this result holds even if all jobs have equal processing times.¹ In this work are provided some heuristics and a branch and bound procedure to solve the problem. For the case of jobs having unitary size, DP algorithms were developed for the problem $1|p\text{-batch}, b|\sum C_j$ in the case of arbitrary or q distinct processing times. The most efficient implementations are presented by Brucker et al. [11] and work respectively in $O(n^{b(b-1)})$ and $O(b^2q^22^q)$ time.

Azizoglu and Webster [5] generalized the model of Uzsoy [69] to the case with both arbitrary job sizes and weights, $1|p\text{-batch}, b, \sigma_j|\sum w_jC_j$. They extended the branch and bound procedure to this case and exploited some dominance property.

¹Note that, despite Uzsoy [69] being from 1994, Brucker et al. [11] in 1998 cast the problem as still open; also the review by Potts and Kovalyov [59], published in 2000, does not cite this paper at all. Only the review by Fowler and Mönch [28], published in 2022, correctly cites this paper. The discrepancy may be due to the fact that Uzsoy [69] was published online only in 2007. See the paper and its details here: <https://doi.org/10.1080/00207549408957026>

In Azizoglu and Webster [6] the same authors adapted the procedure also to the problem with incompatible job families, $1|p\text{-batch}, b, \sigma_j, \text{incomp}| \sum w_j C_j$, where jobs in the same family have equal processing times. Heuristics are provided by Dobson and Nambimadom [19] for the aforementioned problem.

Multiple machines

For the multiple machines case, there is a CG method based approach, studied in Ozturk [55], that considers identical parallel machines. The problem is here decomposed in two stages: firstly CG is used to generate batches and these are then scheduled on machines in a second stage, using a MIP solver.

2.1.5 Minimizing the (weighted) number of tardy jobs

Single machine

Brucker et al. [11] first proved that the batching problem with restricted batch size that minimizes the sum of tardy jobs $1|p\text{-batch}, b| \sum U_j$ is NP-hard, excluding as we mentioned before few exceptions that were proven to be solvable in polynomial time.

Jolai [34] studied the problem $1|p\text{-batch}, b, \text{incomp}| \sum U_j$, where jobs are partitioned into m incompatible families and cannot be processed in the same batch when belonging to the same family. The problem is also shown to be NP-hard and it is solved through DP, which is polynomial for fixed m and b . The case where all jobs from the same family share a common due date is also considered and solved by a pseudo-polynomial time procedure.

Dauzère-Pérès and Mönch [16] presented two MIP position-based formulations for the weighted case with incompatible job families, $1|p\text{-batch}, b, \text{incomp}| \sum w_j U_j$. The two models are compared and a random key genetic algorithm is developed to solve the problem.

2.1.6 Minimizing the (weighted) total tardiness

Single machine

The $1|p\text{-batch}, b| \sum T_j$ is another NP-hard problem. In Lawler [39] was shown that even the single machine weighted total tardiness without batching $1|| \sum w_j T_j$ is NP-hard; later, the unweighted version of the problem $1|| \sum T_j$ was proven to be NP-hard as well by Du and Leung [23].

For the $1|p\text{-batch}, b, \text{incomp}| \sum w_j T_j$ problem, considering incompatible job families, heuristics were developed by Perez et al. [58].

Multiple machines

There are some studies on multiple machine environments. Mönch et al. [49] addressed the minimization of the total weighted tardiness for parallel machines, when incompatible job families exist. The $P|p\text{-batch}, b, \text{incomp}| \sum w_j T_j$ problem is solved by means of a genetic algorithm that assigns batches to machines and each of them is then scheduled as a single machine.

Mathirajan et al. [47] studied a specific case, $P|p\text{-batch}, b, \sigma_j, \text{incomp}| \sum w_j T_j$, coming from the steel casting industries with dynamic jobs arrival, weighted jobs, non-identical jobs sizes, incompatible families and different jobs priorities. Greedy heuristics are proposed to solve the problem.

Tan et al. [66] studied a two stages flow shop, where the batch capacity depends on the stage. Jobs have different weights, different release times and there exists incompatibilities between job families; the problem can be categorized as $F2|p\text{-batch}, b_m, r_j, \text{incomp}| \sum w_j T_j$. A MIP formulation is presented and a hybrid stage-based decomposition approach is developed and compared with simpler heuristics.

2.1.7 Other objective functions

Fan et al. [26] addressed a batch scheduling problem where two agents have to schedule their jobs on a common batching machine. They consider two cases: the first where jobs are compatible, meaning that jobs of both agents can be processed together; the second where this is not possible, meaning that there is incompatibility between jobs of different agents. The objective is to have a schedule that is globally optimal. Both C_{\max} and $\sum C_j$ objectives are considered and complexity results are given.

Shahidi-Zadeh et al. [63] considered the scheduling of unrelated parallel machines with a bi-objective performance criterion that minimizes C_{\max} and penalties for both earliness and tardiness, plus the purchasing cost.

In Shahvari and Logendran [64] a bi-objective batch processing problem with dual resources on unrelated parallel machines is addressed. A mathematical programming model and particle swarm optimization algorithms are proposed for minimizing the production cost including total cost of tardy and early jobs (E-T) along with total batch processing cost, as well as the makespan (C_{\max}) with dual resources.

2.1.8 Contribution of this work

In Alfieri et al. [2] we formulated a new MIP for the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem based on a graph model, where the nodes of the graph represent jobs positions and arcs represent batches. The proposed model considered both arbitrary job processing

times and sizes, is solved by means of a CG technique and allows to compute effective lower bounds and heuristics.

After that, we developed further on this approach, leading us to a complete B&P exact procedure (Alfieri et al. [3]), to address the $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$ problem considering multiple parallel machines (Alfieri et al. [3]), to address the weighted version $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ of the problem (Druetto and Grosso [20]) with arbitrary job weights, and to consider combinations of constraints like incompatibility families and presence of multi-size jobs, $1|p\text{-batch}, b, \sigma_{ij}, \text{incomp}| \sum C_j$ (Druetto et al. [22]).

In our current last work on this topic (Druetto and Grosso [21]) we formulated two innovative MIP for the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem that are polynomial in the number of variables, and that can be directly solved without having to implement the aforementioned CG. In addition to that, the more efficient of the two models was further improved with the rounding heuristic developed for Druetto and Grosso [20]. Although this approach is not as efficient as the approach previously developed for Alfieri et al. [3], it proves that such a difficult problem can be solved even without relying on complicated techniques. To the writer's knowledge, this is the first MIP formulation for the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem that can be handled by a commercial solver in a reasonable amount of time.

2.2 Process scheduling in embedded systems

The problem addressed in the second part of the thesis, and its numerous variants, are at the merger of several technologies and domains of interest. Also, the techniques exploited to address the different challenges imposed by these problems are very different; in the following a complete (to the writer's knowledge) summary of related works is given.

This literature review is organized by grouping together works that employ similar optimization techniques and/or deal with similar problem variants; a last section highlights our contribution especially in comparison with those existing approaches.

2.2.1 Metaheuristic techniques

A large number of previous works makes use of stochastic optimization techniques.

In McLean et al. [111] a Simulated Annealing (SA) algorithm is employed to assign tasks to CPUs and to generate a static schedule of tasks per CPU that is compatible with the communication pattern. An algorithm based on SA was also proposed by Fauberteau and Midonnet [90] to partition non-independent tasks over a multi-core platform and assign them a priority value. Similar to our case, the objective of the mapping is to maximize the robustness of the application against execution

overruns, computed as the time units the execution time of each task can be inflated while still retaining the schedulability of the whole taskset. However, the approach does not address the problem of mapping runnables to tasks and labels to memory.

Genetic Algorithms (GAs) were also used to map tasks over heterogeneous processors (Alexandrescu et al. [74]), though ignoring the placement of shared labels. The combination of eight selected heuristics combined with GAs minimizing the makespan was proposed in Page et al. [116]. Such a goal, however, does not capture the requirements of real-time workloads. GAs are also applied in the context of mapping tasks over virtual machines (Chen et al. [82]).

Among the meta-heuristics, Systematic Memory-based SA was also used to partition AUTOSAR applications (Faragardi et al. [89]). In the context of the AMALTHEA Projects (Wolff et al. [130]), GAs were also proposed to solve the mapping problem (Cuadra et al. [84]). Finally, Bouaziz et al. [80] proposed a Multi-Objective Evolutionary Algorithm to find the Pareto front of the runnables to tasks mapping, whereas in Ferrandi et al. [92] is proposed an ant-colony approach to the problem of mapping both tasks and communication messages over an heterogeneous architecture.

In all of these papers, not only the optimization engine is different. Also, the impact of label allocation on the execution time of runnables is either modeled as a generic communication cost to be minimized (with an extremely simplified representation of the memory accesses overheads and their dependency on the memory structure), or ignored altogether.

Based on past experience and our own, we claim that *stochastic optimization methods are in general unfit for the whole mapping problem*, since it is very difficult to select the right set of transitions (or mutation/crossover) operators to escape from local optima. Consequently, the quality of the found solution is extremely difficult to evaluate. To support this claim, in our work we developed a SA solver and tried several variants without achieving the same quality of the solution found through the proposed Integer Program (IP) formulation developed during my PhD. In fact, our SA alternate solver remained stuck at a local minimum despite executing for more than 20 hours.

2.2.2 Direct Acyclic Graphs

In a significant portion of related literature, applications composed by runnable sharing data are modeled by Direct Acyclic Graphs (DAGs) (Pathan et al. [117]).

Lumpp et al. [108] implemented a mapping and scheduling algorithm for an image processing application and measured the achieved average performance. In Senapati et al. [123] is proposed a Constraint Satisfaction Problem (CSP) formulation to find the mapping and the static schedule that minimizes the makespan of a DAG. The size

of their use cases is, however, two orders of magnitude smaller than our automotive one. While this is suitable whenever a runnable is triggered by the completion of others, models based upon DAGs are not well suited for shared memory applications which are targeted by this research.

A few interesting ideas were published in the context of distributed systems. Two new algorithms are presented by Topcuoglu et al. [126], Heterogeneous Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CPOP), to schedule non-preemptively DAGs over heterogeneous processors, accounting for communication costs and minimizing the makespan. In a similar context, which is more applicable to the cloud computing context, several heuristics based on list scheduling were proposed (Shirvani and Talouki [124]).

The contribution that is closest to our work is probably the one by Höttger et al. [99], in which the partitioning of automotive applications is addressed by:

1. allocating runnables into tasks based on their activation pattern;
2. creating DAGs based on the shared labels;
3. mapping DAGs finally onto the CPUs.

Performing such a grouping of runnables based on the activation pattern, however, does not necessarily lead to a lower resource utilization. Also, mapping DAGs injects an unnecessary level of complexity which prevents the applicability to large applications.

2.2.3 Specific environments

Other methods presented in the past operate in the context of specific architectures or programming paradigms.

Kobayashi et al. [100] proposed the Model-Based Parallelizer (MBP), which maps C applications generated by Simulink's Embedded Coder onto Kalray's MPPA2-256, which is composed by 16 clusters of 16 cores. Such a parallel architecture, however, poses different challenges than the ones arising from the heterogeneous multicore architectures normally used in the automotive context, such as Infineon's TriCore.

In Becker et al. [77] is proposed to partition memories into banks and to schedule accesses to the same bank at different instants. Pazzaglia et al. [119] implemented a partitioning scheme that includes memory allocation using the Logical Execution Time (LET) paradigm. In the LET paradigm, tasks have to meet intermediate deadline at which the communication between tasks happens. The same paradigm was previously proposed by Henzinger et al. [98].

Methods that use Constraint Programming have been proposed, without exploring the complex dimension of the mapping of the variables to memory. Perret et al. [120] proposed a Constraint Programming approach to map a large application over a massively parallel architecture. An IP formulation of the runnable-to-task-to-core mapping was proposed by Saidi et al. [122]. However, the addressed use case was two orders of magnitude smaller than our target automotive application. Optimal partitioning and priority assignment using mathematical optimization is also proposed by Zhao and Zeng [132] and by Casini et al. [81].

Methods to adapt the application partitioning to dynamic workload do exist (Paul et al. [118]). Fernandez et al. [91] identified the “cyclic dependency” between the execution cycles of tasks and their partitioning (later represented in Fig. 8.3) as one of the main challenges. They proposed to break this dependency by establishing ties between tasks of varying strength. However, none of these works explored the mapping of labels with awareness of NUMA.

Voronov et al. [127] considered the analysis of applications over heterogeneous platforms including hardware accelerators; however, the mapping was not addressed. In Wang et al. [128] is addressed the full “V-cycle” of the development of automotive applications. Still, the partitioning of runnables and labels is performed through metaheuristics.

Partitioning automotive applications onto multicore platforms implicitly assumes that the size of the largest runnable is small enough to fit any core. However, as cores are growing in number but getting smaller and smaller, while runnables are growing in complexity, we can imagine a scenario in which single runnables may need to be split. In Lowinski et al. [107] this problem is addressed.

The problem of priority assignment was also addressed in the literature. Mancuso et al. [109] developed an efficient B&B algorithm for assigning priority to tasks. However, their proposed model was customized for control applications without shared data.

2.2.4 Generalization of memory hierarchy

An efficient memory hierarchy and interconnection is crucial in multicore architectures. The necessity of this efficiency has led to the growth of non-standardized architectures in which every vendor proposes a custom solution. The formulation of the optimal placement of embedded applications in presence of non-standard hardware platforms is not ideal as it would require to re-do part of the modeling every time the architecture changes.

In the attempt to generalize typical memory hierarchies, Hamann et al. [95] proposed a memory model split into Local RAM (LRAM) and Global RAM (GRAM) with access times as follows:

- one cycle from a core to its own LRAM;
- nine cycles from a core to other cores' LRAM;
- nine cycles from any core to GRAM.

At hardware level, due to the complexity of COTS (Commercial, Off-The-Shelf) Dynamic Random Access Memory (DRAM) controllers, a significant effort was made to make predictable DRAM controllers.

- Reineke et al. [121] proposed to partition memory banks among the requesting tasks to avoid “by construction” conflicts and then unpredictable interference.
- Time-Division Multiplexing is also a technique used to guarantee predictability of memory accesses. To mitigate the poor worst-case performance of TDM, Hebbache et al. [97] proposed a dynamic arbitration scheme that improves memory utilization, still keeping the worst-case guarantees of TDM.

The proliferation of various memory controllers challenged a comparison among the different approaches. For this reason, MCsim was proposed (Mirosanlou et al. [112]), an extensible cycle-accurate DRAM memory controller simulator. MCsim is able to run as a trace-based simulator as well as provide an interface to connect with external CPU and memory device simulators.

2.2.5 Related patents

Commercially, there are patents that cover some of the steps that are needed for optimization, but in very general terms and not with a realistic modeling of the memory costs. The patents apply to a specific method for allocating tasks to cores, which does not control over the use of the memory resources (and therefore the actual runnable execution time); see, for example, Noriaki et al. [115] and Guan and Tong [94].

Other methods consider the optimization of the memory allocation of data and code (Maspoli et al. [110], Liping [104]) by assuming a given assignment of runnables to tasks and tasks to CPUs. Instead, the originality of our contribution is in the capacity to address all dimensions of the mapping (both runnables and labels) and to achieve superior results with a run-time orders of magnitude smaller than existing meta-heuristic-based approaches.

2.2.6 Contribution of this work

Summarizing, the large majority of related works have ignored the impact of the placement of shared variables over the NUMA architectures. The few works which have considered this additional dimension, have proposed a GA formulation. GAs, however, ignore the peculiarities of the mapping of embedded applications and then hinder a full understanding of the role played by the many available tuning parameters of the model.


Hence, to the writer's knowledge, our IP approach is the first one proposing a tractable problem formulation that wholly addresses the mapping of the runnables over the available cores, the mapping of the labels over the NUMA architecture, the aggregation of runnables into tasks, and the assignment of priority to tasks. Compared to GA formulations, our approach has the advantage of being computationally more efficient and more transparent to the designer, and it enables interactive design space exploration through the linear constraints and cost which can be tuned at design time.

This work, published as Druetto et al. [87], has been presented to the 31st International Conference on Real-Time Networks and Systems (RTNS), held in June 2023. Also, a patent (International Patent Application PCT/EP2022/063829) is pending over the developed technique, named *SOFTWARE OPTIMIZATION METHOD AND DEVICE FOR NUMA ARCHITECTURE*.

Part I

Flow models for parallel batching

Problem description

HE first part of my PhD was dedicated to the study of a difficult family of parallel batch scheduling problems and to the development of efficient algorithms that are able to deliver the new state-of-the-art bounds, very tight heuristics and even an exact approach for the simpler version of the problem.

Here follows a recap on the notation that will be used throughout this entire part.

Tab. 3.1: Notation summary for Parallel Batching.

term	description
n	number of jobs
m	number of machines
n_f	number of families
$b (b_i)$	batch capacity (i -th capacity)
$N = \{1, 2, \dots, n\}$	set of n jobs to be scheduled
p_j	processing time of job j
$s_j (s_{ij})$	size of job j (i -th size of job j)
w_j	weight of job j
$S = (B_1, B_2, \dots, B_t)$	batch sequence of t batches
$p_B = \max\{p_j : j \in B\}$	processing time of batch B
$C_{B_k} = \sum_{l=1}^k p_{B_l}$	completion time of k -th batch
$C_j = C_B, \forall j \in B$	completion time of job j

The considered objective function is the *total completion time*, also called *total flow time*, described in Section 2.1.4. This calls to minimize $\sum C_j$, the sum of completion times for all jobs j , considering that a job is completed when the containing batch completes, the processing time for each batch equals to the maximum processing time among all contained jobs, and completion time for a batch must consider all preceding batches in the sequence.

No assumptions are made on the value of size, weight and processing time parameters for each job. Nevertheless, following what has already been done in the literature, these values are sampled from various distributions and potentially different across all jobs. The only fixed values between different problem categories are the maximum batch capacity and the number of machines (where applicable); both values are, like the others before, taken from the literature. Details on the specific values of these parameter will be given during the tractation of relative chapters.

3.1 Unweighted total completion time: exact approach and parallel machines

Chapter 4 deals with the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ scheduling problem, where jobs are scheduled in batches on a single machine in order to minimize the total completion time. A size is given for each job, such that the total size of each batch cannot exceed a fixed capacity b .

A graph-based model is proposed for computing a very effective lower bound based on linear programming; the model, with an exponential number of variables, is solved by Column Generation (CG), and embedded into both a heuristic Price-and-Branch (P&B) algorithm and an exact Branch-and-Price (B&P) algorithm. The same model is able to handle efficiently parallel machine problems like $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$.

Computational results show that the new lower bound strongly dominates the bounds currently available in the literature, and the proposed heuristic algorithm is able to achieve high quality solutions on large problems in a reasonable computation time. For the single-machine case, the exact B&P algorithm is able to solve all the tested instances with 30 jobs, and a good amount of 40-jobs examples.

3.2 Weighted total completion time: heuristics for the single machine case

Chapter 5, extending the work described in Chapter 4, deals with the single-machine, parallel batching, total weighted completion time scheduling problem; that is, problem $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$.

A new graph-based formulation of the problem is proposed, where such graph has an exponential number of nodes and arcs. The problem is modeled as a Mixed-Integer Program (MIP) on this graph, combining features of a minimum cost flow problem and features of a set-partition problem as well. The continuous relaxation of this integer problem is solved via CG, providing a very tight lower bound. Two different flavors of CG are tested, leading to two models with identical performances in terms of bound tightness but in practice very different in terms of running time.

A simple and effective rounding strategy applied to the faster model allows to generate heuristic solutions with values within a few percentage points from the optimum. Two different variants of the heuristic rounding procedure are tested; one allows to certify the optimality gap, while the other trades the ability to certify the gap with a greater computational speed.

3.3 Unweighted total completion time: heuristics for multi-size jobs and incompatibility families

Chapter 6, extending in another direction the work described in Chapter 4, deals with the single-machine, parallel batching, total completion time scheduling problem, in the presence of multi-size jobs and incompatibility families; that is, problem $1|p\text{-batch}, b_i, \sigma_{ij}, \text{incomp}| \sum C_j$.

Parallel batch scheduling has many applications in the industrial sector, like in material and chemical treatments, mold manufacturing and so on. The number of jobs that can be processed on a machine mostly depends on the shape and size of the jobs and of the machine.

This work investigates the problem of batching jobs with multiple sizes and multiple incompatible families. A flow formulation of the problem is exploited to solve it through two CG heuristics. First, the CG finds the optimal solution of the continuous relaxation, then two heuristics are proposed to move from the continuous to the integer solution of the problem: one is based on the P&B heuristic, the other on a variable rounding procedure.

Experiments with several combinations of parameters are provided to show the impact of the number of sizes and families on computation times and quality of solutions.

3.4 Unweighted total completion time: analysis of two polynomial-size models

Chapter 7 presents a different modeling approach to the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem already tackled in Chapter 4, with the development and analysis of two MIP formulations.

In this work, we present two new integer linear formulations for the problem of minimizing the total completion time on a single parallel-batching machine. One of the two new formulations is strong (in the sense that it delivers a sharp lower bound) and compact (that is, polynomial in size), contrasted to recent successful models (see Chapter 4 and Chapter 5) for the same problem that have exponential size and require to be handled by CG techniques.

The new model is promising: combined with a rounding procedure, it allows to deliver good solutions with small, certified optimality gaps for instances with up to 50 jobs, and we believe it is susceptible of further improvements.

Unweighted total completion time: exact approach and parallel machines



IN this chapter, the parallel batching problem with single and multiple parallel machines is considered. Specifically, given a set of jobs all available at the same time, how to partition them in batches and how to sequence batches on machines is addressed with the objective of minimizing the total completion time.

With respect to the current literature, the problem addressed in the paper is the same problem as Rafiee Parsa et al. [61] with the main difference that it is extended to the parallel machines case. Following the three field notation by Graham et al. [31] described in Tab. 2.1, the problems studied in this chapter are $1|p\text{-batch}, b, \sigma_j| \sum C_j$ and $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$.

The contribution of this chapter is threefold.

- A new graph-based model for $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem is developed; such model induces a very large Linear Program (LP) with an exponential number of variables, which can be handled by Column Generation (CG) techniques. The pricing step is efficiently solved by Dynamic Programming (DP). The new model provides the strongest linear relaxation currently available in literature for the studied problem.
- A heuristic procedure of the so-called Price-and-Branch (P&B) type, following the terminology of Desrosiers and Lübbecke [18], relying on the graph model is developed. Such procedure allows to generate high quality solutions (with certified optimality gaps) for fairly large instances in short computation times. The CG procedure is also embedded in an exact Branch-and-Price (B&P) procedure that is able to deliver optimal solutions for considered instances with up to 40 jobs. Previous state-of-the-art results in Azizoglu and Webster [5] were limited to 25 jobs.
- The graph-based model and the related CG procedure and heuristic are extended also to the $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$ problem, in an environment with multiple parallel machines. Computational experience shows that heuristic solutions for up to 5 machines and 100 jobs can be easily obtained by the

proposed approach with no significant loss of solution quality with respect to the single machine case.

4.1 Single-machine models

This section deals with the single-machine problem. Recalling the notation introduced in Chapter 3, we define $N = \{1, 2, \dots, n\}$ as the set of jobs to be scheduled; for each job $j \in N$, its processing time p_j and its size s_j , both integers, are given. The machine has a given integer capacity denoted by b . When a subset of jobs is packed in a batch B , $p_B = \max\{p_j : j \in B\}$ is used to indicate the batch processing time. Every batch B is required to have $\sum_{j \in B} s_j \leq b$. The machine processes the jobs in a batch sequence $S = (B_1, B_2, \dots, B_t)$, where each job j in the k -th batch B_k shares the batch completion time: $C_j = C_{B_k} = \sum_{l=1}^k p_{B_l}$, $\forall j \in B_k$. The $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem calls for creating the batches and sequencing them in order to minimize $f(S) = \sum_{j \in N} C_j$.

A Mixed-Integer Program (MIP) model for this problem, as given by Rafiee Parsa et al. [61], is the following, where variable $x_{jk} = 1$ iff job j is scheduled in the k -th batch; the model always arranges the jobs in n batches (B_1, B_2, \dots, B_n) some of which can be empty. Other variables of the model are: the completion time C_{B_k} of the k -th batch (note that the model allows for empty batches); the variable p_{B_k} that represents the processing time of the k -th batch; the variable C_j corresponding to the completion time of job j .

$$\text{minimize } \sum_{j=1}^n C_j \quad (4.1)$$

$$\text{subject to } \sum_{k=1}^n x_{jk} = 1 \quad j = 1, \dots, n \quad (4.2)$$

$$\sum_{j=1}^n s_j x_{jk} \leq b \quad k = 1, \dots, n \quad (4.3)$$

$$p_{B_k} \geq p_j x_{jk} \quad j = 1, \dots, n, k = 1, \dots, n \quad (4.4)$$

$$C_{B_1} \geq p_{B_1} \quad (4.5)$$

$$C_{B_k} \geq C_{B_{k-1}} + p_{B_k} \quad k = 2, \dots, n \quad (4.6)$$

$$C_j \geq C_{B_k} - M(1 - x_{jk}) \quad j = 1, \dots, n, k = 1, \dots, n \quad (4.7)$$

$$p_{B_k}, C_{B_k}, C_j \geq 0 \quad j = 1, \dots, n, k = 1, \dots, n \quad (4.8)$$

$$x_{jk} \in \{0, 1\} \quad j = 1, \dots, n, k = 1, \dots, n \quad (4.9)$$

The total completion time is expressed by (4.1). Constraint set (4.2) ensures that each job is assigned exactly to one batch and, since all the jobs assigned to a batch

cannot exceed the batch capacity, constraint set (4.3) has to be defined. Constraint set (4.4) represents the fact that the processing time of a batch is the maximum processing time of all the contained jobs. The completion time for the first batch is simply its processing time since it is the first to be processed by the machine, as stated in constraint (4.5). Constraint set (4.6), instead, ensures that the completion time for all the other batches is evaluated as the sum of its processing time and the completion time of the previous batch. Constraint set (4.7) specifies that the completion time of a job must be the completion time of the corresponding batch (the constant M must be very large).

Model (4.1)–(4.9) is known to be very weak. A state-of-the-art solver like CPLEX can waste hours over 15-jobs instances, with optimality gaps of 100% at the root branching node.

4.1.1 A new problem formulation

In our work, we propose a new model where a batch sequence is represented as a path on a graph. Let $\mathcal{B} = \{B \subseteq N : \sum_{j \in B} s_j \leq b\}$ be the set of all the possible batches. Define a multi-graph $G(V, A)$ with vertex and arc sets as follows:

$$V = \{1, 2, \dots, n + 1\}, \quad (4.10)$$

$$A = \{(i, k, B) : i, k \in V; i < k; B \in \mathcal{B}; |B| = k - i\}. \quad (4.11)$$

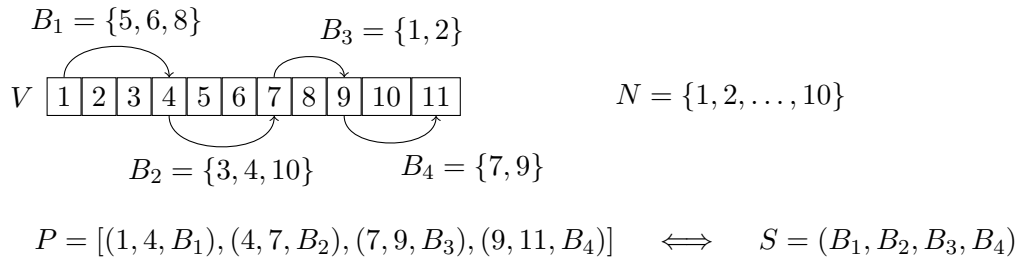
Each arc in A is a triple (i, k, B) with head k and tail i and an associated batch B with $(k - i)$ jobs; it will represent the batch B scheduled in a batch sequence such that exactly $n - i + 1$ jobs are scheduled from batch B up to the end of the sequence. For each arc (i, k, B) , a cost is defined as $c_{ikB} = (n - i + 1)p_B$ with $p_B = \max\{p_j : j \in B\}$, and where the $(n - i + 1)$ factor precisely models the above mentioned $(n - i + 1)$ jobs to the end of the sequence. Note that, in G , a path

$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \dots, (i_r, k_r, B_r)] \quad (i_\ell = k_{\ell-1})$$

connecting nodes from i_1 to k_r has the following property:

$$\begin{aligned} \sum_{\ell=1}^r |B_\ell| &= \sum_{\ell=1}^r (k_\ell - i_\ell) = \sum_{\ell=1}^r k_\ell - \sum_{\ell=1}^r i_\ell \\ &= \sum_{\ell=1}^r k_\ell - \sum_{\ell=2}^r k_{\ell-1} - i_1 = k_r - i_1. \end{aligned} \quad (4.12)$$

Property 1 highlights the relationship between feasible batches and paths of the above defined graph.



$$\begin{aligned}
 \sum_{j=1}^{10} C_j(S) &= \sum_{\kappa=1}^4 C_{B_\kappa}(S) |B_\kappa| \\
 &= |B_1| p_{B_1} + \\
 &\quad |B_2| (p_{B_1} + p_{B_2}) + \\
 &\quad |B_3| (p_{B_1} + p_{B_2} + p_{B_3}) + \\
 &\quad |B_4| (p_{B_1} + p_{B_2} + p_{B_3} + p_{B_4}) \\
 &= 10p_{B_1} + 7p_{B_2} + 4p_{B_3} + 2p_{B_4} = \sum_{(i,k,B) \in P} (n-i+1)p_B = \sum_{(i,k,B) \in P} c_{ikB}
 \end{aligned}$$

Fig. 4.1: Batch sequence as a path on a graph. The job set N is partitioned over the arcs of P . $C_j(S)$ is the completion time of job j induced by sequence S . The path provides information about partitioning the jobs into batches B_1, B_2, B_3, B_4 and their ordered sequence.

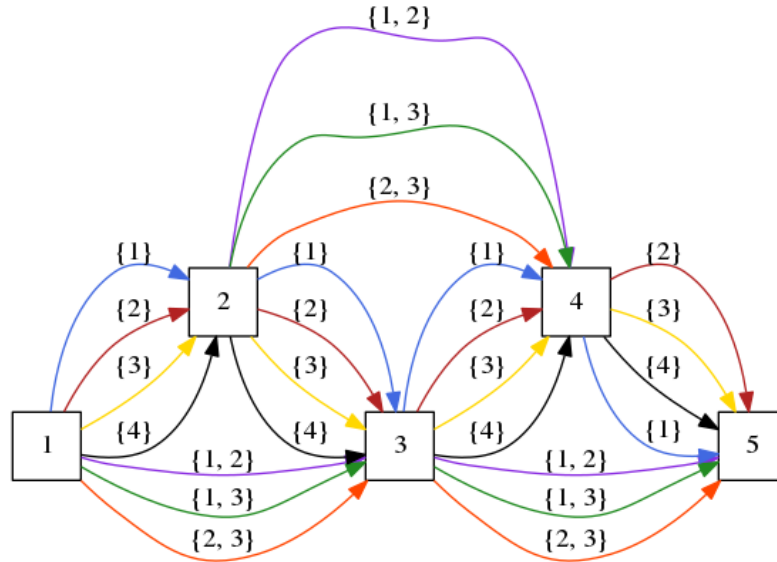


Fig. 4.2: Example of a full graph with $b = 10$ and 4 jobs. Job sizes are $s_1 = 3, s_2 = 4, s_3 = 6, s_4 = 8$.

Example

Fig. 4.2 shows an example of a full graph with all the possible batches. In this small 4-jobs example, $p_1, \dots, p_4 = 42, 37, 21, 16$, $s_1, \dots, s_4 = 3, 4, 6, 8$ and $b = 10$; batches can contain 1 or 2 jobs at most. Note how the graph size is already quite large, even for this small example.

A path from node 1 to node 5 in this graph represents a feasible schedule if the job set $\{1, 2, 3, 4\}$ is partitioned over the arcs of the path. For example, the path $P = [(1, 2, \{4\}), (2, 4, \{1, 3\}), (4, 5, \{2\})]$ represents the feasible batch sequence $S = (\{4\}, \{1, 3\}, \{2\})$. In such sequence, the reader can easily compute that $C_4 = 16$, $C_1 = C_3 = 58$, $C_2 = 95$, and $C_1 + C_2 + C_3 + C_4 = 227$; in the path P , the arc costs are, by definition, $c_{(1,2,\{4\})} = 4p_4 = 64$, $c_{(2,4,\{1,3\})} = 3p_1 = 126$, $c_{(4,5,\{2\})} = p_2 = 37$; the total cost of P is $c_{(1,2,\{4\})} + c_{(2,4,\{1,3\})} + c_{(4,5,\{2\})} = 227$.

On the other hand, a path like $P' = [(1, 2, \{3\}), (2, 4, \{1, 3\}), (4, 5, \{3\})]$ does not represent a feasible batch sequence since it fails to partition the job set $\{1, 2, 3, 4\}$ over its arcs.

Model

By Property 1, then, an optimal solution for the $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem can be computed by identifying on the very large graph $G(V, A)$ a minimum-cost path from node 1 to node $n + 1$ such that the job set N is exactly partitioned over the “ B ” components of the arcs in that path. This problem can be modeled by a very large LP that includes features of a shortest path / minimum cost flow model as well as partition constraints, as follows. Let $\mathbf{a}_B \in \{0, 1\}^n$ be the incidence column-vector of job set B , whose j -th component $(\mathbf{a}_B)_j = 1$ iff $j \in B$, and let $\mathbf{1} = (1, 1, \dots, 1)^\top$ be an all-ones column-vector with exactly n components. Define binary decision variables x_{ikB} for each $(i, k, B) \in A$, so that $x_{ikB} = 1$ iff arc (i, k, B) is on the optimal path from 1 to $(n + 1)$. The LP is written as follows.

$$\text{minimize } \sum_{(i,k,B) \in A} c_{ikB} x_{ikB} \quad (4.15)$$

$$\text{subject to } \sum_{\substack{(k,B): \\ (i,k,B) \in A}} x_{ikB} - \sum_{\substack{(k,B): \\ (k,i,B) \in A}} x_{kiB} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n + 1 \end{cases} \quad (4.16)$$

$$\sum_{(i,k,B) \in A} \mathbf{a}_B x_{ikB} = \mathbf{1} \quad (4.17)$$

$$x_{ikB} \in \{0, 1\} \quad (i, k, B) \in A \quad (4.18)$$

The objective function (4.15) together with the flow conservation constraints (4.16) are a classical formulation of the shortest path problem as a special case of single-source single-sink minimum-cost flow LP; see for example Ahuja et al. [1]. The vector expression of constraint (4.17) represents a group of n set-partitioning constraints on the job set $N = \{1, 2, \dots, n\}$, enforcing the requirement that the job set is exactly partitioned over the arcs selected to be in the path. In scalar form, these constraints are: $\sum_{(i,k,B) \in A} (\mathbf{a}_B)_j x_{ikB} = 1, \forall j \in N$.

4.1.2 Continuous relaxation for the new graph-based formulation: Column Generation

The continuous relaxation of (4.15)–(4.18), where the integrality constraints (4.18) are relaxed to

$$x_{ikB} \geq 0 \quad (i, k, B) \in A, \quad (4.19)$$

is solved by means of a CG procedure. More details on CG can be found in Section 5.1.1.

Model (4.15)–(4.19) is the master problem: a Restricted Master Problem (RMP) is made of a subset $A' \subset A$ of arcs. Introducing dual variables u_1, u_2, \dots, u_{n+1} for constraints (4.16) and v_1, \dots, v_n for constraints (4.17), the dual of (4.15)–(4.19) is

$$\text{maximize } u_1 - u_{n+1} + \sum_{j=1}^n v_j \quad (4.20)$$

$$\text{subject to } u_i - u_k + \sum_{j \in B} v_j \leq c_{ikB} \quad (i, k, B) \in A. \quad (4.21)$$

Solving the RMP leads to a basic feasible solution for the master problem and values for dual variables/simplex multipliers \mathbf{u} and \mathbf{v} . Pricing the arcs $(i, k, B) \in A$ corresponds to finding the most violated dual constraints (4.21). The strategy developed in this paper is to price the arcs separately for each pair of indices i, k with $i < k$, therefore determining minimum (possibly negative) reduced costs

$$\begin{aligned} \bar{c}_{ikB^*} &= \min_B \left\{ c_{ikB} - (u_i - u_k) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b, |B| = k - i \right\} \\ &= \min_B \left\{ p_B(n - i + 1) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b, |B| = k - i \right\} - (u_i - u_k). \end{aligned} \quad (4.22)$$

For fixed indices i, k with $i < k$, the $(u_i - u_k)$ part of (4.22) is constant, and the cardinality of B is also fixed at $|B| = k - i$.

Finding the batch B that minimizes (4.22), for each given pair of indices i, k with $i < k$ and given batch processing time p_B , can be done by exploiting the DP

state space of a family of cardinality-constrained knapsack problems where items correspond to jobs. Assume that the jobs are indexed by Longest Processing Time (LPT) order, so that

$$p_1 \geq p_2 \geq \dots \geq p_n.$$

Define, for $r = 1, \dots, n$,

$$g_r(\tau, \ell) = \max \left\{ \sum_{j=r}^n v_j y_j : \sum_{j=r}^n s_j y_j \leq \tau, \sum_{j=r}^n y_j = \ell, y_j \in \{0, 1\} \right\},$$

where $g_r(\tau, \ell)$ is the optimal value of a knapsack with profits v_j and sizes s_j , limited to items/jobs $r, r+1, \dots, n$, total size $\leq \tau$ and cardinality exactly ℓ . Variable y_j is set to 1, then $y_j = 1$, iff item/job j is included in the solution.

Optimal values for $g_r(\tau, \ell)$ can be recursively computed (see Kellerer et al. [36]) as

$$g_r(\tau, \ell) = \max \begin{cases} g_{r+1}(\tau - s_r, \ell - 1) + v_r & (y_r = 1) \\ g_{r+1}(\tau, \ell) & (y_r = 0) \end{cases}$$

with boundary conditions

$$\begin{aligned} g_r(\tau, 1) &= \begin{cases} v_r & \text{if } s_r \leq \tau \text{ (} y_r = 1 \text{)} \\ 0 & \text{otherwise (} y_r = 0 \text{)} \end{cases} & r = 1, \dots, n, \tau = 0, \dots, b \\ g_r(\tau, 0) &= 0 & r = 1, \dots, n, \tau = 0, \dots, b \\ g_r(\tau, \ell) &= -\infty & \text{if } \ell > n - r + 1 \text{ or } \tau < 0. \end{aligned}$$

The corresponding optimal job sets are denoted by $B_r(\tau, \ell)$; such sets can be retrieved by backtracking. The following property establishes that the state space $g_r(\tau, \ell)$ is sufficient for pricing all the relevant arcs.

Property 2. Let $L = \{1\} \cup \{j > 1 : p_j < p_{j-1}\}$. For any given pair of indices i, k with $i < k$, an arc with minimum reduced cost (i, k, B^*) is one of

$$(i, k, B_r(b, k - i)) \quad r \in L. \quad (4.23)$$

Proof. Every arc (i, k, B) can be shown to have a reduced cost not less than some of the arcs in (4.23). Let $\bar{c}_{ikB} = (n - i + 1)p_B - \sum_{j \in B} v_j - (u_i - u_k)$ be the reduced cost of an arc (i, k, B) . Recall that $|B| = k - i$, and the jobs are numbered in non-increasing order of processing times. Choose r as the smallest job index such that $p_r = p_B$. Note that $B \subseteq \{r, r+1, \dots, n\}$ and $r \in L$. Consider knapsack $g_r(b, k - i)$ and the associated optimal subset $B_r = B_r(b, k - i)$. The batch B is a feasible solution for

knapsack $g_r(b, k - i)$, hence $\sum_{j \in B} v_j \leq g_r(b, k - i)$; also, because of the choice of r , $p_{B_r} \leq p_r = p_B$. Thus,

$$\begin{aligned} \bar{c}_{ikB} &= (n - i + 1)p_B - \sum_{j \in B} v_j - (u_i - u_k) \geq \\ &\geq (n - i + 1)p_{B_r} - g_r(b, k - i) - (u_i - u_k) = \bar{c}_{ikB_r}. \end{aligned}$$

□

All the relevant arcs with minimum reduced cost can be generated by the procedure reported in Algorithm 1 (NEWCOLS).

Algorithm 1 Pricing procedure.

```

1: function NEWCOLS( $N, b, \mathbf{u}, \mathbf{v}$ ) ▷  $\mathbf{u}, \mathbf{v}$  = vectors of multipliers
2:   Sort and renumber jobs in  $N$  such that  $p_1 \geq p_2 \geq \dots \geq p_n$ ;
3:   Set  $H := \emptyset$ ; ▷ set of negative-reduced cost arcs
4:   for  $\ell = 1, \dots, n$  do
5:     Set  $r := 1$ ,  $done := \mathbf{false}$ ;
6:     while not  $done$  do
7:       Retrieve  $g_r(b, \ell)$  and set  $B := B_r(b, \ell)$ ;
8:       for  $i = 1, \dots, n - \ell + 1$  do
9:         Set  $k := i + \ell$ ;
10:        Compute  $\bar{c}_{ikB} = p_B(n - i + 1) - (u_i - u_k) - g_r(b, \ell)$ ;
11:        if  $\bar{c}_{ikB} < 0$  then
12:          Set  $H := H \cup \{(i, k, B)\}$ ;
13:        end if
14:      end for
15:      Set  $r := \min\{j : p_j < p_r\}$ ;
16:      If no such index exists, set  $done := \mathbf{true}$ ;
17:    end while
18:  end for
19:  return  $H$ ;
20: end function

```

The size of the state space required for the pricing is bounded by $\mathcal{O}(n^2b)$, while the pricing procedure can have two bottlenecks.

- The $\mathcal{O}(n^3)$ effort due to the three nested loops on line 4, line 6, and line 8. The **while** loop on line 6 can be executed n times in the worst case.
- Filling the state space $g_r(\tau, \ell)$, which requires at most $\mathcal{O}(n^2b)$ arithmetic operations. A *memoized* DP table is used, so that the execution of the top-down recursion for computing an entry $g_r(\tau, \ell)$ is deferred until the first time the value is queried. Then, the value is kept in storage and accessed in $\mathcal{O}(1)$ time if it is queried again.

Because of these two possible bottlenecks, the running time of NEWCOLS is bounded from above by $\mathcal{O}(\max(n^3, n^2b))$.

4.1.3 Heuristic procedure: Price-and-Branch

The CG described in the previous section is used to solve the continuous relaxation of the master problem; once the relaxed optimum has been found, the resulting RMP is taken, the variables are set to binary type and the resulting MIP is solved by using CPLEX in order to get a heuristic solution for the master. This is often called Price-and-Branch (P&B), as opposed to the exact approach of Branch-and-Price (B&P).

In order to generate the initial column set, the jobs are sorted in Shortest Processing Time (SPT) order and all the possible arcs with feasible batches made of consecutive jobs are generated. This procedure is reported in Algorithm 2 (INITCOLS).

Algorithm 2 Generation of initial arcs.

```
1: function INITCOLS( $N, b$ )
2:   Sort and renumber jobs in  $N$  such that  $p_1 \leq p_2 \leq \dots \leq p_n$ ;
3:   Set  $H := \emptyset$ ; ▷ set of initial arcs
4:   for  $j = 1, \dots, n$  do
5:     Set  $B := \{j\}$ ;
6:     for  $h = j + 1, \dots, n$  do
7:       if  $\sum_{j \in B} s_j + s_h \leq b$  then
8:         Set  $B := B \cup \{h\}$ ;
9:         Set  $H := H \cup \{(i, i + |B|, B) : i = 1, \dots, n - |B| + 1\}$ ;
10:      end if
11:    end for
12:  end for
13:  return  $H$ ;
14: end function
```

The complete P&B heuristic procedure then is able to deliver a lower bound, called Column Generation Lower Bound (CG-LB), and an upper bound, called Column Generation Upper Bound (CG-UB). A sketch of this procedure can be read in Algorithm 3.

4.1.4 Exact approach: Branch-and-Price

Given the strong relaxation from the CG procedure, a natural step is trying to embed it in an exact B&P algorithm. The main issue in this step is to be able to preserve the pricing problem structure at every node in the search tree. Trying to use the classical branching scheme from Foster and Ryan [27], which leverages the partitioning constraints forcing pairs of jobs to be batched always together/never together, would require to handle disjunctive constraints in the pricing problem. This would make the latter a strongly NP-hard disjunctive knapsack problem, ruling out the possibility of using the DP procedure of Algorithm 1 (unless $P = NP$).

Algorithm 3 P&B procedure.

```
1:  $A' \leftarrow \text{INITCOLS}(N, b)$ ;  
2:  $G(V, A') \leftarrow \text{Restricted Master Problem (RMP)}$ ;  
3: while true do  
4:    $z \leftarrow \text{continuous optimum of } G(V, A')$ ;  
5:    $\mathbf{u}, \mathbf{v} \leftarrow \text{optimal dual multipliers}$ ;  
6:    $H \leftarrow \text{NEWCOLS}(N, b, \mathbf{u}, \mathbf{v})$ ;  
7:   if  $|H| = 0$  then  
8:     CG-LB  $\leftarrow z$  — continuous optimum, lower bound  
9:     CG-UB  $\leftarrow \text{CPLEX}(G(V, A'))$  — integer solution, upper bound  
10:    break  
11:  end if  
12:   $A' \leftarrow A' \cup H$   
13: end while
```

Still, branching can be performed on a compact formulation of the problem, building the batch sequence by scheduling one job at a time starting from the first position on. The basic branching mechanism adopted here is the same described in Uzsoy [69] and Azizoglu and Webster [5]. Let $S = (B_1, B_2, \dots, B_t)$ be a partial (possibly empty) batch sequence built at the current search node, and $\hat{N} = N \setminus (B_1 \cup B_2 \cup \dots \cup B_t)$ the set of unscheduled jobs at such node. If the node is not fathomed by bound, two types of branch can take place.

- I A new unscheduled job $j \in \hat{N}$ is added to B_t , provided that there is still available space, and $\sum_{i \in B_t} s_i + s_j \leq b$.
- II Batch B_t is closed and a new one B_{t+1} is started, choosing its longest job among the $j \in \hat{N}$.

New jobs are added to the open batch in non-increasing order of processing time; hence, if a job j has been added to B_t no other job j' with $p_{j'} > p_j$ will enter the same batch in successive branches. Also, the batch B_t is closed only if it is maximal: as far jobs can be added to B_t without exceeding the capacity b , this will prevent type II branches from the current node.

The CG based relaxation is solved at each node of the search tree; batches from the partial batch sequence $(B_1, B_2, \dots, B_{t-1})$ correspond, in the relaxation, to arc-variables fixed at 1. At non-root nodes, the “open” batch B_t at the end of the partial sequence is handled by imposing constraints on the state space to be searched in the pricing step. Three things can be noticed.

- Only items corresponding to jobs in $j \in \hat{N}$ concur to form the state space.
- For pricing arcs (i, k, B) with $i = \sum_{\kappa=1}^{t-1} |B_\kappa|$, $B \supseteq B_t$ (these are arcs extending the “open” batch at the tail of the partial sequence) the items in B_t are

preloaded in the knapsack; hence, only states $g_r(\tau, \ell)$ with capacity $\tau \leq (b - \sum_{i \in B_t} s_i)$ and index $r > \max\{i : i \in B_t\}$ are solved.

- For all the other arcs, a pricing with full capacity b is performed.

The nodes in the search tree are expanded in depth-first order. Feasible solutions are generated at the root node by running the P&B procedure in Algorithm 3, and at the leaves of the search tree when the batch sequence is completed. Although Algorithm 3 implies running a potentially heavy exact procedure over a MIP, it is very fast for the tested problem sizes, and allows to achieve the highest-quality feasible solutions. Running quick-and-dirty heuristics at the intermediate nodes did not significantly improve the performances during preliminary testing.

4.2 Parallel-machines models

Model (4.15)–(4.18) is readily extended to parallel machines cases. Consider the fairly general $Rm|p\text{-batch}, b, \sigma_j| \sum C_j$ problem with m parallel unrelated machines. Let p_{jh} be the processing time of job j on machine h . A special type of arcs with empty batches is added to the graph developed for the single machine case, using the arc set

$$A = \{(i, k, B) : i, k \in V; i < k; B \in \mathcal{B}; |B| = (k - i)\} \cup \{(1, k, \emptyset) : k = 2, \dots, n + 1\}.$$

Arcs $(i, k, B) \in A$ are given machine-dependent costs $c_{ikB}^h = p_{Bh}(n - i + 1)$, with $p_{Bh} = \max\{p_{jh} : j \in B\}$. Empty arcs $(1, k, \emptyset)$ are given costs $c_{1k\emptyset}^h = 0$, for all $k = 2, \dots, n + 1$ and all $h = 1, \dots, m$.

Now equation (4.12) holds, in G , only for paths of *non-empty* arcs connecting nodes from i_1 to k_r ; that is, for

$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \dots, (i_r, k_r, B_r)] \quad B_1 \neq \emptyset$$

the equation $\sum_{l=1}^r |B_l| = k_r - i_1$ holds. Note that at most the first arc in a path can be empty: only if $i_1 = 1$ and $B_1 = \emptyset$.

Empty arcs are all added to the RMP from the beginning, so that they do not need to be considered in the DP pricing procedure. A feasible solution is made of m batch sequences

$$S_h = (B_1^h, \dots, B_{t_h}^h) \quad h = 1, \dots, m \quad (4.24)$$

processed by the m machines. Such batch sequences correspond to m paths (one path for each machine) from 1 to $n + 1$ on the non-empty arcs of which the set of jobs is exactly partitioned. Such paths will have an empty arc as first arc. Note that,

if (i, k, B) is on the h -th path, this means that $n - i + 1$ jobs will be scheduled from B to the end of the h -th batch sequence.

Property 1 can be easily extended to the multi-machine case. Fig. 4.3 reports a sketch of the proof with $m = 2$, where the empty arcs act as placeholders. The following Property 3 then formalizes the idea.

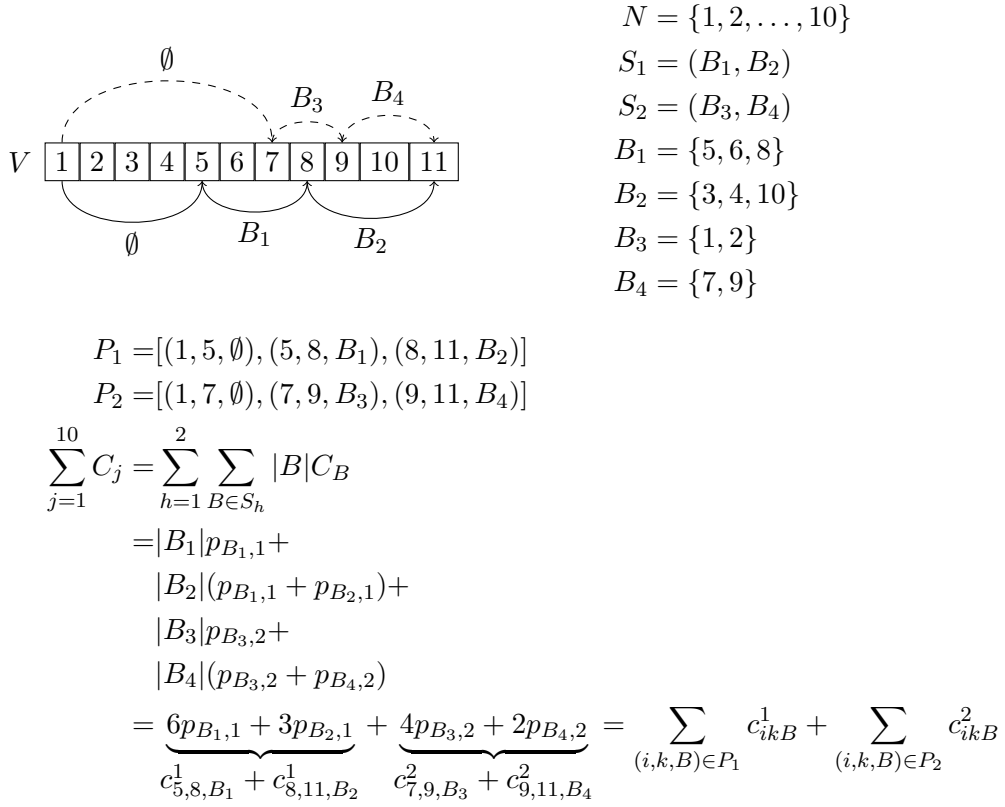


Fig. 4.3: Batch sequences on two machines as a collection of two paths on a graph. The job set N is partitioned into B_1, B_2, B_3, B_4 . The two paths provide a partition into batches and sequencing information.

Property 3. Each feasible set of batch sequences $\{S_h\}_{h=1}^m$ corresponds to a set of paths $\{P_h\}_{h=1}^m$ such that the job set N is partitioned over the non-empty arcs of $\{P_h\}_{h=1}^m$, and

$$\sum_{j \in N} C_j(S_1, \dots, S_m) = \sum_{h=1}^m \sum_{(i,k,B) \in P_h} c_{ikB}^h.$$

Proof. A one-to-one mapping between feasible solutions $\{S_h\}_{h=1}^m$ and collections of paths $\{P_h\}_{h=1}^m$ is quickly established.

Suppose that a feasible collection of batch sequences S_1, \dots, S_m is given. Take a machine h and its batch sequence $S_h = (B_1, \dots, B_t)$; note that the dependence on h

is dropped from the batch notation in order to keep it simple. The batch sequence S_h can be empty (the solution leaves machine h idle) or not.

If $S_h = \emptyset$, define $P_h = [(1, n + 1, \emptyset)]$ with a single empty arc. If $S_h \neq \emptyset$, by definition of the arc set A , the following arcs belong to the graph G :

$$\begin{array}{lll} (i_t, k_t, B_t) & k_t = n + 1, & i_t = k_t - |B_t| \\ (i_\ell, k_\ell, B_\ell) & k_\ell = i_{\ell+1}, & i_\ell = k_\ell - |B_\ell| \quad \ell = t - 1, \dots, 1 \\ (i_0, k_0, \emptyset) & k_0 = i_1, & i_0 = 1 \quad i_1 > 1. \end{array}$$

If $i_1 = 1$, arc (i_0, k_0, \emptyset) is omitted. The arcs are chosen so that $k_\ell = i_{\ell+1}$, $k_t = n + 1$ and the first arc has tail in node 1.

In both the cases addressed above, P_h identifies a path from node 1 to node $n + 1$. Repeat the construction for each machine $h = 1, \dots, m$ to get the collection of paths P_1, \dots, P_m ; the partition of the job set over the nonempty chosen arcs is guaranteed by the fact that the batches in $\{S_h\}_{h=1}^m$ already form a partition of N by hypothesis.

Vice versa, given a collection of paths P_1, \dots, P_m , all connecting node 1 to node $n + 1$, with the job set N partitioned over the non-empty arcs of such collection, take a machine index h , and let $P_h = [(i_0, k_0, B_0), (i_1, k_1, B_1), \dots, (i_t, k_t, B_t)]$. The batch sequence S_h is defined by $S_h = (B_1, \dots, B_t)$ if $B_0 = \emptyset$, else $S_h = (B_0, \dots, B_t)$. Note that, by definition of the arc set A , at most B_0 can be empty. If $P_h = [(1, n + 1, \emptyset)]$, $S_h = \emptyset$ and machine h is left idle. Repeat for each machine $h = 1, \dots, m$ in order to get a batch sequence for each machine. The feasibility of S_1, \dots, S_m is guaranteed by the fact that by hypothesis the job set N is partitioned over the set of non-empty arcs of P_1, \dots, P_m .

It remains to prove that $\sum_{j=1}^n C_j(S_1, \dots, S_m) = \sum_{h=1}^m \sum_{(i,k,B) \in P_h} c_{ikB}^h$. To this aim, it is sufficient to prove that on each machine h ,

$$\sum_{B \in S_h} \sum_{j \in B} C_j = \sum_{(ikB) \in P_h} c_{ikB},$$

then $\sum_{j \in N} C_j = \sum_{h=1}^m \sum_{B \in S_h} \sum_{j \in B} C_j$ and the result follows.

Consider machine h and sequence S_h with the corresponding path

$$P_h = [(i_0, k_0, B_0), (i_1, k_1, B_1), \dots, (i_t, k_t, B_t)],$$

where $i_0 = 1$ and $k_t = n + 1$. For the first arc (i_0, k_0, B_0) , either $B_0 = \emptyset$ or $B_0 \neq \emptyset$.

Assume $B_0 = \emptyset$. Let $p_{B_\ell, h}$ be the processing time of batch B_ℓ on machine h . The algebraic manipulations proceed similarly to the proof of Property 1.

Then, we have the following:

$$\begin{aligned}
\sum_{B \in S_h} \sum_{j \in B} C_j &= \sum_{q=1}^t |B_q| C_{B_q} \\
&= |B_1| p_{B_1, h} + \\
&\quad |B_2| p_{B_1, h} + |B_2| p_{B_2, h} + \\
&\quad |B_3| p_{B_1, h} + |B_3| p_{B_2, h} + |B_3| p_{B_3, h} + \\
&\quad \dots \dots \dots \\
&\quad |B_t| p_{B_1, h} + |B_t| p_{B_2, h} + |B_t| p_{B_3, h} + \dots + |B_t| p_{B_t, h} \\
&= p_{B_1, h} \sum_{\ell=1}^t |B_\ell| + p_{B_2, h} \sum_{\ell=2}^t |B_\ell| + p_{B_3, h} \sum_{\ell=3}^t |B_\ell| + \dots + p_{B_t, h} |B_t| \\
&= \sum_{q=1}^t p_{B_q, h} (n - i_q + 1) = \sum_{q=1}^t c_{i_q k_q B_q}^h = \sum_{(i, k, B) \in P_h} c_{ikB}^h,
\end{aligned}$$

where the last sum can be extended to all the arcs in P_h since for the first arc $(1, k_0, \emptyset) \in P_h$, $c_{1, i_1, \emptyset}^h = 0$.

If $B_0 \neq \emptyset$ then, by equation (4.12), $\sum_{\ell=0}^n |B_\ell| = k_t - i_0 = n$; hence, all the jobs of the problem are scheduled on machine h while the other machines must be left idle. The analysis is reduced to a single machine case and Property 1 ensures that $\sum_{B \in S_h} \sum_{j \in B} C_j = \sum_{(i, k, B) \in P_h} c_{ikB}^h$. \square

Model

The model (4.15)–(4.18) can be extended to the parallel machine case using multi-commodity flow constraints.

$$\text{minimize } \sum_{h=1}^m \sum_{(i, k, B) \in A} c_{ikB}^h x_{ikB}^h \tag{4.25}$$

$$\text{subject to } \sum_{\substack{(k, B): \\ (i, k, B) \in A}} x_{ikB}^h - \sum_{\substack{(k, B): \\ (k, i, B) \in A}} x_{kiB}^h = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n + 1 \end{cases} \quad h = 1, \dots, m \tag{4.26}$$

$$\sum_{h=1}^m \sum_{(i, k, B) \in A} \mathbf{a}_B x_{ikB}^h = \mathbf{1} \tag{4.27}$$

$$x_{ikB}^h \in \{0, 1\} \quad (i, k, B) \in A, \quad h = 1, \dots, m \tag{4.28}$$

Here $x_{ikB}^h = 1$ iff batch B is on the h -th path. Flow conservation constraints (4.26) require that one unit of each commodity is routed from node 1 to node $n + 1$.

Constraints (4.27) enforce the exact partition of the whole job set across the arcs belonging to the m paths.

In the CG framework, with each RMP optimum, constraint multipliers are computed:

$$\begin{aligned} u_1^h, u_2^h, \dots, u_{n+1}^h & \quad \text{for constraints (4.26), } h = 1, \dots, m, \\ v_1, v_2, \dots, v_n & \quad \text{for constraints (4.27).} \end{aligned}$$

The reduced cost is then separately minimized for each combination of pair of indices i, k with $i < k$ and machine h , searching for arcs (i, k, B) with reduced costs

$$\bar{c}_{ikB}^h = \min_B \left\{ p_{Bh}(n-i+1) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b, |B| = k-i \right\} - (u_i^h - u_k^h).$$

This requires calling Algorithm 1 NEWCOLS m times, once per machine, since the LPT ordering on each machine is different and, then, is the state space $g_r(\tau, \ell)$. Hence, the running time for pricing raises to $\mathcal{O}(m \max(n^3, n^2b))$.

Identical parallel machines

A somewhat better situation arises in the case of *identical* parallel machines, with problem $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$. Since each job j has the same processing time p_j on every machine, the state space $g_r(\tau, \ell)$ used for pricing is common to all the machines, and a slightly modified version of NEWCOLS can do the entire pricing, still keeping the running time within $\mathcal{O}(\max(n^3, n^2b))$. The procedure is reported in Algorithm 4.

The key observation is that the p_B and $\sum_{j \in B} v_j$ components of the reduced costs c_{ikB}^h are machine-independent, whereas only the largest difference $\Delta u_{ik} = \max_h \{(u_i^h - u_k^h)\}$ is strictly needed in order to compute minimum reduced costs. Such largest differences are precomputed in time $\mathcal{O}(mn^2)$ on line 3. For any (i, k, B) , let r be the smallest index such that $p_r = p_B$, and let $B_r = B_r(b, k-i)$; then, similarly to what proved in Property 2:

$$\begin{aligned} \bar{c}_{ikB}^h &= (n-i+1)p_B - \sum_{j \in B} v_j - (u_i^h - u_k^h) \geq \\ &\geq (n-i+1)p_{B_r} - g_r(b, k-i) - (u_i^h - u_k^h) \geq \\ &\geq (n-i+1)p_{B_r} - g_r(b, k-i) - \Delta u_{ik}. \end{aligned}$$

Finally, note that also taking into account different capacities for each machine, or even different job sizes on each machine, simply requires to specialize the used knapsack family. Details are omitted for the sake of conciseness.

Algorithm 4 Pricing procedure for identical parallel machines.

```
1: function NEWCOLS( $N, b, \mathbf{u}, \mathbf{v}$ ) ▷  $\mathbf{u}, \mathbf{v}$  = vectors of multipliers
2:   Sort and renumber jobs in  $N$  such that  $p_1 \geq p_2 \geq \dots \geq p_n$ ;
3:   Set  $\Delta u_{ik} := \max_h \{u_i^h - u_k^h\}$  for  $1 \leq i < k \leq n + 1$ ; ▷  $\mathcal{O}(mn^2)$  time
4:   Set  $H := \emptyset$ ; ▷ set of negative-reduced cost arcs
5:   for  $\ell = 1, \dots, n$  do
6:     Set  $r := 1, done := \mathbf{false}$ ;
7:     while not  $done$  do
8:       Retrieve  $g_r(b, \ell)$  and set  $B := B_r(b, \ell)$ ;
9:       for  $i = 1, \dots, n - \ell + 1$  do
10:        Set  $k := i + \ell$ ;
11:        Compute  $\bar{c}_{ikB} = p_B(n - i + 1) - \Delta u_{ik} - g_r(b, \ell)$ ;
12:        if  $\bar{c}_{ikB} < 0$  then
13:          Set  $H := H \cup \{(i, k, B)\}$ ;
14:        end if
15:      end for
16:      Set  $r := \min\{j : p_j < p_r\}$ ;
17:      If no such index exists, set  $done := \mathbf{true}$ ;
18:    end while
19:  end for
20:  return  $H$ ;
21: end function
```

4.3 Computational results

The proposed algorithms, discussed in the previous sections, have been tested on randomly generated instances. For generating all job data, the same approach as Uzsoy [69] and Rafiee Parsa et al. [61] has been used. Specifically, all the job processing times are drawn from a uniform distribution $p_j \in [1, 100]$, while job sizes s_j are drawn from four possible uniform distributions, labeled by $\sigma \in \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

$$\begin{array}{ll} \sigma_1 : s_j \in [1, 10] & \sigma_3 : s_j \in [3, 10] \\ \sigma_2 : s_j \in [2, 8] & \sigma_4 : s_j \in [1, 5]. \end{array}$$

Following the approach of both Uzsoy [69] and Rafiee Parsa et al. [61], the machine capacity is fixed at $b = 10$.

Since the pricing procedure of the P&B heuristic has a pseudopolynomial running time, instances with $b = 30$ and $b = 50$ have been also generated in order to assess how the procedure behaves with a larger capacity. Single-machine instances have been generated with $n \in \{20, 40, 60, 80, 100\}$, and with all the four σ size distributions. For each n, σ and b combinations, 10 random instances have been generated.

With the same job data, the corresponding instances of the parallel machines problem $Pm|p\text{-batch}, b, \sigma_j|\sum C_j$ have been solved for $m = 2, 3, 5$ identical machines. For testing the heuristics in the parallel-machines case and the B&P exact approach for the single-machine, only the $b = 10$ instances have been used.

All the tests ran in a Linux environment equipped with Intel Core i7-6500U CPU @ 2.50GHz processor; C++ language has been used for coding the algorithms, and CPLEX 12.8, called directly from C++ environment using CPLEX callable libraries, has been used to solve relaxed and mixed-integer programs.

Results for the P&B heuristic, in the single-machine and parallel-machines cases as well are discussed in Section 4.3.1 whereas Section 4.3.2 deals with the results of the exact B&P procedure in the single-machine case.

4.3.1 Evaluation of the heuristic algorithms

Both the CG based lower bound CG-LB and the objective value of the heuristic solution CG-UB have been evaluated. As far as the quality of the lower bound is concerned, the continuous relaxation of model (4.1)–(4.9) is not a significant competitor, zero being the typical value found by CPLEX at the root branching node. A more meaningful comparison can be performed against the combinatorial lower bound proposed by Uzsoy [69]. Such bound is based on a relaxation of $1|p\text{-batch}, b, \sigma_j|\sum C_j$ to a preemptive problem on b parallel machines; this lower bound is referred to as Parallel Relaxation (PR) in the following.

As far as the evaluation of CG-UB is concerned, it was difficult to compare the obtained results with the known literature as neither the test instances nor the computer codes used by Uzsoy [69] and Rafiee Parsa et al. [61] have been made available. Hence, some comparison have been made with the results of Rafiee Parsa et al. [61], using instances of *the same type*, but, for this reason, the comparison has to be taken with some care. On the other hand, when CPLEX is fed with model (4.1)–(4.9) and given some time, its internal heuristics do generate a number of heuristic solutions, although it has no chance of certifying optimality. Hence, CPLEX has been run on some set of instances in order to get heuristic solutions with a time limit of 300 seconds.

The times required to compute CG-LB and CG-UB are separately reported. The gap between CG-LB and CG-UB is evaluated as

$$\text{Gap (\%)} = \frac{\text{CG-UB} - \text{CG-LB}}{\text{CG-UB}} \cdot 100\%.$$

Single-machine

Tab. 4.1, Tab. 4.2 and Tab. 4.3 show the results over an increasing number of jobs with batch capacity $b = 10$, $b = 30$ and $b = 50$, respectively; the CG-UB was computed using CPLEX with a time limit of 60 seconds. Values are shown as average over each 10-instance group for the time, and as average, maximum (worst) and minimum (best) over each 10-instance group for the gap. Column “opt” reports the number of instances (out of 10) in which the solution can be certified to be the optimum, that is, in which CG-UB is equal to (the rounded up value of) CG-LB. The comparison between the CG-LB value and PR lower bound is also reported, computing the average, minimum (worst) and maximum (best) over each 10-instance group of the ratio $\frac{\text{CG-LB}}{\text{PR}}$.

Tab. 4.1: Results for CG-UB and CG-LB with $b = 10$.

Param		Times (s)		Gap (%)			$\frac{\text{CG-LB}}{\text{PR}}$			opt
n	σ	CG-LB	CG-UB	avg	worst	best	avg	min	max	
20	σ_1	0.01	0.04	1.30	3.20	0.00	1.25	1.19	1.31	2
	σ_2	0.01	0.03	1.55	3.63	0.00	1.22	1.20	1.27	2
	σ_3	0.01	0.03	0.63	3.30	0.00	1.19	1.15	1.21	7
	σ_4	0.01	0.09	2.15	4.63	0.82	1.29	1.23	1.37	0
40	σ_1	0.03	0.58	1.30	2.34	0.20	1.20	1.16	1.25	0
	σ_2	0.03	0.39	1.17	2.14	0.24	1.16	1.13	1.19	0
	σ_3	0.02	0.18	0.89	1.87	0.00	1.18	1.13	1.27	1
	σ_4	0.07	1.89	2.61	4.03	0.45	1.19	1.15	1.22	0
60	σ_1	0.14	8.34	0.91	2.03	0.23	1.17	1.12	1.21	0
	σ_2	0.12	1.62	0.98	1.90	0.34	1.13	1.10	1.15	0
	σ_3	0.05	0.44	0.49	1.09	0.00	1.16	1.13	1.20	1
	σ_4	0.39	30.41	2.57	3.97	0.94	1.14	1.12	1.16	0
80	σ_1	0.41	7.96	0.74	1.88	0.28	1.14	1.11	1.17	0
	σ_2	0.36	24.06	0.82	1.40	0.07	1.11	1.09	1.13	0
	σ_3	0.19	1.89	0.47	0.85	0.17	1.15	1.12	1.19	0
	σ_4	0.88	limit	5.78	10.77	2.20	1.11	1.10	1.12	0
100	σ_1	0.73	8.76	0.46	0.82	0.06	1.13	1.11	1.14	0
	σ_2	0.45	4.03	0.41	0.75	0.14	1.14	1.11	1.16	0
	σ_3	0.32	0.81	0.17	0.68	0.00	1.15	1.12	1.20	2
	σ_4	1.61	limit	4.44	7.66	1.34	1.09	1.08	1.10	0

In Tab. 4.1, it can be seen that, with $b = 10$, the computation of CG-LB is *very fast* with average CPU time less than 1 second in almost all the cases (with any number of jobs). The σ_4 instances are the most time demanding, with the only average computation time above 1 second. This is due to the fact that a larger set of columns is usually generated on such instances.

The computation of CG-UB is, as expected, the heaviest part of the procedure, with larger CPU time. However, only in the cases $n = 80, 100$ and $\sigma = \sigma_4$ CPLEX time limit is reached. Again, σ_4 instances were the most CPU time demanding, because

of the larger set of columns to be handled. The certified solution quality was very good, with an average optimality gap usually below 1.5%, and only one case ($n = 80$, $\sigma = \sigma_4$) above 5%.

Tab. 4.2: Results for CG-UB and CG-LB with $b = 30$.

Param		Times (s)		Gap (%)			CG-LB PR			
n	σ	CG-LB	CG-UB	avg	worst	best	avg	min	max	opt
20	σ_1	0.02	0.07	1.03	3.69	0.00	1.46	1.36	1.66	3
	σ_2	0.02	0.08	1.28	3.77	0.00	1.39	1.29	1.49	5
	σ_3	0.01	0.05	1.10	5.46	0.00	1.35	1.30	1.40	4
	σ_4	0.02	0.09	0.00	0.00	0.00	1.81	1.62	2.11	10
40	σ_1	0.21	3.11	3.13	6.43	0.21	1.30	1.23	1.39	0
	σ_2	0.18	1.72	3.83	5.62	2.33	1.27	1.21	1.33	0
	σ_3	0.10	2.95	4.37	6.29	2.98	1.20	1.17	1.26	0
	σ_4	0.71	2.19	1.18	5.13	0.07	1.51	1.41	1.60	0
60	σ_1	0.77	limit	6.78	10.27	3.51	1.22	1.18	1.27	0
	σ_2	0.72	37.44	5.21	8.46	1.88	1.20	1.17	1.22	0
	σ_3	0.42	30.35	3.74	5.79	1.44	1.15	1.13	1.18	0
	σ_4	2.38	10.59	2.28	4.24	0.05	1.36	1.31	1.41	0
80	σ_1	1.68	limit	11.05	17.40	3.40	1.21	1.16	1.23	0
	σ_2	1.41	limit	11.68	41.32	2.65	1.16	1.13	1.20	0
	σ_3	0.88	limit	7.94	12.24	3.53	1.12	1.11	1.13	0
	σ_4	4.75	limit	7.01	18.67	2.41	1.29	1.27	1.32	0
100	σ_1	3.27	limit	15.43	18.54	12.12	1.16	1.13	1.19	0
	σ_2	2.87	limit	9.23	11.46	3.08	1.13	1.12	1.14	0
	σ_3	1.79	limit	11.66	15.45	8.87	1.10	1.09	1.11	0
	σ_4	8.85	limit	6.38	9.54	3.32	1.26	1.23	1.29	0

From Tab. 4.2, it can be noticed that CPU time for CG-LB increases; this is expected, since a larger number of possible batches are generated with an increased capacity. The larger RMP obviously affect also the computation of CG-UB, which reaches the time limit in all the cases for $n = 80, 100$. The average optimality gaps worsen, but the largest increase is not found on σ_4 instances; instead, it affects more heavily σ_1 instances, especially for large n .

Overall, increasing capacity also increments the distance between the two lower bounds CG-LB and PR; the former performs better in every combination, ranging from an average 10% gain when $b = 30$, $n = 100$, and $\sigma = \sigma_3$ to an average 81% when $b = 30$, $n = 20$, and $\sigma = \sigma_4$. This is reasonable since PR is based on a preemptive relaxation to b parallel machines and allowing to split jobs on more machines weakens the relaxation.

Tab. 4.3 shows the results of the tests with capacity $b = 50$ that confirm the impact of b . The instances belonging to class σ_4 are still the most computationally demanding, both for lower bounding and heuristic solution. Instances with $\sigma = \sigma_1$ are the worse in terms of solution quality (with the exception of small 20 job instances) but,

Tab. 4.3: Results for CG-UB and CG-LB with $b = 50$.

Param		Times (s)		Gap (%)			$\frac{\text{CG-LB}}{\text{PR}}$			opt
n	σ	CG-LB	CG-UB	avg	worst	best	avg	min	max	
20	σ_1	0.02	0.09	0.00	0.00	0.00	1.70	1.48	1.94	10
	σ_2	0.02	0.09	0.10	0.38	0.00	1.65	1.53	1.77	7
	σ_3	0.02	0.08	0.28	2.66	0.00	1.52	1.35	1.62	7
	σ_4	0.02	0.10	0.00	0.00	0.00	2.13	1.90	2.39	10
40	σ_1	0.35	1.29	1.58	3.58	0.00	1.44	1.38	1.51	2
	σ_2	0.44	1.31	1.78	3.16	0.35	1.41	1.34	1.54	0
	σ_3	0.30	1.09	2.24	4.82	0.00	1.34	1.29	1.40	1
	σ_4	0.75	2.45	0.11	0.81	0.00	1.80	1.67	1.87	8
60	σ_1	1.39	23.59	4.47	6.64	1.24	1.38	1.30	1.44	0
	σ_2	1.45	8.38	2.54	5.54	0.08	1.32	1.29	1.37	0
	σ_3	0.76	13.37	4.30	6.65	1.81	1.23	1.20	1.27	0
	σ_4	6.72	12.56	1.25	2.73	0.00	1.52	1.43	1.64	3
80	σ_1	3.43	limit	7.40	10.26	3.17	1.32	1.27	1.38	0
	σ_2	3.71	limit	3.95	4.64	2.63	1.26	1.24	1.27	0
	σ_3	1.93	limit	4.94	10.32	2.00	1.19	1.17	1.20	0
	σ_4	18.34	limit	1.90	4.85	0.11	1.46	1.42	1.51	0
100	σ_1	7.09	limit	15.66	58.23	8.14	1.23	1.19	1.27	0
	σ_2	7.04	limit	7.19	10.33	4.27	1.21	1.19	1.23	0
	σ_3	3.50	limit	7.86	11.47	3.51	1.15	1.14	1.17	0
	σ_4	38.65	limit	3.26	5.70	0.84	1.39	1.35	1.47	0

curiously, the gap lowers on $n = 80$ instances when passing from $b = 30$ to $b = 50$. The worst average gap, 15.66%, is reached with $n = 100$ and $\sigma = \sigma_1$; also, compared to the previous case, PR worsens considerably with respect to CG-LB.

Rafiee Parsa et al. [61] provide a Hybrid Max-Min Ant System (HMMAS) that is, to the writer's knowledge, a state-of-the-art heuristic. A recent paper from the same authors has been published on the same problem (Rafiee Parsa et al. [62]) where a new Heuristic Neural Network approach, called HNN, is proposed. In this new paper, a comparison with HMMAS is presented and, as in the previous case, only capacity $b = 10$ is considered; a statistical analysis is also performed. This paper shows that results quality of the new procedure HNN is not better than HMMAS; on the contrary, apparently the average quality appears to be slightly worse.

Neither the source code nor the tested instances appear to be currently available; hence, an attempt to compare CG-UB with HMMAS has been made by testing CG-UB on generated instances of the same type and size as those used in Rafiee Parsa et al. [61]. Moreover, as in Rafiee Parsa et al. [61] only the capacity $b = 10$ was investigated, the performed comparison is limited to such value. Besides that, they do not even provide a new lower bound; in fact, they used the previously mentioned PR described in Uzsoy [69]. The reader being warned of the difficulty of such comparison, Tab. 4.4 points to the following situation: the results show that the

Tab. 4.4: Comparison between HMMAS (values from Rafiee Parsa et al. [61]) and CG-UB algorithms.

Param		Heuristic PR		$\frac{\text{CG-UB}}{\text{CG-LB}}$
n	σ	HMMAS	CG-UB	
20	σ_1	1.25	1.27	1.01
	σ_2	1.25	1.24	1.02
	σ_3	1.21	1.20	1.01
	σ_4	1.28	1.31	1.02
40	σ_1	1.19	1.21	1.01
	σ_2	1.19	1.18	1.01
	σ_3	1.18	1.19	1.01
	σ_4	1.20	1.22	1.03
60	σ_1	1.17	1.18	1.01
	σ_2	1.16	1.14	1.01
	σ_3	1.18	1.17	1.01
	σ_4	1.18	1.17	1.03
80	σ_1	1.16	1.15	1.01
	σ_2	1.16	1.12	1.01
	σ_3	1.16	1.15	1.01
	σ_4	1.16	1.18	1.06
100	σ_1	1.16	1.13	1.01
	σ_2	1.15	1.14	1.01
	σ_3	1.15	1.16	1.01
	σ_4	1.15	1.14	1.05

performance of CG-UB, evaluated against PR lower bound, seems to be very similar to that of HMMAS. Thus, it can be speculated that the two algorithms could give similar results for the upper bound, when they are run on the same instance set.

On the other hand, the availability of CG-LB allows to certify a narrower optimality gap for CG-UB. Considering the CG-LB lower bound, differences appear, as it can be noticed from the last column of the table. In fact, the use of CG-LB certifies that our found upper bounds CG-UB are very close to the integer optima for all the combinations (maximum gap is around 6%). This last consideration even more certifies that CG-LB is a better approximation of the integer optimum than the previously known PR.

It must be stressed, however, that, as the instances of Rafiee Parsa et al. [61] were not available, the optimality gap of their results against a strong lower bound is unknown. Thus, even if the results seem to suggest that the upper bounds are comparable, the algorithm quality cannot be directly benchmarked.

The quality of CG-UB has been compared to the quality of the heuristic solution reached by CPLEX, called CPLEX-UB, after 300 seconds of computation using model (4.1)–(4.9). The optimality gap of CPLEX-UB is most of the times well above 90% because the lower bound is zero or almost zero. Anyway, using the proposed

Tab. 4.5: Comparison between CPLEX-UB (300 seconds) and CG-UB.

n	Param	CPLEX-UB Gap (%)				CG-UB Gap (%)			
		avg	worst	best	#win	avg	worst	best	#win
20	σ_1	1.15	3.73	0.00	8	1.31	3.27	0.00	6
	σ_2	1.17	3.58	0.00	7	1.56	3.73	0.00	7
	σ_3	0.69	3.30	0.00	8	0.63	3.30	0.00	10
	σ_4	3.51	7.14	0.82	4	2.15	4.63	0.82	9
40	σ_1	9.60	14.63	5.97	0	1.30	2.34	0.20	10
	σ_2	9.34	16.02	5.43	0	1.17	2.14	0.24	10
	σ_3	4.41	8.13	2.34	0	0.89	1.87	0.00	10
	σ_4	12.03	18.16	9.09	0	2.61	4.03	0.45	10
60	σ_1	49.28	77.33	38.61	0	0.91	2.03	0.23	10
	σ_2	48.86	59.69	33.98	0	0.98	1.90	0.34	10
	σ_3	33.78	41.74	22.84	0	0.49	1.09	0.00	10
	σ_4	45.86	66.77	24.59	0	2.57	3.97	0.94	10
80	σ_1	73.77	88.55	59.86	0	0.74	1.88	0.28	10
	σ_2	66.12	77.45	53.45	0	0.82	1.40	0.07	10
	σ_3	52.79	73.82	38.40	0	0.47	0.85	0.17	10
	σ_4	84.09	102.96	62.95	0	5.78	10.77	2.20	10

stronger lower bound, a more realistic optimality gap can be computed for CPLEX-UB as

$$\frac{\text{CPLEX-UB} - \text{CG-LB}}{\text{UB}^*} \cdot 100\% \quad \text{with } \text{UB}^* = \min\{\text{CPLEX-UB}, \text{CG-UB}\}.$$

The gap for CG-UB is recomputed as $\frac{\text{CG-UB} - \text{CG-LB}}{\text{UB}^*} \cdot 100\%$ for uniformity.

The comparison is reported in Tab. 4.5, in terms of average, worst and best gap, as done in previous comparison. Column “#win” counts the number of instances (out of 10) for which each algorithm achieves the best solution; in the case of a draw, a “#win” is counted for both, so the two columns can sum to more than 10. Instances with $n = 20, 40, 60, 80$ and $b = 10$ have been tested. CPLEX ran for the full 300 seconds on all the instances, without proving optimality for any of them; CG-UB on the other hand ran with the same 60 seconds time limit as in Tab. 4.1. Basically, except for the small $n = 20$ instances, CPLEX solution is consistently worse than CG-UB.

Parallel-machines

With the same data, the $Pm|p\text{-batch}, b, \sigma_j | \sum C_j$ problem has been solved with $m = 2, 3, 5$ identical machines. The tests have been limited to the case $b = 10$, and time limit for the branching phase of the heuristic was raised to 180 seconds.

The results are reported in Tab. 4.6, Tab. 4.7 and Tab. 4.8. Apparently, increasing the number of machines has a very mild impact on the CPU time needed for computing the lower bound. The growth of the computational cost is much higher for the branching phase, but with a certain variability on the four classes of instances, with

classes σ_1 and σ_4 exhibiting the largest growth. Again, class σ_4 broke the time limit in all the instances. The quality of the solution, as measured by the percentage gap, does not suffer seriously, except for the case $n = 100$, $m = 5$, $\sigma = \sigma_4$. The worst average gap of 14.09% is caused by one single instance with a very large gap of 81.62%; if a larger but still acceptable time limit of 300 seconds is allowed, the average gap for this class decreases to 4.63% with a max gap of 12.56%.

The lower bound by Uzsoy [69] is easily extended to the parallel machines case allowing a relaxation to mb parallel machines. Tab. 4.6, Tab. 4.7 and Tab. 4.8 also compare CG-LB with PR extended to the parallel machine case. The ratio between the two bounds is apparently unaffected by the growth of m .

Tab. 4.6: Results for CG-UB and CG-LB with $b = 10$ and 2 parallel machines.

Param		Times (s)		Gap (%)			CG-LB PR			opt
n	σ	CG-LB	CG-UB	avg	worst	best	avg	min	max	
20	σ_1	0.03	0.12	0.78	2.00	0.00	1.24	1.18	1.29	2
	σ_2	0.03	0.09	1.43	4.78	0.00	1.21	1.19	1.25	2
	σ_3	0.02	0.07	0.57	3.04	0.00	1.18	1.14	1.19	7
	σ_4	0.04	0.22	2.08	4.11	0.00	1.28	1.22	1.35	1
40	σ_1	0.06	0.82	1.14	2.28	0.01	1.19	1.16	1.24	0
	σ_2	0.05	0.90	1.08	1.73	0.23	1.16	1.13	1.18	0
	σ_3	0.03	0.22	0.75	1.39	0.00	1.18	1.13	1.26	1
	σ_4	0.13	5.01	2.07	3.73	0.37	1.18	1.15	1.22	0
60	σ_1	0.25	10.22	0.83	1.64	0.23	1.16	1.12	1.20	0
	σ_2	0.21	1.72	0.90	1.56	0.32	1.12	1.10	1.15	0
	σ_3	0.13	0.60	0.38	1.00	0.01	1.16	1.13	1.20	0
	σ_4	0.49	76.71	2.35	3.72	1.69	1.14	1.12	1.16	0
80	σ_1	0.62	45.96	0.76	1.94	0.21	1.14	1.10	1.17	0
	σ_2	0.54	40.97	0.71	1.00	0.08	1.11	1.09	1.13	0
	σ_3	0.35	3.29	0.46	0.76	0.18	1.14	1.12	1.18	0
	σ_4	1.30	limit	5.65	10.31	1.33	1.11	1.10	1.12	0
100	σ_1	1.08	19.16	0.44	0.77	0.07	1.13	1.11	1.14	0
	σ_2	0.79	3.66	0.41	0.73	0.12	1.13	1.11	1.16	0
	σ_3	0.59	1.63	0.21	0.66	0.00	1.15	1.11	1.20	2
	σ_4	2.25	limit	7.19	26.27	1.08	1.09	1.08	1.10	0

4.3.2 Evaluation of the exact approach

The B&P exact algorithm has been tested on the single machine $b = 10$ generated instances.

The reference algorithm for the exact approaches on $1|p\text{-batch}, b, \sigma_j| \sum C_j$ is the Branch-and-Bound (B&B) of Azizoglu and Webster [5], which was developed for the weighted version $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ but the authors also reported results for the unweighted version. In the latter case, the lower bound by Azizoglu and Webster [5] reduces to the one by Uzsoy [69]. The B&P procedure presented in Section 4.1.4

Tab. 4.7: Results for CG-UB and CG-LB with $b = 10$ and 3 parallel machines.

Param		Times (s)		Gap (%)			CG-LB PR			opt
n	σ	CG-LB	CG-UB	avg	worst	best	avg	min	max	
20	σ_1	0.03	0.13	0.52	1.29	0.00	1.24	1.18	1.29	3
	σ_2	0.03	0.10	1.15	2.90	0.00	1.21	1.19	1.25	2
	σ_3	0.03	0.08	0.45	2.49	0.00	1.18	1.14	1.19	7
	σ_4	0.05	0.30	1.77	3.61	0.00	1.30	1.23	1.36	1
40	σ_1	0.12	1.68	1.11	2.02	0.00	1.19	1.16	1.24	1
	σ_2	0.09	1.00	0.88	1.38	0.00	1.16	1.13	1.18	1
	σ_3	0.06	0.43	0.80	1.64	0.00	1.17	1.12	1.25	1
	σ_4	0.19	4.81	1.76	3.37	0.19	1.19	1.16	1.22	0
60	σ_1	0.35	3.89	0.73	1.54	0.24	1.16	1.12	1.20	0
	σ_2	0.32	3.47	0.86	1.50	0.29	1.12	1.10	1.15	0
	σ_3	0.24	1.09	0.38	1.12	0.00	1.15	1.13	1.19	1
	σ_4	0.62	76.82	2.00	3.92	1.01	1.15	1.12	1.16	0
80	σ_1	1.04	16.93	0.62	1.57	0.18	1.14	1.10	1.16	0
	σ_2	0.75	47.57	0.71	1.12	0.08	1.11	1.09	1.12	0
	σ_3	0.58	3.99	0.47	0.72	0.19	1.14	1.12	1.18	0
	σ_4	1.49	limit	5.23	13.93	1.20	1.12	1.10	1.13	0
100	σ_1	1.99	20.33	0.44	0.73	0.07	1.12	1.11	1.14	0
	σ_2	1.12	4.92	0.39	0.69	0.12	1.13	1.10	1.16	0
	σ_3	1.18	2.83	0.17	0.65	0.00	1.15	1.11	1.20	2
	σ_4	3.09	limit	5.44	19.24	1.58	1.09	1.08	1.10	0

is based on the same branching scheme as the one in Azizoglu and Webster [5], with the same dominance conditions.

The results are reported in Tab. 4.9 for instances with up to 40 jobs. Cumulative results for all the 40 instances have been reported by number of jobs, across job size distributions. For times, nodes, and gap, the average value is reported; for the number of found optima, the total sum over the 40 instances is reported.

The comparison considered here is between these two approaches.

- The B&P equipped with the proposed lower bound CG-LB obtained by CG and the CG-UB heuristic for the root node upper bound.
- The B&B based on the same branching scheme, with the same dominance conditions, equipped with:
 - an evaluation of two different lower bounds of increasing complexity as described in Section 3 of Azizoglu and Webster [5], using the best one for the actual lower bound;
 - the procedure described in Section 2.4 of Azizoglu and Webster [5] for the root node upper bound and evaluated at every non-leaf node to further improve the quality of the upper bound.

Tab. 4.8: Results for CG-UB and CG-LB with $b = 10$ and 5 parallel machines.

Param	σ	Times (s)		Gap (%)			CG-LB PR			opt
		CG-LB	CG-UB	avg	worst	best	avg	min	max	
20	σ_1	0.05	0.20	0.47	1.95	0.00	1.27	1.18	1.30	2
	σ_2	0.05	0.16	0.84	1.77	0.00	1.22	1.20	1.24	3
	σ_3	0.05	0.10	0.30	1.57	0.00	1.19	1.15	1.20	7
	σ_4	0.05	0.26	0.83	2.20	0.00	1.36	1.27	1.44	2
40	σ_1	0.23	1.94	0.91	1.59	0.24	1.19	1.17	1.23	0
	σ_2	0.18	1.62	0.71	1.25	0.00	1.16	1.14	1.19	1
	σ_3	0.15	0.68	0.72	1.79	0.02	1.17	1.13	1.24	0
	σ_4	0.27	6.02	1.28	2.43	0.32	1.21	1.17	1.26	0
60	σ_1	0.53	5.34	0.70	1.25	0.25	1.16	1.13	1.20	0
	σ_2	0.55	7.23	0.81	1.78	0.34	1.12	1.10	1.15	0
	σ_3	0.45	2.17	0.36	0.89	0.00	1.15	1.13	1.19	1
	σ_4	0.69	89.00	1.65	3.14	0.37	1.16	1.13	1.18	0
80	σ_1	1.33	17.41	0.61	1.56	0.11	1.14	1.11	1.16	0
	σ_2	0.93	38.02	0.62	0.99	0.01	1.11	1.09	1.12	0
	σ_3	0.78	4.37	0.39	0.60	0.16	1.14	1.12	1.17	0
	σ_4	1.53	limit	2.50	4.57	1.27	1.13	1.11	1.14	0
100	σ_1	2.76	20.14	0.43	0.79	0.05	1.12	1.11	1.14	0
	σ_2	1.41	6.34	0.38	0.61	0.12	1.13	1.10	1.15	0
	σ_3	1.54	3.50	0.16	0.63	0.00	1.14	1.11	1.19	2
	σ_4	3.22	limit	14.09	81.62	0.87	1.10	1.09	1.11	0

- The node exploration policy in both algorithms was kept depth-first.

Both exact methods were run with a 900 seconds time limit. The table compares the CPU time and the number of open nodes; column “opt” counts the number of certified optima obtained within the time limit. On the left part of the table, the results for the B&P equipped with CG-LB and CG-UB are reported; on the right side, there are the results for the B&B equipped with lower bound and heuristic by Azizoglu and Webster [5].

By allowing the B&P to run without time limit, all the optimal values of the 40-jobs instances were also collected, even if a few instances required several hours of computation; hence, the “Gap” columns report the percentage gap $\frac{UB-OPT}{OPT} \cdot 100\%$ between the best upper bound UB obtained within the time limit and the exact optimum OPT for both algorithms.

The B&B has an impressively fast node processing, since the lower bound by Uzsoy [69] has a very cheap running time. As the number of jobs increases, however, the more accurate lower bound obtained by CG allows the branch and price to outperform the competitor. The latter algorithm is able to optimally solve all the instances with up to 30 jobs and more than a half of the 40-jobs instances, with optimality gaps mostly below 1%.

Tab. 4.9: Comparison of exact approaches.

Param	Branch-and-Price (CG-LB and CG-UB)						Branch-and-Bound (depth-first)										
	Times (s)			Nodes			Gap (%)			Times (s)			Nodes			Gap (%)	
n	σ	avg	max	avg	max	opt	avg	max	opt	avg	max	avg	max	opt	avg	max	opt
10	σ_1	0.06	0.48	4.2	32	10	0.00	0.00	10	0.01	0.01	1266.3	2222	10	0.00	0.00	10
	σ_2	0.02	0.08	15.6	147	10	0.00	0.00	10	0.01	0.01	953.0	1910	10	0.00	0.00	10
	σ_3	0.01	0.05	5.4	54	10	0.00	0.00	10	0.01	0.01	514.9	1244	10	0.00	0.00	10
	σ_4	0.03	0.05	21.5	56	10	0.00	0.00	10	0.01	0.01	1895.6	3119	10	0.00	0.00	10
10		0.03	0.48	12.7	147	40	0.00	0.00	40	0.01	0.01	1157.4	3119	40	0.00	0.00	40
20	σ_1	1.64	4.27	824.0	2658	10	0.00	0.00	10	35.55	67.14	9556527.9	18522644	10	0.00	0.00	10
	σ_2	2.84	19.99	1639.6	11820	10	0.00	0.00	10	14.07	34.69	3553628.2	9094599	10	0.00	0.00	10
	σ_3	4.38	38.05	3508.0	31389	10	0.00	0.00	10	5.95	16.56	1276560.2	3745567	10	0.00	0.00	10
	σ_4	3.59	13.30	985.1	3351	10	0.00	0.00	10	30.64	56.14	7439538.0	14484092	10	0.00	0.00	10
20		3.11	38.05	1739.2	31389	40	0.00	0.00	40	21.55	67.14	5456563.6	18522644	40	0.00	0.00	40
30	σ_1	3.83	19.05	715.9	4212	10	0.00	0.00	10	900.00	900.00	133616540.3	167662461	0	1.45	4.91	0
	σ_2	82.65	655.98	50281.3	467906	10	0.00	0.00	10	900.00	900.00	133304717.2	144452851	0	1.27	3.58	0
	σ_3	16.06	136.41	7353.7	67290	10	0.00	0.00	10	900.00	900.00	129637409.5	145358435	0	0.14	1.05	0
	σ_4	90.90	307.28	11255.6	35881	10	0.00	0.00	10	900.00	900.00	126362353.9	139369516	0	2.16	5.35	0
30		48.36	655.98	17401.6	467906	40	0.00	0.00	40	900.00	900.00	130730255.2	167662461	0	1.26	5.35	0
40	σ_1	619.85	901.11	40824.5	71268	6	0.13	0.63	6	900.00	900.00	100200931.0	121947526	0	5.83	9.06	0
	σ_2	547.41	900.93	52105.6	151857	6	0.18	0.99	6	900.00	900.00	82962783.4	89454946	0	4.25	8.09	0
	σ_3	499.44	900.19	80444.0	165196	7	0.08	0.51	7	900.00	900.00	88661788.3	96750260	0	2.18	4.75	0
	σ_4	841.06	905.80	25943.0	53933	5	0.93	3.11	5	900.00	900.00	83033865.0	96502979	0	5.04	8.44	0
40		626.94	905.80	49829.3	165196	22	0.33	3.11	22	900.00	900.00	88714841.9	121947526	0	4.33	9.06	0

4.4 Final remarks

In this work, CG techniques for solving problem $1|p\text{-batch}, b, \sigma_j| \sum C_j$ have been explored, generalizing such techniques to problems with parallel machines. The exponential size model (4.15)–(4.18), handled by means of CG, allows to find (to the writer’s knowledge) the tightest known lower bound for $1|p\text{-batch}, b, \sigma_j| \sum C_j$. Embedded in a simple P&B approach, it achieves high-quality solutions for instances up to 100 jobs in size, with certified optimality gaps.

Thus it can be claimed that model (4.15)–(4.18) is *strong*: its relaxation gives a sharp bound, and the generated columns can be effectively composed into high-quality feasible solutions. The comparison with state-of-the-art (meta)heuristics like HMMAS is admittedly problematic because of lack of available code and instances, but the P&B heuristic is, in the writer’s view, at least as accurate as the state-of-the-art heuristics, faster and *simpler*, since it mostly relies on a MIP solver, with the addition of some ad-hoc code.

Tab. 4.10: Comparison between CG-UB and real optima with $b = 10$.

Param		Gap (%)		
n	σ	avg	worst	best
10	σ_1	0.00	0.00	0.00
	σ_2	0.39	3.89	0.00
	σ_3	0.00	0.00	0.00
	σ_4	0.22	1.13	0.00
10		0.15	3.89	0.00
20	σ_1	0.26	0.63	0.00
	σ_2	0.44	1.54	0.00
	σ_3	0.01	0.14	0.00
	σ_4	0.74	3.12	0.00
20		0.36	3.12	0.00
30	σ_1	0.08	0.73	0.00
	σ_2	0.16	0.68	0.00
	σ_3	0.01	0.03	0.00
	σ_4	0.78	2.11	0.00
30		0.26	2.11	0.00
40	σ_1	0.30	0.92	0.01
	σ_2	0.33	0.99	0.00
	σ_3	0.11	0.51	0.00
	σ_4	1.32	3.11	0.25
40		0.51	3.11	0.00

Embedded in an exact algorithm, the new lower bound allows to extend the size of solvable $1|p\text{-batch}, b, \sigma_j| \sum C_j$ instances towards 40 jobs. Having available all the optimal solutions for the single machine problems up to 40-jobs instances, the actual relative error of the CG-UB heuristic on such instances can be seen to be even lower

than the figures estimated in the experiments of Section 4.3.1; see Tab. 4.10, where a comparison between CG-UB and the real optima is performed, evaluating the gap as $\frac{UB-OPT}{OPT} \cdot 100\%$.

Although all the random distributions for job data mentioned in literature have been used in the tests, the interested reader might be worried about the relatively narrow distributions for job sizes given in classes $\sigma_1, \dots, \sigma_4$. Hence some tests have been performed with capacity $b = 50$ and a new class σ_5 , with a wider distribution of $s_j \in [1, 50]$. The results of CG-UB on such instances, for the single and parallel machines cases as well, are summarized in Tab. 4.11, showing that CG-UB still handles such instances within practical time limits, guaranteeing narrow optimality gaps.

Tab. 4.11: Results for CG-UB and CG-LB with $b = 50$ and $\sigma = \sigma_5$ considering $m = \{1, 2, 3, 5\}$ parallel machines. Time limit for CG-UB set to 60 seconds for the single machine case, and to 180 seconds for the multiple machines cases.

Param		Times (s)		Gap (%)			$\frac{CG-LB}{PR}$			
n	m	CG-LB	CG-UB	avg	worst	best	avg	min	max	opt
20	1	0.01	0.03	0.88	3.19	0.00	1.27	1.16	1.39	2
	2	0.01	0.04	0.66	2.95	0.00	1.26	1.15	1.36	4
	3	0.01	0.11	0.70	3.20	0.00	1.26	1.16	1.34	4
	5	0.02	0.12	0.50	1.51	0.00	1.28	1.18	1.36	4
40	1	0.04	0.52	0.90	1.51	0.00	1.23	1.19	1.29	1
	2	0.05	0.56	0.74	1.50	0.00	1.22	1.19	1.27	1
	3	0.11	0.97	0.64	1.39	0.00	1.22	1.18	1.26	1
	5	0.21	1.32	0.55	1.26	0.00	1.22	1.18	1.25	2
60	1	0.16	3.30	0.77	1.35	0.10	1.19	1.16	1.22	0
	2	0.26	1.58	0.51	0.90	0.12	1.18	1.16	1.21	0
	3	0.43	3.75	0.54	1.24	0.08	1.18	1.16	1.21	0
	5	0.83	4.79	0.46	0.79	0.07	1.18	1.15	1.21	0
80	1	0.57	19.06	0.61	1.15	0.18	1.17	1.15	1.20	0
	2	0.79	17.68	0.53	1.10	0.19	1.17	1.15	1.20	0
	3	1.15	33.10	0.50	0.97	0.15	1.17	1.15	1.19	0
	5	1.53	23.49	0.52	0.98	0.19	1.17	1.15	1.19	0
100	1	1.20	22.58	0.54	0.86	0.34	1.16	1.12	1.19	0
	2	1.64	48.68	0.50	0.81	0.30	1.16	1.12	1.18	0
	3	2.92	41.17	0.45	0.79	0.26	1.15	1.12	1.18	0
	5	3.54	64.40	0.47	0.82	0.25	1.15	1.12	1.18	0

The new model relies on Property 1 in order to express the linear objective function by means of “positional” coefficients. Property 2 is crucial in order to develop an efficient pricing procedure. The approach is flexible enough to be extended to problems with parallel machines with a very limited effort, and while it is not simple to address weighted $\sum w_j C_j$ objectives, the next chapter shows in detail how this successful approach can be extended to the $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ problem.

Weighted total completion time: single machine and heuristics



WE now consider the single-machine scheduling problem of minimizing the total weighted completion time (also called *total weighted flow time*) in a parallel-batching environment. Recalling the notation introduced in Chapter 3, a set of jobs $N = \{1, 2, \dots, n\}$ is to be partitioned into batches and processed on a single machine. Each job $j \in N$ has a given processing time p_j , a weight w_j and a size s_j ; jobs are partitioned and processed without interruptions in a batch sequence $S = (B_1, B_2, \dots, B_t)$, and each batch B_k in S must satisfy $\sum_{j \in B_k} s_j \leq b$, where b is the machine capacity. The jobs in a same batch B_k are processed simultaneously, with the longest job determining the processing time for the whole batch, having that $p_B = \max\{p_j : j \in B\}$. All the jobs in the same batch B_k share the same completion time, that is $C_j = C_{B_k} = \sum_{l=1}^k p_{B_l}$. The considered problem calls for finding S that minimizes $f(S) = \sum_{j \in N} w_j C_j$.

This problem can be denoted as $1|p\text{-batch}, b, \sigma_j | \sum w_j C_j$ in the classical three-fields notation by Graham et al. [31].

Recently, new interest has grown about batching problems, since additive manufacturing often requires a batch production to optimize the chamber space (Zhang et al. [71]). However, processing times depend on different factors than those of conventional production and there are typically more size constraints to be addressed.

The contribution of this chapter is twofold.

- Working along the lines of Alfieri et al. [2] and Alfieri et al. [3], a strong lower bound for the $1|p\text{-batch}, b, \sigma_j | \sum w_j C_j$ problem is developed. The lower bound is based on the continuous relaxation of a very large integer Linear Program (LP) solved by means of Column Generation (CG) techniques. The LP does not rely on time-indexing, instead it uses a graph-based formulation based on peculiarities of the $\sum w_j C_j$ objective. The lower bound given by this formulation is stronger than any other currently available bound.
- The lower bound delivered by CG is sharp but still too computationally heavy for supporting an exact Branch-and-Price (B&P) procedure. Anyway, simple but effective rounding heuristics applied to the relaxed problem allow to quickly derive feasible solutions whose cost is within a few percentage points from the optimum.

5.1 Column Generation models

5.1.1 The graph-based model

For the $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ problem, we consider two distinct (but equivalent) CG models; both are based on a very large (multi)graph specified as follows. Recalling the notation introduced in Chapter 3, and following the definition (4.11) given in Section 4.1, let $\mathcal{B} = \{B \subseteq N : \sum_{j \in B} s_j \leq b\}$ be the set of all the possible batches.

The graph $G(V, A)$ is made of a set of vertices and arcs as follows:

$$V = \{1, 2, \dots, W + 1\}, \quad W = \sum_{j=1}^n w_j,$$

$$A = \left\{ (i, k, B) : 1 \leq i < k \leq W + 1; B \in \mathcal{B}; \sum_{j \in B} w_j = k - i \right\}$$

Every arc $(i, k, B) \in A$ connects two nodes i, k and is associated to a possible batch B . Each arc is given a cost $c_{ikB} = (W - i + 1)p_B$, with $p_B = \max\{p_j : j \in B\}$. We call a path P from node 1 to node $W + 1$ a *partition path* if the sets $\{B : (i, k, B) \in P\}$ form a partition of N . Feasible batch sequences are mapped onto partition paths and vice versa, as established by the following Property 4, written following the lines of Property 1.

Property 4. $S = (B_1, B_2, \dots, B_t)$ is a feasible batch sequence iff

$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \dots, (i_t, k_t, B_t)] \quad (i_1 = 1, k_t = W + 1)$$

is a partition path $1 \rightarrow W + 1$ in $G(V, A)$, and $f(S) = \sum \{c_{ikB} : (i, k, B) \in P\}$.

Proof. The one-to-one mapping of batch sequences to paths is easily established. Given a batch sequence $S = (B_1, B_2, \dots, B_t)$, the path

$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \dots, (i_t, k_t, B_t)] \quad (5.1)$$

can be built from arcs of G , choosing them from the arc set as follows:

$$\begin{aligned} i_1 &= 1, & k_1 &= i_1 + w_{B_1} \\ i_\ell &= k_{\ell-1}, & k_\ell &= i_\ell + w_{B_\ell} \quad (\ell = 2, \dots, t). \end{aligned} \quad (5.2)$$

Note that $k_t - i_1 = \sum_{\ell=1}^t w_{B_\ell} = W$, hence $k_t = W + 1$, and P is a path $1 \rightarrow W + 1$ in G ; the job set N is guaranteed to be partitioned over the arcs of P because it is partitioned in the batches of S by hypothesis.

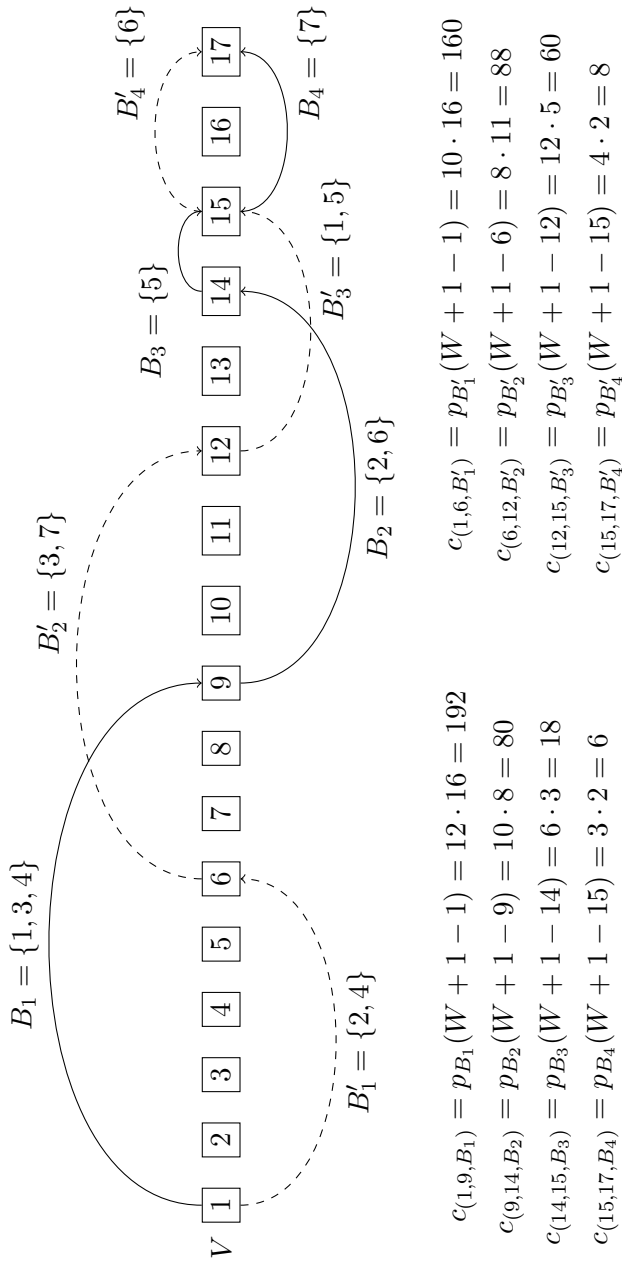


Fig. 5.1: Example of partition paths. Batch sequence $S = (B_1, B_2, B_3, B_4)$ corresponds to partition path $P = [(1, 9, B_1), (9, 14, B_2), (14, 15, B_3), (15, 17, B_4)]$ (continuous lines). Batch sequence $S' = (B'_1, B'_2, B'_3, B'_4)$ corresponds to partition path $P' = [(1, 6, B'_1), (6, 12, B'_2), (12, 15, B'_3), (15, 17, B'_4)]$ (dashed lines).

Example

As an example for Property 4, consider a 7-jobs instance with machine capacity $b = 10$, job set $N = \{1, 2, \dots, 7\}$ and:

$$\begin{aligned} p_1, \dots, p_7 &= 12, 10, 8, 8, 6, 4, 3; \\ s_1, \dots, s_7 &= 3, 3, 3, 4, 4, 7, 7; \\ w_1, \dots, w_7 &= 2, 3, 4, 2, 1, 2, 2; \quad (W = 16). \end{aligned}$$

The graph $G(V, A)$ is made on node set $V = \{1, 2, \dots, 17\}$; referring to Fig. 5.1, it is not practical to show all the arcs even for this small instance. For example between node 1 and 5 all the arcs $(1, 5, \{3\})$, $(1, 5, \{1, 4\})$, $(1, 5, \{1, 6\})$, $(1, 5, \{2, 5\})$, etc., hence all batches B with $\sum_{j \in B} w_j = 5 - 1 = 4$, should appear; the same arcs should appear between node 2 and node 6, and so on.

In Fig. 5.1 two examples of partition paths, P and P' , are sketched. Path P corresponds to the batch sequence $S = (B_1, B_2, B_3, B_4)$; note how B_1, B_2, B_3, B_4 form a partition of the job set N . Obviously $C_{B_1} = p_{B_1} = p_1 = 12$, $C_{B_2} = p_{B_1} + p_{B_2} = p_1 + p_2 = 22$, $C_{B_3} = p_{B_1} + p_{B_2} + p_{B_3} = p_1 + p_2 + p_5 = 28$, $C_{B_4} = p_{B_1} + p_{B_2} + p_{B_3} + p_{B_4} = p_1 + p_2 + p_5 + p_7 = 31$, and $w_{B_1} = w_1 + w_3 + w_4 = 8$, $w_{B_2} = w_2 + w_6 = 5$, $w_{B_3} = w_5 = 1$, $w_{B_4} = w_7 = 2$. The objective function computed for S is

$$\begin{aligned} f(S) &= w_{B_1}C_{B_1} + w_{B_2}C_{B_2} + w_{B_3}C_{B_3} + w_{B_4}C_{B_4} \\ &= 8 \cdot 12 + 5 \cdot 22 + 1 \cdot 28 + 2 \cdot 31 = 296, \end{aligned}$$

but also

$$\begin{aligned} f(S) &= p_{B_1}w_{B_1} + \\ & p_{B_1}w_{B_2} + p_{B_2}w_{B_2} + \\ & p_{B_1}w_{B_3} + p_{B_2}w_{B_3} + p_{B_3}w_{B_3} + \\ & p_{B_1}w_{B_4} + p_{B_2}w_{B_4} + p_{B_3}w_{B_4} + p_{B_4}w_{B_4} = \\ &= p_{B_1} \cdot 16 + p_{B_2} \cdot 8 + p_{B_3} \cdot 3 + p_{B_4} \cdot 2 = \\ &= c_{(1,9,B_1)} + c_{(9,14,B_2)} + c_{(14,15,B_3)} + c_{(15,17,B_4)} = \\ f(S) &= c(P) = 192 + 80 + 18 + 6 = 296 \end{aligned}$$

Similarly, the reader can verify that for S' and P' , $f(S') = w_{B'_1}C_{B'_1} + w_{B'_2}C_{B'_2} + w_{B'_3}C_{B'_3} + w_{B'_4}C_{B'_4} = 316$ and $c(P') = 316$.

Column Generation

Determining an optimal batch sequence amounts to computing a minimum cost $1 \rightarrow W + 1$ partition path on G ; the huge number of arcs involved in such problem

can be handled by means of CG techniques. In general, CG techniques apply the simplex method to very large LP models like

$$\min \left\{ \sum_{i \in \mathcal{S}} c_i x_i : \mathbf{a}_i x_i = \mathbf{b}, x_i \geq 0, i \in \mathcal{S} \right\} \quad (\mathbf{a}_i \in \mathbb{R}^m) \quad (5.3)$$

where the number of variables x_i and columns \mathbf{a}_i , for $i \in \mathcal{S}$, is so large that the whole program cannot be kept in memory. In the main iteration of a CG procedure, the *master problem* in (5.3) is solved on a restricted set $\mathcal{S}' \subset \mathcal{S}$. The solution of such Restricted Master Problem (RMP) is then proved (or disproved) optimal for (5.3) by identifying columns having minimum reduced cost; this is done using the simplex multipliers/dual variables $\boldsymbol{\eta} \in \mathbb{R}^m$ associated with the optimal basis of the RMP, computing

$$r^* = \min\{r_i = c_i - \boldsymbol{\eta}^\top \mathbf{a}_i : i \in \mathcal{S}\}. \quad (5.4)$$

If $r^* < 0$, one or more columns with reduced cost $r_i < 0$ are found and they are added to the restricted set \mathcal{S}' for a new iteration; otherwise, optimality of the current solution is proved. The whole CG procedure can usually be made computationally efficient if the *pricing problem* (5.4) presents a suitable combinatorial structure. A broad presentation of CG can be found, for example, in Desrosiers and Lübbecke [17].

Two “natural” procedures arise for solving the (continuous relaxation of the) problem of computing a minimum-cost partition path by means of CG techniques, leading to different master programs and pricing problems. The first one relies on a master problem which combines characteristics of an arc-based minimum cost flow problem formulation with additional set-partitioning constraints. The second one relies on a path-based minimum cost flow formulation with additional set-partitioning constraints. In both cases, a crucial step for the pricing procedure consists in solving a cardinality-constrained knapsack problem that is handled by Dynamic Programming (DP).

5.1.2 An arc-based flow model

Master problem

The problem of finding a minimum-cost $1 \rightarrow W + 1$ partition path can be represented by a very large binary LP. For a batch B , we denote by $\mathbf{a}_B \in \{0, 1\}^n$ the incidence vector of set B , where each of the n components $a_{B,j} = 1$ iff $j \in B$; also, let us introduce a vector constant $\mathbf{1} = (1, 1, \dots, 1)^\top$. As control variables we define binaries $x_{ikB} = 1$ iff arc $(i, k, B) \in A$ is used in the minimum cost $1 \rightarrow W + 1$ path. The LP for finding the best $1 \rightarrow W + 1$ partition path, very similar to the one (4.15)–(4.18) previously described for the unweighted case, writes out as follows.

$$\text{minimize } \sum_{(i,k,B) \in A} c_{ikB} x_{ikB} \quad (5.5)$$

$$\text{subject to } \sum_{\substack{(k,B): \\ (i,k,B) \in A}} x_{ikB} - \sum_{\substack{(k,B): \\ (k,i,B) \in A}} x_{kiB} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, W \\ -1 & i = W + 1 \end{cases} \quad (5.6)$$

$$\sum_{(i,k,B) \in A} \mathbf{a}_B x_{ikB} = \mathbf{1} \quad (5.7)$$

$$x_{ikB} \in \{0, 1\} \quad (i, k, B) \in A \quad (5.8)$$

The objective function (5.5) and constraints (5.6) require a unit of integer flow to be pushed along a $1 \rightarrow W + 1$ path at minimum total cost: constraints (5.6) are flow conservation constraints written for nodes $i = 1, \dots, W + 1$, with source 1 and sink $W + 1$. This is a well known (Ahuja et al. [1]) LP formulation of a shortest path problem.

The vector constraint (5.7) subsumes n scalar constraints requiring that the path is actually a partition path, that is, the job set $N = \{1, 2, \dots, n\}$ is exactly partitioned over the arcs (i, k, B) with $x_{ikB} = 1$. A lower bound for this integer program is immediately obtained by replacing (5.8) with

$$x_{ikB} \geq 0 \quad (i, k, B) \in A. \quad (5.9)$$

The dual of (5.5)–(5.9) is the following, with dual variables u_1, \dots, u_{W+1} and v_1, \dots, v_n .

$$\text{maximize } u_1 - u_{W+1} + \sum_{j=1}^n v_j \quad (5.10)$$

$$\text{subject to } u_i - u_k + \sum_{j \in B} v_j \leq c_{ikB} \quad (i, k, B) \in A \quad (5.11)$$

The number of constraints in program (5.5)–(5.9) is $n + W + 1$, which can be quite large but is still manageable on most instances, up to a reasonable number of jobs; also, the columns of constraints (5.6) and constraints (5.7) are rather sparse. On the other hand, the number of variables x_{ikB} is too high, and requires a CG approach. We consider a subset $A' \subset A$ of arcs/variables and formulate a RMP like (5.5)–(5.9) over the arcs in A' only.

Pricing

Following the lines of what we did in Section 4.1.2, now we illustrate how new arcs can be added.

The dual of the RMP is (5.10)–(5.11) with constraints restricted to the set A' . Once the RMP is solved, the optimal dual variables for the restricted dual are also available: u_1, \dots, u_{W+1} from the simplex multipliers of constraints (5.6) and v_1, \dots, v_n from the simplex multipliers of constraints (5.7). Such values are used to formulate a pricing problem.

For a fixed pair of node indices $1 \leq i < k \leq W + 1$, an arc with minimum reduced cost \bar{c}_{ikB^*} must satisfy

$$\begin{aligned} \bar{c}_{ikB^*} &= \min_B \left\{ c_{ikB} - (u_i - u_k) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b, w_B = k - i \right\} = \\ &= \min_B \left\{ p_B(W - i + 1) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b, w_B = k - i \right\} - (u_i - u_k). \end{aligned} \quad (5.12)$$

Note that fixing i, k determines the required weight for the batch B^* . Finding the batch B^* that minimizes this equation, for each given pair of indices $1 \leq i < k \leq W + 1$ and given batch processing time p_{B^*} , can be done by exploiting the DP state space of a family of knapsack problems, where items correspond to jobs and an additional constraint must be enforced on the total weight packed into the batch. Assume that the jobs are indexed by Longest Processing Time (LPT) order, so that $p_1 \geq p_2 \geq \dots \geq p_n$.

We define, for $r = 1, \dots, n$, the following problem:

$$g_r(\tau, \ell) = \max \left\{ \sum_{j=r}^n v_j y_j : \sum_{j=r}^n s_j y_j \leq \tau, \sum_{j=r}^n w_j y_j = \ell, y_j \in \{0, 1\} \right\}.$$

In this, $g_r(\tau, \ell)$ is the optimal value of a knapsack with profits v_j and sizes s_j , limited to jobs $r, r + 1, \dots, n$, total size $\leq \tau$ and total weight equal to ℓ . Variable y_j is set to 1, that is, $y_j = 1$, iff job j is included in the solution.

Optimal values for $g_r(\tau, \ell)$ can be recursively computed as

$$g_r(\tau, \ell) = \max \begin{cases} g_{r+1}(\tau - s_r, \ell - w_r) + v_r & (y_r = 1) \\ g_{r+1}(\tau, \ell) & (y_r = 0) \end{cases}$$

with boundary conditions

$$\begin{aligned}
g_r(\tau, w_r) &= \begin{cases} v_r & \text{if } s_r \leq \tau \text{ (} y_r = 1 \text{)} \\ 0 & \text{otherwise (} y_r = 0 \text{)} \end{cases} & r = 1, \dots, n, \tau = 0, \dots, b \\
g_r(\tau, 0) &= 0 & r = 1, \dots, n, \tau = 0, \dots, b \\
g_r(\tau, \ell) &= -\infty & \text{if } \ell > \sum_{i=r}^n w_i \text{ or } \ell < 0 \text{ or } \tau < 0.
\end{aligned}$$

The corresponding optimal job sets are denoted by $B_r(\tau, \ell)$; such sets can be retrieved by backtracking. Extending Property 2, the following Property 5 establishes that the state space $g_r(\tau, \ell)$ for $r \in L$ is sufficient for pricing all the relevant arcs.

Property 5. Consider the subset of jobs $L = \{1\} \cup \{j > 1 : p_j < p_{j-1}\}$ which holds one job, with index as small as possible, for each processing time represented in the problem. For any given pair of indices $i < k$, an arc with minimum reduced cost (i, k, B^*) is one of

$$(i, k, B_r(b, k - i)) \quad r \in L. \quad (5.13)$$

Proof. Every arc (i, k, B) can be shown to have a reduced cost not less than some of the arcs in (5.13). Let $\bar{c}_{ikB} = (W - i + 1)p_B - \sum_{j \in B} v_j - (u_i - u_k)$ be the reduced cost of an arc (i, k, B) . Recall that $w_B = k - i$, and the jobs are numbered in non-increasing order of processing times. Choose r as the smallest job index such that $p_r = p_B$. Note that $B \subseteq \{r, r + 1, \dots, n\}$ and $r \in L$. Consider knapsack $g_r(b, k - i)$ and the associated optimal subset $B_r^* = B_r(b, k - i)$. The batch B is a feasible solution for knapsack $g_r(b, k - i)$, hence $\sum_{j \in B} v_j \leq g_r(b, k - i)$; also, because of the choice of r , $p_{B_r^*} \leq p_r = p_B$. Thus:

$$\begin{aligned}
\bar{c}_{ikB} &= (W - i + 1)p_B - \sum_{j \in B} v_j - (u_i - u_k) \geq \\
&\geq (W - i + 1)p_{B_r^*} - g_r(b, k - i) - (u_i - u_k) = \bar{c}_{ikB_r^*}
\end{aligned} \quad (5.14)$$

□

Given the optimal multipliers from the RMP, Algorithm 5 (NEWCOLSW) generates a set of arcs with negative reduced cost, if some exist, or certifies optimality for (5.5)–(5.9) if none is found. It is worth noting that this algorithm is adapted from the previous Algorithm 1 developed for the unweighted version of the problem.

The size of the state space required for the pricing is bounded by $\mathcal{O}(nWb)$. A memoized CG table is used, so that the execution of the top-down recursion for computing an entry $g_r(\tau, \ell)$ is deferred until the first time the value is queried.

Algorithm 5 Pricing procedure (weighted).

```
1: function NEWCOLSW( $N, b, \mathbf{u}, \mathbf{v}$ )           ▷  $\mathbf{u}, \mathbf{v}$  = vectors of multipliers
2:   Sort and renumber jobs in  $N$  such that  $p_1 \geq p_2 \geq \dots \geq p_n$ ;
3:   Set  $H := \emptyset$ ;                       ▷ set of negative-reduced cost arcs
4:   Set  $W := \sum_{j=1}^N w_j$ ;
5:   for  $\ell = 1, \dots, W$  do                 ▷ for each weight
6:     for  $r = 1, \dots, n$  do                 ▷ for each job index
7:       Retrieve  $g_r(b, \ell)$  and set  $B := B_r(b, \ell)$ ;
8:       for  $i = 1, \dots, W - \ell + 1$  do     ▷ for each position
9:         Set  $k := i + \ell$ ;
10:        Compute  $\bar{c}_{ikB} = p_B(W - i + 1) - (u_i - u_k) - g_r(b, \ell)$ ;
11:        if  $\bar{c}_{ikB} < 0$  then                 ▷ if the reduced cost is negative...
12:          Set  $H := H \cup \{(i, k, B)\}$ ;     ▷ ... add the corresponding arc
13:        end if
14:      end for
15:    end for
16:  end for
17:  return  $H$ ;
18: end function
```

Algorithm 6 Generation of initial arcs (weighted).

```
1: function INITCOLSW( $N, b$ )
2:   Sort and renumber jobs in  $N$  such that  $p_1 \leq p_2 \leq \dots \leq p_n$ ;
3:   Set  $H := \emptyset$ ;                       ▷ set of initial arcs
4:   Set  $W := \sum_{j=1}^N w_j$ ;
5:   for  $i = 1, \dots, W$  do                 ▷ for each starting position
6:     for  $\ell = 1, \dots, n$  do                 ▷ for each job index
7:       Set  $B := \emptyset$ ;
8:       for  $j = \ell, \dots, 0$  do             ▷ for every previous job
9:         if  $\sum_{h \in B} s_h + s_j \leq b$  then   ▷ add this job only if it fits
10:          Set  $B := B \cup \{j\}$ ;
11:          Set  $k := i + \sum_{h \in B} w_h$ ;
12:          Set  $H := H \cup \{(i, k, B)\}$ ;
13:        end if
14:      end for
15:    end for
16:  end for
17:  return  $H$ ;
18: end function
```

Then, the value is kept in storage and accessed in $\mathcal{O}(1)$ time if it is queried again. NEWCOLSW exhibits three nested loops that account for a $\mathcal{O}(nW^2)$ complexity; taking into account the filling (memoized or not) of the DP table, the running time of the pricing procedure is bounded from above by $\mathcal{O}(\max(nW^2, nWb))$.

A minimal subset of starting columns is required to evaluate the first optimal multipliers and start generating new columns. To populate in a meaningful way the RMP, we order all jobs j in a non-decreasing order with regard to the processing time p_j and generate all possible batches, made by grouping subsequent jobs together, starting from every job in the sequence. These batches B are then added at the problem as new columns in every possible position (i, k, B) where $k - i = \sum_{j \in B} w_j$ as shown in Algorithm 6 (INITCOLSW), derived from the one developed for the unweighted version of the problem, Algorithm 2.

These INITCOLSW and NEWCOLSW algorithms can then be combined in a procedure for solving program (5.5)–(5.9) by CG; this continuous optimum is a lower bound for the original program (5.5)–(5.8), and will be called Column Generation Lower Bound (CG-LB) from now on. We note that this procedure is the same as the one described in Section 4.1.3 to obtain the lower bound for the unweighted version of the problem.

5.1.3 A path-based model

Master problem

Program (5.5)–(5.8) is basically made of a “flow” section (constraints (5.6)) and a “partition” section (constraints (5.7)). The flow section carries along with it some features of flow models, among them the presence of heavily degenerate bases in cases (like this) with a single source and a single sink. This can (possibly, but not certainly) lead to stalling/slow convergence behaviors.

In (5.5)–(5.8) the flow moves along arcs in the graph. Moving the flow on whole paths can be seen as a workaround for the problems stemming from degenerate bases. We consider the set \mathcal{P} of all possible paths $1 \rightarrow W + 1$ and define binary variables x_P for each $P \in \mathcal{P}$. Let us introduce a vector $\mathbf{a}_P \in \mathbb{N}^n$ where each component a_{Pj} counts the number of times that job j appears in the batches B of the arcs $(i, k, B) \in P$; we stress that here generally P is not necessarily a partition path, so more than one batch on the path might contain the same job j , or some job j can even fail to appear on the path P . The search for a minimum cost partition path is then captured by the following program, where $c_P = \sum_{(ikB) \in P} c_{ikB}$.

$$\text{minimize } \sum_{P \in \mathcal{P}} c_P x_P \quad (5.15)$$

$$\text{subject to } \sum_{P \in \mathcal{P}} x_P = 1 \quad (5.16)$$

$$\sum_{P \in \mathcal{P}} \mathbf{a}_P x_P = \mathbf{1} \quad (5.17)$$

$$x_P \in \{0, 1\} \quad P \in \mathcal{P} \quad (5.18)$$

We replace (5.18) with

$$x_P \geq 0 \quad P \in \mathcal{P} \quad (5.19)$$

when considering the continuous relaxation. Constraint (5.16) forces the integer flow to move along a single path. The vector constraint (5.17) forces the selected path to be a partition-path: actually (5.17) subsumes n scalar constraints enforcing that each job $j \in \{1, 2, \dots, n\}$ is counted exactly once in the selected path. If the continuous relaxation is considered, constraints (5.17) requires that summing up the number of times that each job is used over all the paths P with $x_P > 0$ must exactly add up to 1, for each job. We remind the reader that while only one path can take $x_P = 1$ in the solution of the binary program, the continuous relaxation can in general have several $x_P > 0$.

The dual of (5.15)–(5.19) is the following, with dual variables μ for constraint (5.16) and $\lambda_1, \dots, \lambda_n$ for the n scalar constraints from (5.17).

$$\begin{aligned} & \text{maximize } \mu + \sum_{j \in N} \lambda_j \\ & \text{subject to } \mu + \sum_{j=1}^n a_{Pj} \lambda_j \leq c_P \quad P \in \mathcal{P} \end{aligned}$$

Pricing

Given optimal multipliers/dual variables $\mu, \lambda_1, \dots, \lambda_n$, a path P^* with minimum reduced cost must satisfy

$$\bar{c}_{P^*} = \min_{P \in \mathcal{P}} \left\{ c_P - \mu - \sum_{j=1}^n a_{Pj} \lambda_j \right\} = \min_{P \in \mathcal{P}} \left\{ \sum_{(ikB) \in P} \left[c_{ikB} - \sum_{j \in B} \lambda_j \right] \right\} - \mu. \quad (5.20)$$

The problem in equation (5.20) amounts to determine a minimum cost path $1 \rightarrow W + 1$ on the very large graph G , using arcs with modified costs

$$c_{ikB} = p_B(W - i + 1) - \sum_{j \in B} \lambda_j.$$

Graph G is naturally layered, only arcs (i, k, B) with $i < k$ are present, and a minimum cost path $1 \rightarrow W + 1$ on G can be computed basically by an implementation of the shortest path algorithm by Gondran et al. [30]. Given a node k and a predecessor $i < k$, the cheapest arc from i to k must be associated with a batch B such that

$$\hat{c}_{ikB} = \min_B \left\{ p_B(W - i + 1) - \sum_{j \in B} \lambda_j : \sum_{j \in B} s_j \leq b, w_B = k - i \right\}. \quad (5.21)$$

Equation (5.21) is completely analogous to (5.12), and can be solved by relying on a DP table similar to the one described for the arc-based model:

$$g_r(\tau, \ell) = \max \left\{ \sum_{j=r}^n \lambda_j y_j : \sum_{j=r}^n s_j y_j \leq \tau, \sum_{j=r}^n w_j y_j = \ell, y_j \in \{0, 1\} \right\}.$$

Given such table, whose size is limited by $\mathcal{O}(nWb)$, the computation of a path with minimum reduced cost is done by Algorithm 7 (MINIMUMCOSTPATH) in a running time bounded by $\mathcal{O}(nW^2)$.

Algorithm 7 Minimum cost path procedure.

```

1: function MINIMUMCOSTPATH( $\mu, \lambda_1, \dots, \lambda_n$ )
2:   Set  $\Pi_1 := 0, \Pi_k := \infty$  for  $k = 2, \dots, W + 1$ ;
3:   for  $i = 0, \dots, W$  do                                      $\triangleright$  for each starting position
4:     for  $k = i + 1, \dots, W + 1$  do                            $\triangleright$  for each ending position
5:       for  $r \in N$  do                                            $\triangleright$  for each job
6:         Set  $\Pi_k := \min\{\Pi_k, \Pi_i + p_r(W - i + 1) - g_r(b, k - i)\}$ 
7:       end for
8:     end for
9:   end for
10:  return  $\Pi_{W+1} - \mu$ ;
11: end function

```

Again, a minimal subset of starting columns (paths in this case) is required to evaluate the first optimal multipliers and start generating new columns. To populate in a meaningful way the RMP, we first add to the problem the path corresponding to Azizoglu and Webster [5] heuristic. Then, we order all jobs j in a non-decreasing order with regard to the processing time p_j and partition them in batches, creating a new batch when the previous is full (that is, adding the next job in the previous batch would invalidate max capacity constraint), adding the resulting path to the

RMP. Finally, we generate a number of random sequences and partition them in batches again as we do with the ordered sequence.

5.2 Upper bounding: heuristics

This section outlines simple but effective rounding strategies for generating integer solutions from the fractional optimal solution of the relaxed model. The focus is kept on the arc-based model, because the arc-based model is especially well-suited for a strategy where one arc at a time is rounded to an integral flow value in order to build a partition path. Rounding a single (or few) arc variables in program (5.5)–(5.9) is easily managed in the LP solver, while partially rounding a path-variable in a program like (5.15)–(5.19) would require a trickier handling of the residual problem. Also, despite of the attractive feature of having a reduced number of constraints in the master problem, the path-based model turned out to be computationally heavier than the arc-based model (see Section 5.3 ahead).

Once program (5.5)–(5.9) has been solved, a lower bound is available; a minimal-effort strategy for getting an upper bound could be setting all variables back to the binary type and solve the integer version of the RMP by Branch-and-Bound (B&B), possibly truncating the process when a limit on computation time and/or processed branch nodes is reached. This approach, also called Price-and-Branch (P&B) as opposed to the exact B&P approach, was pursued in Alfieri et al. [2] and Alfieri et al. [3] for the special case of the problem where $w_j = 1$ for all $j \in N$. For the general $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ problem anyway, the large size of the involved LP models involved makes using a B&B impractical, especially for large instances.

We investigated simple strategies based on rounding fractional variables in order to generate feasible solutions within shorter computation times. This approach is inspired by Mourgaya and Vanderbeck [53] where rounding heuristics are applied to vehicle routing problems.

5.2.1 Variable Rounding Upper Bound

In the basic rounding approach, we build a partition path by rounding flow values, starting from the source and moving towards the sink node. Given the optimal fractional flow on the final RMP, we search for the arc (i, k, B) with maximal nonzero flow that lies as near as possible to the source, and round its flow value to 1. If the previous value of the flow on (i, k, B) was less than 1, the RMP is reoptimized, possibly adding other columns in the reoptimization process.

Note that rounding a fractional variable to 1 could in principle make the RMP unfeasible; such an event cannot happen in our implementation, because of how the initial set of columns is generated by Algorithm 6 (INITCOLSW); in particular, it

is worth noting that a partition path made of single-job arcs is always available in order to build a (possibly bad) feasible solution for the residual problem.

We then iterate this rounding step until an arc reaching the sink node $W + 1$ is fixed, thus completing a feasible solution. The value of such solution is called Variable Rounding Upper Bound (VR-UB) in the following, and Algorithm 8 sketches the procedure.

Algorithm 8 Variable Rounding Upper Bound (VR-UB) procedure.

```

1:  $G(V, A') \leftarrow$  optimum solution of the RMP;
2:  $\mathbf{x} \leftarrow \{x_{ikB} : (i, k, B) \in A'\};$  ▷ set of all RMP variables
3:  $i \leftarrow 1;$ 
4: while  $i < W + 1$  do
5:    $(i, k, B) \leftarrow \arg \max\{x_{ik'B'} : (i, k', B') \in A'\};$  ▷ arc with maximum flow
6:   if  $x_{ikB} = 1$  then ▷ if maximum flow is integral, no need to reoptimize
7:     Fix variable  $x_{ikB}$  to 1;
8:   else ▷ if maximum flow is not integral, reoptimize
9:     Fix variable  $x_{ikB}$  to 1;
10:     $G(V, A') \leftarrow$  optimum solution of the RMP;
11:     $\mathbf{x} \leftarrow \{x_{ikB} : (i, k, B) \in A'\};$  ▷ set of all RMP variables
12:   end if
13:    $i \leftarrow k;$ 
14: end while
15: VR-UB  $\leftarrow$  optimum solution of the RMP; ▷ integral solution

```

5.2.2 Early Rounding Upper Bound

The CG-LB plus VR-UB procedure delivers in a *relatively* short time both a lower and an upper bound for the $1|p\text{-batch}, b, \sigma_j|\sum w_j C_j$ problem, but for larger instances (for example, more than 50 jobs) and some size distributions the execution could go on for a considerable amount of time (more than 3 minutes for 60 jobs, more than 30 minutes for 100 jobs).

In order to speed up the construction of a partition path without losing too much of the quality of our bounds, a possible strategy consists of trusting the information gathered in the solution of the continuous relaxation before (sometimes even *well* before) convergence to the relaxed optimum is achieved, and proceeding to round variables anyway. This strategy is called *early rounding* in what follows. After a certain number of “unfruitful” iterations of CG exhibiting only a small improvement in objective function, an arc variable (i, k, B) with i as small as possible and maximum flow value is fixed to 1, by a logic analogous to the one used in Algorithm 8.

This obviously prevents the delivery of a valid lower bound at the end, offering in exchange a faster shrinking of the residual problems to be optimized in the next iterations. This tradeoff turns out to be practically advantageous, allowing to consistently reduce computation times without giving up much of the solution

quality. The procedure is summarized in Algorithm 9, where *threshold* and *steps* are parameters determining respectively the minimum relative improvement in objective function below which the iteration is considered unfruitful, and the maximum number of unfruitful iterations to be tolerated before a rounding is performed. The value of the resulting solution is called Early Rounding Upper Bound (ER-UB).

Algorithm 9 Early Rounding Upper Bound (ER-UB) procedure.

```

1:  $A' \leftarrow \text{INITCOLSW}(N, b)$ ;
2:  $W \leftarrow \sum_{j=1}^n w_j$ ,  $k \leftarrow 1$ ,  $s \leftarrow 0$ ;
3:  $z, \mathbf{u}, \mathbf{v} \leftarrow \text{optimum of } G(V, A')$ ; ▷ evaluating to obtain duals
4: while  $i < W + 1$  do
5:    $A' \leftarrow A' \cup \text{NEWCOLSW}(N, b, \mathbf{u}, \mathbf{v})$ ; ▷ generating new columns
6:    $z', \mathbf{u}, \mathbf{v} \leftarrow \text{optimum of } G(V, A')$ ; ▷ evaluating new optimum
7:   if  $(z - z')/z < \text{threshold}$  then ▷ if difference is less than threshold...
8:      $s \leftarrow s + 1$ ; ▷ ... count an “unfruitful” iteration
9:   else
10:     $s \leftarrow 0$ ;
11:   end if
12:   if  $s \geq \text{steps}$  then
13:      $(i, k, B) \leftarrow \arg \max\{x_{ik'B'} : (i, k', B') \in A'\}$ ; ▷ arc with maximum flow
14:     Fix variable  $x_{ikB}$  to 1;
15:      $i \leftarrow k$ ; ▷ new starting position is old ending position
16:      $s \leftarrow 0$ ; ▷ reset “unfruitful” counter
17:      $z, \mathbf{u}, \mathbf{v} \leftarrow \text{optimum of } G(V, A')$ ; ▷ evaluating to obtain duals
18:   end if
19: end while
20: ER-UB  $\leftarrow z$ ; ▷ integral solution

```

5.3 Computational results

All the tests ran in a Linux environment equipped with Intel Core i7-6500U CPU @ 2.50GHz processor; all algorithms have been implemented in D (using a language subset that practically guarantees C-like performances); the LP solver used was CPLEX 12.8.

A number of test instances were generated following the de-facto standard for this type of parallel batching problems; this is the same scheme used for tests presented in Section 4.3 with the addition of the job weights.

- The machine capacity was fixed to $b = 10$.
- The processing times p_j were randomly drawn from the uniform discrete distribution $[1, 100]$.

- The job sizes s_j were randomly drawn from four possible uniform discrete distributions, labeled by $\sigma \in \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

$$\begin{array}{ll} \sigma_1 : s_j \in [1, 10] & \sigma_3 : s_j \in [3, 10] \\ \sigma_2 : s_j \in [2, 8] & \sigma_4 : s_j \in [1, 5]. \end{array}$$

- The job weights w_j were drawn from the uniform discrete distribution $[1, 50]$.

A batch with ten instances for each σ class was generated, plus a larger batch with twenty-five examples for each σ for more extensive testing; the latter is named “Extended Set” in the following.

Tab. 5.1: CPU times and RMP size for arc-based model vs path-based model.

Param		Arc-based		Path-based	
n	σ	Time (s)	Columns	Time (s)	Columns
10	σ_1	0.05	2096	1.26	76
	σ_2	0.05	2017	0.89	85
	σ_3	0.03	1955	0.75	79
	σ_4	0.07	2344	1.55	90
20	σ_1	1.33	9215	118.24	459
	σ_2	0.98	8686	53.37	367
	σ_3	0.59	8487	92.74	667
	σ_4	3.30	13592	112.50	295
30	σ_1	9.90	23944	912.98	965
	σ_2	5.67	22907	664.42	1013
	σ_3	2.70	21858	605.11	1322
	σ_4	24.30	33242	1670.77	828

5.3.1 Performance of basic models

The first tests compared the performances of the arc-based model against the path-based model. Both models were run on the same set of instances in order to compute the continuous lower bound. The path-based model proved to be unable to handle instances with more than 30 jobs within reasonable computation times; detailed results are given in Tab. 5.1. The lower bound is (as expected by the theory) the same for both models, even if the computation times are heavily different.

Profiling the algorithms allows to observe that the solution processes for the two models have different bottlenecks.

- For the arc-based model, most of the CPU time is spent in the solution of the RMP; the typical basis in the master problem is quite degenerate, often exhibiting more than 80% of basic variables fixed to zero. On the other hand, a very short time is spent in the pricing routine: despite of the pseudopolynomial worst-case time complexity, thanks to the *memoization* approach usually a

small portion of the DP state space actually need to be exploited, and time measures (omitted in table for conciseness) show that usually about 15 % of the CPU time is spent generating new columns.

- For the path-based model, the basis is still heavily degenerate (apparently only slightly less than in the arc-based model); the size of the RMP (both in rows and columns) is considerably smaller than for the arc-based model, and the time spent solving the master is proportionally smaller. Anyway, a more efficient method for generating columns is required in order to make the path-based model competitive. The overall complexity of the pricing procedure in Algorithm 7 is comparable with that of Algorithm 5 used for the arc-based model, but whereas Algorithm 5 can generate a set of columns to be added to the RMP, only one column at a time will emerge from the application of Algorithm 7. This usually results in a snap reoptimization of the heavily degenerate RMP with only a small (or even null) decrease in the objective value and a new call to the pricing phase. The number of calls to the pricing procedure for the path-based model is so high that more than 90% of the computation time is spent in the pricing procedure, with a much higher number of iterations required to reach convergence.

In view of this, the use of arc-based models has been pursued in this work. The path-based model still remains interesting because of the smaller size of the RMP, but some more research effort has to be devoted to enhancing the pricing phase.

5.3.2 Generating feasible solutions by rounding

From now on, the arc-based continuous relaxation (5.5)–(5.9) is considered. Results with the basic variable rounding heuristic are presented first. In this approach, once convergence has been reached for the continuous relaxation, Algorithm 8 is applied, iteratively selecting an arc carrying a fractional flow and rounding it to 1.

Very few heuristics and/or relaxations are available in literature for this problem, among them are both a relaxation and a greedy heuristic developed by Azizoglu and Webster [5] in the context of a B&B algorithm for $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$. Such procedures are labeled AW-LB and AW-UB respectively.

All gaps in the following tables are evaluated by the relative difference between upper bound and lower bound, using the formula

$$\text{Gap (\%)} = \frac{\text{UB} - \text{LB}}{\text{UB}} \cdot 100\%$$

where LB is CG-LB in all cases, and UB is VR-UB for Tab. 5.2. In Tab. 5.3, Tab. 5.4 and Tab. 5.5 both VR-UB and ER-UB are considered as UB, depending on the column sets. Average, highest and lowest gap values are presented. In Tab. 5.3, Tab. 5.4

and Tab. 5.5 the values highlighted in **bold** refer to the best performing algorithm, VR-UB or ER-UB, considering execution time and gap quality.

“LB” and “UB” columns in Tab. 5.2 give a comparison between our lower/upper bounds and the lower/upper bounds by Azizoglu and Webster [5]. In “LB”, the higher is the value the better is the result for CG-LB; in “UB”, the lower is the value the better is the result for VR-UB. Finally, the “opt” column in Tab. 5.2 shows how many optima (amongst the 10 instances of the test set) the combination of CG-LB and VR-UB have certified; this happens when CG-LB equals VR-UB and the lower bound has zero variables with fractional value.

Tab. 5.2: Results for CG-LB and VR-UB on the weighted model. The third-to-last and second-to-last columns compare CG-LB and VR-UB with Azizoglu & Webster’s lower (AW-LB) and upper (AW-UB) bounds.

Param		Times (s)		Gap (%)			LB		UB	opt
n	σ	CG-LB	VR-UB	avg	worst	best	CG-LB AW-LB	VR-UB AW-UB		
10	σ_1	0.05	0.07	0.33	3.33	0.00	1.39	0.83	9	
	σ_2	0.05	0.06	0.04	0.40	0.00	1.31	0.94	9	
	σ_3	0.03	0.04	0.08	0.80	0.00	1.27	0.97	9	
	σ_4	0.07	0.08	0.02	0.18	0.00	1.41	0.82	9	
20	σ_1	1.33	1.55	0.46	3.84	0.00	1.29	0.87	3	
	σ_2	0.98	1.18	0.39	2.11	0.00	1.25	0.86	6	
	σ_3	0.59	0.82	0.31	2.24	0.00	1.21	0.94	7	
	σ_4	3.30	4.05	1.09	2.53	0.00	1.35	0.78	2	
30	σ_1	9.90	11.25	0.72	1.73	0.00	1.18	0.81	3	
	σ_2	5.67	7.62	0.69	3.63	0.00	1.21	0.86	1	
	σ_3	2.70	4.01	0.48	2.25	0.00	1.18	0.95	5	
	σ_4	24.30	31.10	1.25	4.03	0.00	1.27	0.73	2	
40	σ_1	28.09	41.38	0.74	2.38	0.00	1.20	0.76	3	
	σ_2	33.26	46.61	0.42	1.19	0.00	1.17	0.83	3	
	σ_3	17.50	24.18	0.80	2.13	0.00	1.18	0.87	4	
	σ_4	64.58	107.58	1.30	2.14	0.13	1.21	0.78	0	
50	σ_1	43.05	62.84	0.30	0.87	0.00	1.15	0.73	1	
	σ_2	40.43	55.88	0.41	1.37	0.00	1.16	0.84	2	
	σ_3	19.55	25.37	0.28	1.50	0.00	1.19	0.94	6	
	σ_4	128.90	203.24	1.10	2.25	0.00	1.18	0.76	1	
60	σ_1	127.81	205.69	0.61	1.54	0.00	1.17	0.72	1	
	σ_2	117.85	202.06	1.04	1.86	0.22	1.13	0.79	0	
	σ_3	59.00	92.81	0.70	1.57	0.01	1.17	0.88	0	
	σ_4	234.28	418.74	1.42	2.68	0.49	1.18	0.77	0	
80	σ_1	244.65	478.08	0.95	2.02	0.25	1.14	0.71	0	
	σ_2	289.18	570.80	0.48	1.43	0.07	1.11	0.78	0	
	σ_3	197.93	278.11	0.34	1.04	0.00	1.15	0.85	2	
	σ_4	563.20	1249.07	1.29	2.55	0.47	1.15	0.75	0	
100	σ_1	625.20	1363.70	0.44	1.34	0.06	1.11	0.76	0	
	σ_2	499.86	938.33	0.53	1.33	0.00	1.13	0.83	1	
	σ_3	291.80	408.78	0.22	0.46	0.00	1.16	0.93	1	
	σ_4	1245.92	2916.67	1.26	2.02	0.61	1.11	0.73	0	

Table 5.2 points to the performance comparison between CG-LB + VR-UB and AW-LB + AW-UB. The “Times (s)” columns report the average CPU times for computing the CG based lower bound (CG-LB) and the time needed to reach a completely integral solution by the rounding/fixing procedure (VR-UB), including both computational times. Instances from classes σ_1, σ_4 turn out to be computationally harder than the easy σ_3 instances, with σ_2 somehow in the middle; this conforms to what is reported in literature about similar problems. From the point of view of solution quality, both CG-LB and VR-UB strongly dominate AW-LB and AW-UB: our lower bound is 10% to 40% higher the one by Azizoglu & Webster, whereas our upper bound improves up to 25% over their. This does not come as a surprise, since AW-LB and AW-UB are cheap, quick-and-dirty components for a more complex B&B method. What really points to the added value of CG-LB + VR-UB is the very narrow optimality gap that can be certified for instances up to $n = 100$ jobs; such gap in no case raised above 5%, usually being much smaller. On the other hand the computation time required by such procedures is probably not acceptable for a heuristic algorithm on large (say $n > 50$ jobs) instances.

Tab. 5.3: Comparison between VR-UB and ER-UB on the weighted model. Reference lower bound for gap evaluation is CG-LB in both cases.

Param		Times (s)		VR-UB Gap (%)			ER-UB Gap (%)		
n	σ	VR-UB	ER-UB	avg	worst	best	avg	worst	best
20	σ_1	1.55	0.90	0.46	3.84	0.00	0.82	3.70	0.00
	σ_2	1.18	0.83	0.39	2.11	0.00	1.29	3.49	0.00
	σ_3	0.82	0.60	0.31	2.24	0.00	0.42	2.52	0.00
	σ_4	4.05	1.30	1.09	2.53	0.00	1.04	2.50	0.00
40	σ_1	41.38	14.67	0.74	2.38	0.00	1.79	3.79	0.20
	σ_2	46.61	13.75	0.42	1.19	0.00	1.62	2.85	0.62
	σ_3	24.18	12.40	0.80	2.13	0.00	0.88	2.58	0.08
	σ_4	107.58	15.84	1.30	2.14	0.13	1.81	3.44	0.94
60	σ_1	205.69	48.54	0.61	1.54	0.00	1.21	3.01	0.31
	σ_2	202.06	47.67	1.04	1.86	0.22	1.59	3.13	0.35
	σ_3	92.81	45.19	0.70	1.57	0.01	0.84	2.45	0.10
	σ_4	418.74	56.35	1.42	2.68	0.49	2.34	3.82	1.62
80	σ_1	478.08	129.63	0.95	2.02	0.25	1.02	1.93	0.35
	σ_2	570.80	116.23	0.48	1.43	0.07	1.31	2.17	0.33
	σ_3	278.11	112.99	0.34	1.04	0.00	0.53	1.12	0.12
	σ_4	1249.07	148.11	1.29	2.55	0.47	2.50	3.05	1.21
100	σ_1	1363.70	282.04	0.44	1.34	0.06	0.89	2.24	0.14
	σ_2	938.33	280.70	0.53	1.33	0.00	0.59	1.15	0.13
	σ_3	408.78	260.11	0.22	0.46	0.00	0.21	0.54	0.02
	σ_4	2916.67	302.78	1.26	2.02	0.61	1.89	3.07	1.27

Algorithm 9 (ER-UB) is specifically considered in order to overcome this performance issue; as explained in Section 5.2.2 the price to pay is that of no longer having a valid lower bound at the end of the computation. The experimental results show that solutions delivered by ER-UB retain most of the high-quality features of the

solutions delivered by VR-UB, while leading to drastic savings in computation times. This means that the (relaxation of the) RMP can provide valuable information for building a good feasible solution from the very first stages of the optimization. In the tests, the algorithm parameters were set to $steps = 3$ and $threshold = 0.001$ after a few sample trials, without attempting a fine calibration. In Tab. 5.3, where the gaps are evaluated against CG-LB, is shown that such gaps for ER-UB are generally not unacceptably higher than those of VR-UB, with much smaller CPU times.

Tab. 5.4: Comparison between VR-UB and ER-UB on the weighted model over the Extended Set. Reference lower bound for gap evaluation is CG-LB in both cases.

Param		Times (s)		VR-UB Gap (%)			ER-UB Gap (%)		
n	σ	VR-UB	ER-UB	avg	worst	best	avg	worst	best
20	σ_1	1.81	1.23	0.49	2.61	0.00	2.02	9.61	0.00
	σ_2	2.60	1.61	0.79	5.82	0.00	1.96	6.19	0.00
	σ_3	1.59	1.17	0.56	4.61	0.00	0.90	4.79	0.00
	σ_4	5.42	1.77	1.07	4.67	0.00	1.64	6.52	0.00
40	σ_1	37.44	12.81	0.89	2.82	0.00	1.37	4.03	0.20
	σ_2	48.32	15.03	0.77	3.54	0.00	1.55	3.69	0.22
	σ_3	17.88	12.04	0.55	2.85	0.00	0.83	3.27	0.00
	σ_4	86.01	15.34	1.52	3.51	0.01	2.62	6.81	0.54
60	σ_1	176.55	52.82	0.80	2.26	0.00	1.04	2.36	0.34
	σ_2	163.18	50.68	0.56	1.35	0.00	1.21	3.57	0.30
	σ_3	98.46	50.03	0.60	2.16	0.00	0.66	1.74	0.00
	σ_4	407.89	60.13	1.58	4.11	0.67	2.34	3.90	1.25
80	σ_1	627.62	147.53	0.72	2.11	0.10	1.02	2.57	0.13
	σ_2	522.87	145.79	0.65	2.31	0.00	0.85	1.58	0.23
	σ_3	269.40	128.21	0.43	1.51	0.00	0.53	1.53	0.01
	σ_4	1127.97	155.38	1.21	2.55	0.17	2.21	4.18	0.75
100	σ_1	1536.97	354.50	0.54	1.17	0.17	0.79	1.47	0.27
	σ_2	1128.45	280.00	0.52	1.12	0.02	0.81	1.76	0.21
	σ_3	680.02	277.80	0.37	1.27	0.00	0.35	1.10	0.00
	σ_4	2399.67	465.23	1.20	2.21	0.37	1.92	3.15	0.91

Results for VR-UB and ER-UB on an Extended Set of instances (25 per each σ class instead of 10) are reported in Tab. 5.4. This further analysis certifies, on a bigger instance pool, what already happened in Tab. 5.3: algorithm ER-UB runs in significantly smaller CPU times, while algorithm VR-UB gives slightly more accurate gaps. It is worth noting that, especially with the increase in number of jobs n , the percentage difference in gaps between VR-UB and ER-UB always remains very small.

In Tab. 5.5 a new job size distribution, $\sigma_5 : s_j \in [1, 50]$, and a new batch capacity, $b = 50$, are introduced. The $(\sigma_5, b = 50)$ combination is very similar to the $(\sigma_1, b = 10)$ one, because it can be seen as a “rescaling” of values over two different ranges (remember that $\sigma_1 : s_j \in [1, 10]$). Obviously this increase in job size granularity and in max capacity has an impact over time/space complexity of the algorithm, but it is

Tab. 5.5: Comparison between VR-UB and ER-UB on the weighted model over the Extended Set, considering job size distribution $\sigma_1 : s_j \in [1, 10]$ with batch capacity $b = 10$ and job distribution $\sigma_5 : s_j \in [1, 50]$ with batch capacity $b = 50$. Reference lower bound for gap evaluation is CG-LB in both cases.

Param			Times (s)		VR-UB Gap (%)			ER-UB Gap (%)		
n	b	σ	VR-UB	ER-UB	avg	worst	best	avg	worst	best
20	10	σ_1	1.81	1.23	0.49	2.61	0.00	2.02	9.61	0.00
	50	σ_5	2.98	1.62	0.54	4.70	0.00	1.41	7.28	0.00
40	10	σ_1	37.44	12.81	0.89	2.82	0.00	1.37	4.03	0.20
	50	σ_5	52.64	18.03	0.72	2.86	0.00	1.47	3.26	0.22
60	10	σ_1	176.55	52.82	0.80	2.26	0.00	1.04	2.36	0.34
	50	σ_5	321.99	70.51	0.94	2.15	0.00	1.29	3.80	0.23
80	10	σ_1	627.62	147.53	0.72	2.11	0.10	1.02	2.57	0.13
	50	σ_5	817.38	184.49	0.77	2.74	0.00	0.95	1.88	0.20
100	10	σ_1	1536.97	354.50	0.54	1.17	0.17	0.79	1.47	0.27
	50	σ_5	1684.46	372.23	0.71	1.88	0.09	1.04	2.43	0.33

not as high as one could have feared. For example in the case $n = 100$ the average time increase of VR-UB is 9.6% and of ER-UB is 5.0%. On the other hand, the gap quality of the two algorithms seems to not suffer much from the change in values of σ and b , especially with increasing number n of jobs. This is probably due to the fact that the two combinations are very similar except a “rescaling” of values.

5.4 Final remarks

The $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ problem can be effectively tackled by means of CG techniques. The CG-LB lower bound obtained by solving the relaxed arc-based model (5.5)–(5.9) is, to the writer’s knowledge, the sharpest bound currently available for this problem. The proposed bounding procedure generates arcs (columns) to be added to a RMP via DP. The resulting RMP are large but still manageable by an LP solver like CPLEX. The same bound could be obtained using a path-based model for the problem, like (5.15)–(5.19), resulting in a more compact RMP; anyway some more efficient pricing procedure is needed in order to speed up the optimization for such model.

The computation time required for computing CG-LB is still too high for using it into a B&P exact procedure, but a simple rounding strategy turns out to be quite effective in building heuristic solutions whose value are within a few percentage points from the optimum, with optimality gap certified by CG-LB. In order to speed up the heuristic procedure, variable rounding can be performed well before complete convergence to the relaxed optimum of the RMP. This loses the availability of a valid lower bound at the end of the heuristic procedure, but computational experience

shows that in practice the quality of the heuristic solution is not severely affected by this early rounding.

Whereas the arc-based CG model allows to generate good heuristic solutions in reasonable computation times, it seems unlikely that, with the current performances, it can be embedded into an exact B&P procedure: computing CG-LB is still computationally heavy for that kind of application. Some directions for future research are currently under consideration. Among others, the INITCOLSW procedure for initializing the column set in the RMP is not very sophisticated; significant speedups could be obtained being more aggressive in generating “good” columns in this startup phase. Also, an enhanced (exact or heuristic) pricing procedure for the path-based model could pave the way for interesting developments.

Unweighted total completion time: heuristics for multi-size jobs and incompatibility families



THIS chapter addresses another variant of the parallel batch scheduling problem. We recall that jobs in each batch are processed in parallel, so the processing time of the whole batch is equal to the maximum processing time amongst the jobs that compose it. We also recall that the objective is to compose the batches and to sequence them, in order to optimize a performance measure.

Usually, batches have a maximum size, depending on the technological characteristics of the process (Potts and Kovalyov [59]); for instance, the batch might have a maximum weight, or a maximum volume. Thus, to form the batches, the size constraint must be taken into account.

The batch scheduling problem is typical of semiconductor industries, mold manufacturing (Liu et al. [44]), medical device sterilization (Ozturk et al. [57]), heat-treating ovens (Mönch and Unbehaun [52]), chemical processes in tanks or kilns (Takamatsu et al. [65]), semiconductor and wafer fabrication industries (Mönch et al. [51]), and testing of electrical circuits (Hulett et al. [32]). Also additive manufacturing often requires batch production to optimize the chamber space, see for example the work by Zhang et al. [71]. However, here the processing times depend on different factors than those of conventional production, and the resulting scheduling problem could be different.

Sometimes, batch production needs to address multiple sizes in composing the batches; for instance, in additive manufacturing, chambers can produce various parts simultaneously, either by placing products in the 2-dimensional space or by stacking products and hence using the 3-dimensional space of the chamber. Thus, when creating the batches, constraints on several dimensions must be considered; for instance, if the additive manufacturing technology in use is able to stack parts in the chamber, then there is a maximum vertical span (height) and a maximum horizontal area that cannot be exceeded. These dimensions (height, horizontal area) will be called *sizes* in the following. Similarly, other industries could have the same batch requirements.

Moreover, for technological reasons, in shop floors where various product families are produced, batches must be composed of jobs of the same family. This is the case for *families* that need different manufacturing operations.

Due to the current industrial challenges, this chapter addresses the single machine batch scheduling problem with multiple sizes and incompatible job families. The aim is to find the batch schedule that minimizes the total completion time; thus, with the three-field standard notation by Graham et al. [31], the addressed problem is defined as $1|p\text{-batch}, b_i, \sigma_{ij}, \text{incomp}|\sum C_j$, where $p\text{-batch}$ defines the parallel batching, $\sigma_{ij} \leq b_i$ the multiple sizes, and incomp the incompatible job families.

To the writer's knowledge, this is the first attempt to address the batch scheduling problem with total completion time minimization, and jobs with multiple sizes and incompatible families. As the problem is NP-hard, a flow formulation is exploited to solve it through two Column Generation (CG) heuristics. The CG finds a continuous-relaxed solution, then two different heuristics from the literature are used to move from the continuous to the integer solution of the problem: the first is based on the so-called Price-and-Branch (P&B) heuristic (Alfieri et al. [3]) like the one used in Chapter 4, the other on a variable rounding procedure (Druetto and Grosso [20]) akin to the one presented in Chapter 5.

An extensive experimental campaign was run to compare the two heuristics, which both prove to be very effective for this scheduling problem. As the results will show, variable rounding is the most effective both in terms of computation time and quality of solution. Other useful insights for practical applications will be derived from the results.

6.1 Multiple sizes and incompatible families

Recalling the notation introduced in Chapter 3, a set of jobs $N = \{1, 2, \dots, n\}$ must be grouped in batches, and the batches must be scheduled on the machine. Setup times between batches are assumed to be negligible and are not considered.

The machine processes jobs in batches, and batches must respect physical constraints, such as maximum height, volume, weight, and so on. Let d be the number of different sizes to be addressed (that is, $d = 3$ if the physical constraints imply maximum height, volume and weight), and b_i be the batch capacity associated to the size i (with $i = 1, \dots, d$). Each job j has a processing time p_j , and d size values s_{ij} , with $i = 1, \dots, d$. For instance, if batches must respect a maximum volume (b_1) and a maximum height (b_2), then each job j will be characterized by its own volume s_{1j} and its own height s_{2j} .

In the machine, jobs must be clustered in batches, such that the sum of job sizes s_{ij} of all jobs in a batch does not exceed the batch dimension b_i (with $i = 1, \dots, d$). In addition, jobs are grouped into families that represent different process requirements. Hence, jobs from different families must be processed in different batches. There are n_f families, and each job j belongs to a family f , hence $f = 1, \dots, n_f$.

A solution to the problem is made by a batch schedule, that is, a sequence of feasible batches $S = B_1, \dots, B_{n_B}$. The total number of batches n_B is not known a priori, but it depends on how jobs are clustered in each solution. However, the number of batches n_B must be in the range between n_f and n , as there should exist at least one batch for each family, and there are at maximum n batches each composed of one single job.

All jobs in a batch B are assumed to be processed simultaneously and the processing time p_B of batch B equals the longest processing time of jobs contained in it; that is, $p_B = \max\{p_j : j \in B, j = 1, \dots, n\}$. Also, a job is completed when all other jobs in the same batch are completed, that is, the completion time C_j of each job equals the completion time of the batch it belongs to. The aim is to find the sequence $S = B_1, \dots, B_{n_B}$ that minimizes the total completion time.

In this work, the three combinations of constraints will be considered; not only the full problem $1|p\text{-batch}, b_i, \sigma_{ij}, \text{incomp}| \sum C_j$ with multi-size jobs and incompatible families, but also the multi-size problem without families $1|p\text{-batch}, b_i, \sigma_{ij}| \sum C_j$, and the family problem with single size-jobs $1|p\text{-batch}, b, \sigma_j, \text{incomp}| \sum C_j$.

6.2 Column Generation-based heuristics

The flow model described in Chapter 4 (Alfieri et al. [3]) is exploited and adapted to the problem at hand to develop two heuristic algorithms. The flow model considers the feasible batches as binary variables and associates a cost to each of them, whose sum is to be minimized in the objective function. Usual flow constraints guarantee the flow preservation from a source node to the last and a second set of constraints guarantee that each job is scheduled exactly once; for the details on the flow formulation, refer to Section 4.1.

Some modifications in the underlying Dynamic Programming (DP) structure of the flow model are made, and two heuristics are used to solve the problem. Both heuristics rely on an initial CG algorithm, which solves the continuous relaxation of the flow model. Then, the heuristics use different techniques to move from the continuous solution found by CG to the final integer solution of the problem, as will be explained in the following.

6.2.1 The CG-LB Column Generation algorithm

The CG starts from a restricted formulation of the relaxed continuous flow model and iteratively selects promising variables (that is, promising batches) through a so-called *pricing* procedure. The CG finds as an output the optimal solution of the continuous relaxation of the flow model. This solution is then used as a starting point to find the integer solution of the initial problem in the heuristics. The details

of the CG are explained in the following; refer to Section 4.1.2 for more details, since this section is an adaption to the approach previously described.

Initialization

First, an initialization phase is needed: we adapted the procedure INITCOLS described in Algorithm 2 to handle multiple sizes and families. Jobs are first sorted according to the Shortest Processing Time (SPT) rule; then, some feasible batches are generated by clustering together the sorted jobs until one of the maximum batch dimensions is reached, or until a job belonging to a different family is selected. When the batch is closed, it is placed on all possible schedule positions, and then, a new batch is developed with the same rule. The final output of the initialization is the set H , which contains a set of feasible batches.

The set H is set as the initial column set in the CG, which is used to solve the continuous relaxation of the flow model. The algorithm iterates between solving the Restricted Master Problem (RMP) (that is, the problem with only a subset of variables) and the pricing problem (the problem of selecting new columns to be added to the RMP) until the continuous optimum is found.

Pricing

In the pricing procedure, the dual multipliers associated with the RMP are computed to find the most promising variables, that is, those with the most negative reduced cost, to be included in the next iteration of the RMP. This procedure is an adaption of the NEWCOLS one described in Algorithm 1, with only a modification in the DP state space, as we are going to see in the following.

The pricing problem searches for the minimum reduced cost \bar{c}_{ikB^*} associated to arc (i, k, B^*) , among all arcs (i, k, B) that correspond to feasible batches. Let v_j be the dual multiplier corresponding to the constraints that guarantee that all jobs are included in the final schedule, and let u_i, u_k be the dual multipliers corresponding to the flow maintenance constraints. Recall that each job j belongs to a family f_j and has d different sizes s_{hj} ($h \in \{1, \dots, d\}$), corresponding to the d batch capacities b_h ; furthermore, the batch cardinality $|B|$ must be equal to $(k - i)$. Then, the problem can be formulated as follows:

$$\bar{c}_{ikB^*} = \min_B \left\{ \begin{array}{l} c_{ikB} - \sum_{j \in B} v_j : \\ \sum_{j \in B} s_{hj} \leq b_h \quad \forall h \in \{1, \dots, d\}, \\ f_j = f_{j'} \quad \forall j, j' \in B, \\ |B| = (k - i) \end{array} \right\} - (u_i - u_k). \quad (6.1)$$

For each new possible feasible batch B , the evaluation of the corresponding reduced cost depends on the position in the sequence, on the processing time and on the

jobs included in the batch. By isolating the parts depending on the position in an external loop that considers every i, k with $k > i$, the problem at hand is reduced to the computation of the feasible batch containing $k - i$ jobs with maximum value. This problem can be formulated as a cardinality-constrained multi-weight knapsack, which has to be solved for every starting and ending position. Indeed, jobs are the items, their dual multipliers are profits, and their sizes are the weights.

The solution space of the multi-weight knapsack problem with cardinality constraint is explored via a DP algorithm. Specifically, for every pair of i, k with $k > i$, and for each job $r = 1, \dots, n$, let $g_r(\tau_1, \dots, \tau_d, \ell)$ be the optimal knapsack with capacities τ_h for all $h \in \{1, \dots, d\}$, cardinality $\ell = k - i$, and that considers only jobs $r, r + 1, \dots, n$. The binary variable $y_j \in \{0, 1\}$ equals 1 if job j is selected for the knapsack, 0 otherwise. Then, the algorithm searches for:

$$g_r(\tau_1, \dots, \tau_d, \ell) = \max \left\{ \begin{array}{l} \sum_{j=r}^n v_j y_j : \\ \sum_{j=r}^n s_{1j} y_j \leq \tau_1, \dots, \sum_{j=r}^n s_{dj} y_j \leq \tau_d, \\ \sum_{j=r}^n y_j = \ell, f_j = f_r, y_j \in \{0, 1\} \end{array} \right\};$$

this can be computed (Kellerer et al. [36]) with the following recursion

$$g_r(\tau_1, \dots, \tau_d, \ell) = \max \left\{ \begin{array}{l} g_{r'} \left(\begin{array}{l} \tau_1 - s_{1r}, \dots, \tau_d - s_{dr}, \\ \ell - 1 \end{array} \right) + v_r \quad (y_r = 1) \\ g_{r'}(\tau_1, \dots, \tau_d, \ell) \quad (y_r = 0) \end{array} \right.$$

where r' is the next job in the Longest Processing Time (LPT) ordering that belongs to the same family of the job that started the recursion; in this way, family incompatibilities are addressed. The boundary conditions are as follows:

$$g_r(\tau_1, \dots, \tau_d, 1) = \begin{cases} v_r & \text{if } s_{hr} \leq \tau_h \quad (y_r = 1) \\ 0 & \text{otherwise} \quad (y_r = 0) \end{cases} \quad r = 1, \dots, n, \tau_h = 0, \dots, b_h \quad (6.2)$$

$$g_r(\tau_1, \dots, \tau_d, 0) = 0 \quad r = 1, \dots, n, \tau_h = 0, \dots, b_h \quad (6.3)$$

$$g_r(\tau_1, \dots, \tau_d, \ell) = -\infty \quad \text{if } \ell > n - r + 1 \text{ or } \tau_h < 0. \quad (6.4)$$

Equations (6.2)–(6.4) hold for all $h \in \{1, \dots, d\}$ since all sizes must be considered. It is worth noting that the DP state space increases of one dimension for every size of the jobs.

The variables with negative reduced cost found through this pricing procedure are added to the RMP, and the procedure is repeated until no negative reduced costs are found. The final solution of the CG is the optimum for the continuous problem, and a lower bound for the integer problem, called Column Generation Lower Bound (CG-LB) in the following.

6.2.2 The CG-UB and VR-UB heuristic procedures

Two heuristics are proposed to find an integer solution for the problem, and both starts from the lower bound given by CG-LB.

The first heuristic is called Column Generation Upper Bound (CG-UB), and it is the same P&B approach described in Section 4.1.3 (Alfieri et al. [3]); refer to Algorithm 3 for more details. Once the continuous optimum CG-UB is found, the variable domain of the problem is changed from continuous to binary, and the Mixed-Integer Program (MIP) is solved to get a heuristic integer solution, namely CG-UB. This evaluation can be quite slow, especially when the number of jobs increases: solving such a MIP to the optimum is NP-hard, indeed. Some strategies to speed up the evaluation could include stopping the evaluation after a fixed number of open nodes (but that amount should increase with the number of considered jobs), or after a certain time limit has been reached (but it can lead to large CG-UB over CG-LB gaps), or after a certain gap is reached (but the timing to reach such a gap could be too slow again). A time limit will be set for the numerical results of CG-UB.

The second heuristic is the Variable Rounding Upper Bound (VR-UB), and consists in the variable rounding procedure already described in Section 5.2.1 (Druetto and Grosso [20]). Differently from CG-UB, this approach generates good upper bounds within shorter computation times. Promising variables are sequentially fixed to 1, that is, a batch is forced to be included in the solution. Then, the leftover part of the RMP is re-optimized with continuous variables, up to the point when the entire sequence is established. By doing so, there is no need to solve a model with integer variables since they are selected manually, and only the continuous relaxation of the MIP is solved (which is way faster than solving the integer version).

The detailed variable rounding procedure can be seen in Algorithm 8, and it works as follows. First, CG-LB is computed and the RMP is populated; then, among all columns that start from $i = 1$, that is, all batches in the first position of the sequence, the one whose flow value is the closest to 1 is selected and enforced to be part of the solution. Also, no other columns can start from the same position. The procedure is repeated on the remaining part of the problem, by keeping all the already generated columns; then, again, a new selection is made among all columns that start from $i = k$, where k was the ending position of the column fixed in the previous step. The CG and variable rounding procedures are iterated until the value $i = |N| + 1$ is reached, that is, when there exist a sequence of batches from 1 to $|N| + 1$, where all non-zero flow variables are equal to 1; this is the heuristic solution VR-UB for the original problem.

It is worth noting that Algorithm 8, implemented for the *weighted* version of the problem $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$, can seamlessly run over all *unweighted* versions

of the problem such as $1|p\text{-batch}, b_i, \sigma_{ij}, \text{incomp}| \sum C_j$ by changing only the **while** condition in line 4 from $i < W + 1$ to $i < |N| + 1$.

The VR-UB procedure, as the numerical results will show, is substantially faster than CG-UB and gives gaps of almost the same quality.

6.3 Computational results

Random instances are generated to test the proposed algorithm. The generation approach (Alfieri et al. [3], Druetto and Grosso [20]) already used in Section 4.3 and Section 5.3, with the addition of job families and considering jobs with more than one size, is here replicated.

In the experiment, some factors are varied to test the proposed approach in various scenarios. Specifically, the larger is the number of jobs n , the more complex is the problem to be solved; hence, n has been varied in $n = \{20, 40, 60, 80, 100, 200\}$. Jobs have various sizes, and each size can be sampled from two different uniform distributions: $\sigma_{10} : s_{ij} \in [1, 10]$ and $\sigma_5 : s_{ij} \in [1, 5]$; the results should confirm that smaller intervals make the problem more complex, as more feasible batches can be created. The evaluated numbers of sizes per job are $d = \{1, 2, 3\}$, to test the efficiency of the approach in the multi-size cases. For each size, a distribution must be chosen between σ_{10} and σ_5 and, for each value of d , all possible combinations of σ_{10} and σ_5 are evaluated. Last, the tested numbers of families are $n_f = \{1, 3, 5, 7, 10\}$, to address the case of incompatible job families. Some system characteristics have been fixed as parameters in all the instances. Specifically, for each job, the processing time is sampled from a uniform distribution $p_j \in [1, 100]$. If more than one family is considered, the family is randomly assigned to every job with equal probability. Also, for each size, the batch capacity b_i is fixed to 10, as commonly used in the literature (Azizoglu and Webster [6]).

All in all, 174 combinations of factors are tested. For each combination, 10 instances are solved, thus leading to 1740 experiments. For conciseness purposes, only results for $n = \{20, 60, 100, 200\}$ are shown below; the trend highlighted by this subset of experiments is confirmed by the experiments on the other sizes. Beside the instances generated as mentioned above, additional tests are run with $b_i = 50$ and with size distribution $\sigma_{50} : s_{ij} \in [1, 50]$. The aim is to test the algorithm performance in the case where jobs have sizes with higher granularity.

Two heuristic algorithms are compared: the CG combined with the MIP solver (CG-UB), and the CG with the variable rounding procedure (VR-UB). The proposed methods are developed in C++, and the optimization procedure is done by calling the CPLEX solver, version 12.9. Tests are run on a computer having a Intel Core i7 CPU @ 3.70GHz with 32 GB of RAM.

In all the tables, each row reports various statistics for a single combination of factors (grouping together the 10 instances) on the performance of CPLEX integer solution (CG-UB) and of the variable rounding procedure (VR-UB). Specifically, each row is characterized by:

- the number of jobs n ;
- the distributions for each size (columns s_1, s_2, s_3 contain respectively the distribution of size 1, 2 or 3, if jobs have one; else – is displayed);
- the number of incompatible families n_f ;
- the number of reached optima (opt), knowing that the optimum is reached when CG-LB is equal to CG-UB and/or VR-UB;
- the average computational time (Time (s)) over all the 10 instances, expressed in seconds;
- the average percentage gap (Gap (%)) over all the 10 instances.

Percentage gaps between heuristics (CG-UB, VR-UB) and the lower bound (CG-LB) are evaluated as

$$\text{CG-UB Gap (\%)} = \frac{\text{CG-UB} - \text{CG-LB}}{\text{CG-UB}} \cdot 100\%,$$

$$\text{VR-UB Gap (\%)} = \frac{\text{VR-UB} - \text{CG-LB}}{\text{VR-UB}} \cdot 100\%.$$

The CG procedure that generates promising batches and builds a fractional solution works very fast. Finding the optimal integer solution using those batches is instead a time consuming issue for large instances. For this reason, a time limit of 100 seconds has been set; when this limit is reached, the `limit` diciture is shown in the relevant column.

6.3.1 Standard instances ($b_i = 10$)

All the results are shown separately for the following cases: multi-size with no families (Tab. 6.1); incompatible families with one size (Tab. 6.2); multi-size and incompatible families (Tab. 6.3 and Tab. 6.4, showing only the cases of 3 and 7 families for reasons of conciseness). Results of single-size single-family instances (that is, $1|p\text{-batch}, b, \sigma_j| \sum C_j$ problem, which CG-UB results are discussed in Section 4.3.1) are not reported here, for reasons of conciseness.

First, both the CG-UB and VR-UB approaches are very fast; indeed, for instances up to 60 jobs, the overall computation time is of the order of one second. Across all instances, the largest (maximum, not reported in the tables) computation time

for VR-UB is 96.2 seconds, while CG-UB sometimes reaches the 100 seconds time limit.

Tab. 6.1: Computational results of CG-UB and VR-UB for the multi-size case.

n	s_1, s_2, s_3	opt	CG-UB		VR-UB	
			Time (s)	Gap (%)	Time (s)	Gap (%)
20	$\sigma_{10}, \sigma_{10}, -$	7	0.03	0.27	0.02	0.20
20	$\sigma_{10}, \sigma_5, -$	6	0.03	0.52	0.02	0.36
20	$\sigma_5, \sigma_5, -$	2	0.05	1.13	0.02	1.11
60	$\sigma_{10}, \sigma_{10}, -$	0	0.19	0.30	0.14	0.48
60	$\sigma_{10}, \sigma_5, -$	0	0.50	0.53	0.30	0.86
60	$\sigma_5, \sigma_5, -$	0	10.74	2.01	0.97	1.85
100	$\sigma_{10}, \sigma_{10}, -$	1	1.17	0.16	0.99	0.36
100	$\sigma_{10}, \sigma_5, -$	0	8.55	0.60	1.79	1.08
100	$\sigma_5, \sigma_5, -$	0	97.66	3.76	4.11	1.84
200	$\sigma_{10}, \sigma_{10}, -$	0	20.91	0.12	7.91	0.29
200	$\sigma_{10}, \sigma_5, -$	0	97.40	0.51	11.51	0.60
200	$\sigma_5, \sigma_5, -$	0	limit	56.80	27.91	1.37
20	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	10	0.03	0.00	0.03	0.00
20	$\sigma_{10}, \sigma_{10}, \sigma_5$	8	0.04	0.06	0.03	0.07
20	$\sigma_{10}, \sigma_5, \sigma_5$	3	0.05	0.78	0.05	1.29
20	$\sigma_5, \sigma_5, \sigma_5$	0	0.07	1.94	0.07	2.34
60	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	0.18	0.12	0.34	0.14
60	$\sigma_{10}, \sigma_{10}, \sigma_5$	1	0.25	0.34	0.60	0.70
60	$\sigma_{10}, \sigma_5, \sigma_5$	0	0.40	0.49	0.87	1.08
60	$\sigma_5, \sigma_5, \sigma_5$	0	7.24	1.62	2.09	1.81
100	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	4	0.83	0.03	1.84	0.08
100	$\sigma_{10}, \sigma_{10}, \sigma_5$	4	1.21	0.10	2.69	0.24
100	$\sigma_{10}, \sigma_5, \sigma_5$	0	6.08	0.51	5.75	0.94
100	$\sigma_5, \sigma_5, \sigma_5$	0	82.57	1.88	9.04	1.33
200	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	4	5.76	0.03	13.90	0.08
200	$\sigma_{10}, \sigma_{10}, \sigma_5$	0	12.05	0.10	25.21	0.28
200	$\sigma_{10}, \sigma_5, \sigma_5$	0	90.16	0.34	44.27	0.69
200	$\sigma_5, \sigma_5, \sigma_5$	0	limit	22.85	78.35	1.14

The instances with sizes distributed following σ_5 are more difficult than those following σ_{10} . Indeed, if comparing the instances in Tab. 6.2 with 200 jobs, the average computation time of CG-UB moves from the order of 20 seconds for σ_{10} cases to times larger than 100 seconds for σ_5 . The same is true for VR-UB, although computation times are lower for both cases. As σ_5 reduces the maximum size a job can have, there are more possible combinations of jobs in a single batch, thus both computation times and gaps increase.

However, if more than one size constraint is considered (Tab. 6.1, Tab. 6.3 and Tab. 6.4), even with σ_5 the number of combinations decreases, and the DP that searches for feasible batches is able to cut uninteresting combinations. Thus, the greater the numerosity of job sizes, the faster the algorithm works.

The number of jobs negatively impacts the computation time and the gaps, both for CG-UB and VR-UB. Specifically, CG-UB often needs more than 100 seconds to solve instances with 200 jobs, especially in the cases of σ_5 size distributions. Instead, VR-UB is able to handle these instances in most cases, but its computation time largely increases with respect to the instances with a small number of jobs.

Moreover, with smaller numbers of jobs, both algorithms are more likely able to find optimal solutions; indeed, the number of optimal solutions (opt) is larger with smaller instances.

Tab. 6.2: Computational results of CG-UB and VR-UB for the incompatible families case.

n	s_1	n_f	opt	CG-UB		VR-UB	
				Time (s)	Gap (%)	Time (s)	Gap (%)
20	σ_{10}	3	6	0.04	0.54	0.01	1.28
20	σ_{10}	5	7	0.03	0.44	0.01	0.49
20	σ_{10}	7	7	0.04	0.41	0.01	0.42
20	σ_{10}	10	7	0.04	0.22	0.01	0.43
60	σ_{10}	3	0	0.29	1.34	0.15	1.97
60	σ_{10}	5	0	0.19	0.65	0.09	1.17
60	σ_{10}	7	3	0.18	0.76	0.06	1.08
60	σ_{10}	10	2	0.12	0.83	0.06	1.11
100	σ_{10}	3	0	2.51	0.76	1.24	1.31
100	σ_{10}	5	0	1.29	0.92	1.01	1.70
100	σ_{10}	7	0	1.28	0.81	0.81	1.18
100	σ_{10}	10	0	0.97	1.21	0.74	1.87
200	σ_{10}	3	0	53.45	0.58	8.63	1.01
200	σ_{10}	5	0	16.33	0.74	8.05	1.52
200	σ_{10}	7	0	16.94	0.83	6.58	1.39
200	σ_{10}	10	0	11.09	0.80	5.95	1.39
20	σ_5	3	3	0.05	1.12	0.01	1.43
20	σ_5	5	4	0.04	1.34	0.01	2.15
20	σ_5	7	5	0.04	0.63	0.01	1.09
20	σ_5	10	6	0.04	0.99	0.01	1.25
60	σ_5	3	0	2.05	2.49	0.69	3.30
60	σ_5	5	0	0.75	2.72	0.51	3.40
60	σ_5	7	0	0.61	2.79	0.40	3.51
60	σ_5	10	0	0.28	2.02	0.26	2.75
100	σ_5	3	0	87.51	3.19	3.50	3.11
100	σ_5	5	0	24.05	2.39	2.72	3.34
100	σ_5	7	0	8.13	2.49	2.30	3.02
100	σ_5	10	0	3.00	2.02	2.02	2.66
200	σ_5	3	0	limit	40.70	23.97	2.35
200	σ_5	5	0	limit	39.83	18.71	2.94
200	σ_5	7	0	limit	5.41	17.14	3.23
200	σ_5	10	0	limit	3.86	14.75	3.00

When incompatible families are addressed, a noticeable difference in computation times can be seen when the number of families increases. This is shown in Tab. 6.2,

Tab. 6.3 and Tab. 6.4. In the former tables, the average computation time decreases with the increase of the number of families.

Tab. 6.3: Computational results of CG-UB and VR-UB for the multi-size and incompatible families case, with two sizes.

n	s_1, s_2, s_3	n_f	opt	CG-UB		VR-UB	
				Time (s)	Gap (%)	Time (s)	Gap (%)
20	$\sigma_{10}, \sigma_{10}, -$	3	9	0.02	0.06	0.01	0.31
20	$\sigma_{10}, \sigma_{10}, -$	7	10	0.02	0.00	0.01	0.00
20	$\sigma_{10}, \sigma_5, -$	3	6	0.04	0.86	0.01	0.90
20	$\sigma_{10}, \sigma_5, -$	7	4	0.04	0.40	0.01	1.33
20	$\sigma_5, \sigma_5, -$	3	2	0.07	1.18	0.02	1.61
20	$\sigma_5, \sigma_5, -$	7	2	0.07	1.92	0.01	2.79
60	$\sigma_{10}, \sigma_{10}, -$	3	2	0.13	0.29	0.10	0.68
60	$\sigma_{10}, \sigma_{10}, -$	7	5	0.09	0.16	0.06	0.36
60	$\sigma_{10}, \sigma_5, -$	3	0	0.33	1.00	0.18	1.41
60	$\sigma_{10}, \sigma_5, -$	7	2	0.12	0.82	0.11	1.09
60	$\sigma_5, \sigma_5, -$	3	0	1.36	2.38	0.77	3.79
60	$\sigma_5, \sigma_5, -$	7	0	0.43	2.46	0.43	4.14
100	$\sigma_{10}, \sigma_{10}, -$	3	3	0.78	0.32	0.77	0.57
100	$\sigma_{10}, \sigma_{10}, -$	7	2	0.47	0.45	0.61	0.73
100	$\sigma_{10}, \sigma_5, -$	3	0	1.77	0.50	1.39	0.93
100	$\sigma_{10}, \sigma_5, -$	7	0	1.00	0.96	0.99	1.52
100	$\sigma_5, \sigma_5, -$	3	0	75.00	2.26	3.59	2.42
100	$\sigma_5, \sigma_5, -$	7	0	8.74	2.17	2.55	2.98
200	$\sigma_{10}, \sigma_{10}, -$	3	1	6.36	0.17	6.35	0.34
200	$\sigma_{10}, \sigma_{10}, -$	7	0	4.83	0.25	4.99	0.53
200	$\sigma_{10}, \sigma_5, -$	3	0	52.38	0.54	11.12	1.16
200	$\sigma_{10}, \sigma_5, -$	7	0	14.58	0.81	9.06	1.63
200	$\sigma_5, \sigma_5, -$	3	0	limit	28.97	24.27	2.24
200	$\sigma_5, \sigma_5, -$	7	0	limit	6.38	20.17	2.50

For instance (Tab. 6.2), with $n = 100$ and $s_1 = \sigma_{10}$, the average computation time goes from 2.51 with 3 families to 0.97 with 10 families for CG-UB, and from 1.24 to 0.74 for VR-UB. When multiple sizes are involved, the same decrease in computation time is shown in Tab. 6.3 and Tab. 6.4; interestingly, if σ_5 distribution is given to the sizes, the difference in computation time with different families is even larger. As an example, if comparing the cases $\{n = 100, m = 3, s_i = \sigma_{10}\}$ and $\{n = 100, m = 3, s_i = \sigma_5\}$, the difference in the average computation times from 3 to 7 families for the CG-UB moves from the order of 0.68 seconds to the order of 39.41 seconds.

In the same instances, for the VR-UB approach, there is no difference between computation times of 3 and 7 families for the σ_{10} instances, and there is a difference of the order of 1.52 seconds for the σ_5 instances.

Tab. 6.4: Computational results of CG-UB and VR-UB for the multi-size and incompatible families case, with three sizes.

n	s_1, s_2, s_3	n_f	opt	CG-UB		VR-UB	
				Time (s)	Gap (%)	Time (s)	Gap (%)
20	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	10	0.04	0.00	0.03	0.00
20	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	10	0.03	0.00	0.02	0.00
20	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	7	0.07	0.56	0.04	0.48
20	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	8	0.03	0.12	0.03	0.47
20	$\sigma_{10}, \sigma_5, \sigma_5$	3	4	0.08	0.91	0.04	1.16
20	$\sigma_{10}, \sigma_5, \sigma_5$	7	6	0.04	0.88	0.04	1.19
20	$\sigma_5, \sigma_5, \sigma_5$	3	1	0.09	2.38	0.06	2.99
20	$\sigma_5, \sigma_5, \sigma_5$	7	3	0.05	1.61	0.05	2.50
60	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	8	0.16	0.03	0.31	0.16
60	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	7	0.13	0.18	0.32	0.36
60	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	3	0.26	0.35	0.44	0.43
60	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	6	0.16	0.37	0.44	0.59
60	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	0.50	0.88	0.91	1.51
60	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	0.27	1.05	0.64	1.76
60	$\sigma_5, \sigma_5, \sigma_5$	3	1	1.39	1.84	1.61	2.14
60	$\sigma_5, \sigma_5, \sigma_5$	7	0	0.36	1.98	1.19	2.61
100	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	6	1.08	0.04	1.04	0.07
100	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	5	0.40	0.20	1.05	0.23
100	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	2	1.31	0.20	2.25	0.36
100	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	1	0.65	0.42	2.33	0.80
100	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	2.56	0.58	4.05	1.17
100	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	1.10	0.81	4.07	1.32
100	$\sigma_5, \sigma_5, \sigma_5$	3	0	43.04	1.57	7.97	2.04
100	$\sigma_5, \sigma_5, \sigma_5$	7	0	3.63	1.69	6.45	2.88
200	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	3	5.07	0.03	15.40	0.09
200	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	4	2.92	0.10	8.98	0.15
200	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	0	8.88	0.17	19.70	0.36
200	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	1	6.13	0.27	21.78	0.60
200	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	38.87	0.46	40.75	0.91
200	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	11.20	0.60	32.97	1.10
200	$\sigma_5, \sigma_5, \sigma_5$	3	0	limit	9.52	71.87	1.86
200	$\sigma_5, \sigma_5, \sigma_5$	7	0	limit	4.12	59.99	2.37

As explained in Section 6.2.1, the search in the state space from job r that belongs to family f can be restricted to the subset of jobs belonging to the same family. This modification of the search procedure does not impact on the structure of the state space, and leads to a decrease in computation time proportional to the number of families.

Lastly, comparing the results of CG-UB and VR-UB across all results, the proposed variable rounding tends to be faster than solving the MIP, and obtains comparable

or better performance. Specifically, for the single-size multi-family cases VR-UB is always faster than CG-UB. Indeed, Tab. 6.2 shows that average computation times are smaller for VR-UB in all the cases. When the single-family multi-size cases are considered (Tab. 6.1), CG-UB and VR-UB have comparable performance in the instances with a small number of jobs and σ_{10} distribution. However, VR-UB achieves remarkable improvements in gaps for the cases with large numbers of jobs and sizes with σ_5 distributions. For instance, in the experiments with $\{n = 200, s_i = \sigma_5\}$, the average gap moves from 56.80% (two sizes) and 22.85% (three sizes) for CG-UB to 1.37% (two sizes) and 1.14% (three sizes) for VR-UB, respectively. It is worth noting that the large CG-UB gap for larger instances is due to the enforced time limit. In terms of computation times, VR-UB results to be faster than CG-UB in almost all the cases; also, the latter often reaches the time limit when 200 jobs are considered. For the multi-size multi-family case, the same considerations hold, proving the outperforming of VR-UB with respect to CG-UB.

Although some gaps are slightly better for CG-UB (except for the cases where the time limit is exceeded), the relative difference is not worth the extra time required in comparison to the faster VR-UB. For instance, consider the single-size multi-family case with $\{n = 200, s_1 = \sigma_{10}, n_f = 7\}$ in Tab. 6.2. The algorithm CG-UB finds the optimum value (without reaching the time limit) in an average time of almost 17 seconds, with a mean gap equal to 0.83%; on the same instances VR-UB takes only 8 seconds in the worst (maximum, not reported in the tables) case, way less than the half of CG-UB average case, with an average gap equal to 1.39%, only half a point worse than CG-UB.

As the additional tests of Section 6.3.2 show, the efficiency of the algorithms is not affected for instances with larger job sizes and batch capacity. The average gaps are competitive for both algorithms; however, they suffer from the computation point of view since the pricing procedure becomes more difficult in these cases.

6.3.2 Extra instances ($b_i = 50$)

Table 6.5 shows the results of some additional tests; in these tests, a different size distribution and a different batch capacity are considered. The instances were generated with all jobs sizes sampled from a uniform distribution $\sigma_{50} : s_{ij} \in [1, 50]$ while the batch capacity b_i is set to 50 for all sizes $i = 1, \dots, d$. The tests are run for all combinations used in the previous Section 6.3.1. For issues related to excessive RAM usage, tests for instances up to 100 jobs are run.

The results on computation times show that having jobs with *higher granularity* with regards to their packing in batches, that is, jobs with sizes included in a wider interval, makes the problem more difficult to solve. The pricing procedure requires

in fact to optimally solve the cardinality-constrained multi-weight knapsack, which becomes harder when the number of feasible batches that can be formed increases.

Tab. 6.5: Computational results of CG-UB and VR-UB for the multi-size case, for the incompatible families case, and for the multi-size and incompatible families case over the extra instances.

n	s_1, s_2, s_3	n_f	opt	CG-UB		VR-UB	
				Time (s)	Gap (%)	Time (s)	Gap (%)
MULTI-SIZE (SINGLE FAMILY)							
20	$\sigma_{50}, \sigma_{50}, -$	1	6	0.06	0.67	0.04	0.84
60	$\sigma_{50}, \sigma_{50}, -$	1	2	0.60	0.18	0.33	0.30
100	$\sigma_{50}, \sigma_{50}, -$	1	1	3.76	0.12	1.83	0.30
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	10	1.25	0.00	0.58	0.00
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	7	34.27	0.15	5.26	0.23
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	4	limit	0.12	18.19	0.06
INCOMPATIBLE FAMILIES (SINGLE SIZE)							
20	$\sigma_{50}, -, -$	3	7	0.03	0.25	0.02	0.31
20	$\sigma_{50}, -, -$	5	6	0.03	0.72	0.02	0.84
20	$\sigma_{50}, -, -$	7	10	0.02	0.00	0.02	0.00
20	$\sigma_{50}, -, -$	10	8	0.02	0.29	0.02	0.29
60	$\sigma_{50}, -, -$	3	1	0.21	0.84	0.21	1.43
60	$\sigma_{50}, -, -$	5	0	0.23	0.97	0.13	1.29
60	$\sigma_{50}, -, -$	7	0	0.17	0.80	0.10	1.57
60	$\sigma_{50}, -, -$	10	0	0.15	0.78	0.09	1.09
100	$\sigma_{50}, -, -$	3	0	2.49	0.53	1.62	1.07
100	$\sigma_{50}, -, -$	5	0	2.50	1.33	1.38	1.89
100	$\sigma_{50}, -, -$	7	0	1.32	1.21	1.06	1.98
100	$\sigma_{50}, -, -$	10	0	0.91	0.73	0.94	1.44
MULTI-SIZE AND INCOMPATIBLE FAMILIES							
20	$\sigma_{50}, \sigma_{50}, -$	3	9	0.04	0.05	0.03	0.14
20	$\sigma_{50}, \sigma_{50}, -$	7	7	0.04	0.26	0.03	0.72
60	$\sigma_{50}, \sigma_{50}, -$	3	3	0.67	0.37	0.26	0.82
60	$\sigma_{50}, \sigma_{50}, -$	7	5	0.47	0.23	0.19	0.38
100	$\sigma_{50}, \sigma_{50}, -$	3	1	2.77	0.24	1.07	0.45
100	$\sigma_{50}, \sigma_{50}, -$	7	3	2.24	0.36	0.75	0.65
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	9	1.33	0.22	0.60	0.21
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	9	1.26	0.02	0.60	0.09
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	6	30.27	0.16	5.27	0.22
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	7	27.73	0.12	5.24	0.30
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	5	limit	0.23	17.66	0.12
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	8	limit	0.14	17.41	0.07

Also, as noted in Section 6.2.1, the DP state space increases of one dimension for every size of the jobs, and the magnitude of these state space dimensions are exactly the maximum batch capacities b_i for every $i \in \{1, \dots, d\}$. Thus, having a larger batch capacity leads to a considerably higher memory usage.

With respect to the percentage gaps, the algorithms still perform well, with very good average gaps, showing that the quality is not affected by the granularity of job sizes.

6.4 Final remarks

The parallel batch scheduling problem has become more and more addressed by the scientific and industrial communities because of its applications in many industrial fields. Specifically, the parallel batch scheduling problem with multiple size constraints has been starting to get attention for its applicability to the growing Additive Manufacturing technology, where parts are placed in chambers in 2- or 3-dimensional spaces. Also, including job families with incompatibilities between families in the batch composition has getting attention, as jobs of different families often need different operations in shop floors.

This work addresses the $1|p\text{-batch}, b_i, \sigma_{ij}, \text{incomp}| \sum C_j$ problem; to the writer's knowledge, this is the first attempt in the literature to consider multiple sizes and family incompatibility constraints together. Also, no assumptions are made on the distribution and/or the value of the processing times.

The solution approach is based on the flow formulation of the problem described in Chapter 4 (Alfieri et al. [3]) and with the heuristic improvements described in Chapter 5 (Druetto and Grosso [20]); this formulation is, in fact, exploited to develop two heuristics based upon CG techniques: one is the P&B heuristic CG-UB, the other is the variable rounding procedure VR-UB. The CG finds a continuous-relaxed solution, then the two heuristics are used to move from the continuous to the integer solution of the problem. An extensive experimental campaign compared the two heuristics, which both proved to be very effective for this scheduling problem. Indeed, the proposed approaches can handle instances up to 200 jobs, and both find very good optimality gaps in all the addressed instances. Moreover, with a small number of jobs, the proposed algorithms are able to find optimal solutions in most of the cases.

Numerical results show that the smaller the numerosity of job sizes, the more difficult the batch scheduling problem becomes. Having more than one size constraint simplifies the problem from the computation standpoint; also, having more families simplifies the problem, as the number of feasible combinations of jobs is reduced.

Comparing the two heuristics, interesting results emerged. In simple instances (that is, with a small number of jobs, large numerosity of job sizes and a small number of families) the difference between the two approaches is not appreciable. The real gain can be perceived in difficult instances, where both computation times and gaps largely decrease. Specifically, instances with 200 jobs can not be solved by the

CG-UB approach in 100 second time limit; however, the variable rounding VR-UB is able to achieve good gaps in less than the time limit. In general, almost all instances are solved within a minute and the gap rarely overcomes 5%. The variable rounding procedure is therefore shown to be a valid alternative to the other approach.

At its current state, the proposed approach is able to solve single-machine batch scheduling problems. Further research will be devoted to adapting the approach to parallel machines (as for the $Pm|p\text{-batch}, b, \sigma_j| \sum C_j$ problem described in Section 4.2) and to weighted completion times (as for the $1|p\text{-batch}, b, \sigma_j| \sum w_j C_j$ problem described in Section 5.1).

As an alternative significant and nontrivial direction for future research, one might consider identifying *families* with *agents*. The main difference with this model is that in a multiagent setting each agent pursues its own objective; that is, the minimization of the total completion time of its jobs only. To the writer's knowledge, this problem has not yet been addressed in the context of bounded $p\text{-batch}$ scheduling problems and multi-size jobs, while a pseudopolynomial algorithm exists for the unbounded case and generic sum-type objectives (Li and Yuan [43]). It would be interesting to analyze whether the approach proposed in this work can be modified to the multiagent setting.

Unweighted total completion time: analysis of two polynomial-size models

IN recently proposed, so-called arc-flow models, the structure of a batch sequence is encoded as a flow configuration on a suitable type network. An original arc-flow Mixed-Integer Program (MIP) model is proposed by Trindade et al. [67] for makespan minimization, leading to excellent computational results on very large instances.

In Chapter 4 (Alfieri et al. [3]) a different arc-flow model is proposed for the $1|p\text{-batch}, b, \sigma_j|\sum C_j$ problem, leading to exact solution of 40-jobs instances and heuristic solution of 100-jobs instances. The size of such model is exponential in the number of jobs, and Column Generation (CG) techniques are necessary to solve it. In Chapter 5 (Druetto and Grosso [20]) this approach is extended to the weighted case with the introduction of bounding heuristics, and in Chapter 6 (Druetto et al. [22]) is further extended to cope with multi-size and incompatible families variant.

From a practical point of view, it can be appealing to get rid of the technicalities that arise in CG. We present the first strong MIP model for problem $1|p\text{-batch}, b, \sigma_j|\sum C_j$ which is also “compact” in size, that is, with a number of variables and constraints bounded by a polynomial in the number of jobs.

The continuous relaxation of such model delivers a sharp lower bound, competitive with the bounds provided by the CG techniques described in Chapter 4. Combined with a variable rounding heuristic, inspired by the one implemented in Chapter 5, it can generate very good solutions with certified optimality gaps for instances with up to 50 jobs. Although this performance is still behind that of CG, the compact model is extremely promising.

7.1 Models description

Recalling the notation introduced in Chapter 3, a set of jobs $N = \{1, 2, \dots, n\}$ is to be partitioned into batches and processed on a single machine. Each job $j \in N$ has a given processing time p_j and a size s_j ; jobs are partitioned and processed without interruptions in a batch sequence $S = (B_1, B_2, \dots, B_t)$, and each batch B_k in S must satisfy $\sum_{j \in B} s_j \leq b$, where b is the machine capacity. The jobs in a same batch B_k are processed simultaneously, with the longest job determining the processing time

for the whole batch, having that $p_B = \max\{p_j : j \in B\}$. All the jobs in the same batch B_k share the same completion time, that is $C_j = C_{B_k} = \sum_{l=1}^k p_{B_l}$.

The considered problem calls for finding S that minimizes $f(S) = \sum_{j \in N} C_j$.

7.1.1 Arc-flow models

Arc-flow models for parallel batching problems are a recent development, where batch schedules are mapped on flow configurations on a suitable network.

In Trindade et al. [67] an arc-flow model for minimizing makespan in a single parallel-batching machine is proposed, with very good results. In Alfieri et al. [3] an arc-flow model is proposed for the $1|p\text{-batch}, b, \sigma_j|\sum C_j$ problem, leading to handle 100-jobs instances. This arc-flow model is exponential in size and must be handled via CG techniques; here we develop, from the same modeling ideas, arc-flow models whose size is polynomial in the number on jobs n .

For the key modeling ideas on arc-flow representations of $1|p\text{-batch}, b, \sigma_j|\sum C_j$, refer to Section 4.1.1. In particular, equation 4.11 defines the $G(V, A)$ graph, Property 1 shows that the structure of a batch sequence S can be mapped onto a path P_S from node 1 to node $n + 1$ in the graph G , and in Fig. 4.1 conveys the idea behind the proof.

7.1.2 A polynomial-size flow-based model

In this section, we formulate a MIP that calls for finding a minimum-cost path P_S from node 1 to $n + 1$ on graph G , partitioning jobs on the arcs of P_S .

Each decision variable x_{ik} will be set to 1 if arc (i, k) belongs to P_S ; constraints (7.2) are flow conservation constraints, with a unit flow routed from node 1 to node $n + 1$; we note that this is the classical MIP formulation of a path construction problem. Each decision variable $y_{ikj} = 1$ if job j is assigned to the batch B_{ik} corresponding to arc (i, k) . Constraints (7.3) require that each job is assigned to an arc/batch, constraints (7.4) limits the total size packed into a batch, and constraints (7.5) enforce the correct cardinality of a batch. Variable π_{ik} is set by constraints (7.6) to the processing time of the longest job in the batch.

The total completion time is expressed by (7.1), multiplying processing times π_{ik} for the number of still-to-be-scheduled jobs *in advance*, since completion time of each job j is the processing time of the batch it is currently scheduled in, plus the processing time of all previous batches.

$$\text{minimize } \sum_{(i,k) \in A} (n - i + 1) \pi_{ik} \quad (7.1)$$

$$\text{subject to } \sum_{(i,k) \in A} x_{ik} - \sum_{(k,i) \in A} x_{ki} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n + 1 \end{cases} \quad (7.2)$$

$$\sum_{(i,k) \in A} y_{ikj} = 1 \quad j \in N \quad (7.3)$$

$$\sum_{j=1}^n s_j y_{ikj} \leq b x_{ik} \quad (i, k) \in A \quad (7.4)$$

$$\sum_{j=1}^n y_{ikj} = (k - i) x_{ik} \quad (i, k) \in A \quad (7.5)$$

$$\pi_{ik} \geq p_j y_{ikj} \quad j \in N, (i, k) \in A \quad (7.6)$$

$$\pi_{ik} \geq 0 \quad (i, k) \in A \quad (7.7)$$

$$x_{ikt}, y_{jikt} \in \{0, 1\} \quad j \in N, (i, k) \in A \quad (7.8)$$

7.1.3 A stronger model

Model (7.1)–(7.8) requires $\mathcal{O}(n^3)$ variables and constraints. It is a reasonably compact MIP, but computational experience shows that it still has severe limitations; although the lower bound delivered by the continuous relaxation of (7.1)–(7.8) is much higher than the one computed on (4.1)–(4.9), the optimality gap is still large, and CPLEX cannot solve (7.1)–(7.8) on large instances.

A stronger arc-flow model can be developed at the cost of using a larger (but still polynomial) number of variables. Let $\mathcal{T} = \{p_j : j \in N\}$ the set of all *distinct* processing times listed in the considered problem instance. The MIP model still calls for finding a minimum cost path P_S from node 1 to node $n + 1$; we use a larger set of decision variables x_{ikt} , where $x_{ikt} = 1$ iff an arc (i, k) is in the paths and the corresponding batch B_{ik} has processing time $p_{B_{ik}} = t \in \mathcal{T}$. Another set of decision variables y_{jikt} is defined, with $y_{jikt} = 1$ iff job j is in the batch B_{ik} and the latter has batch processing time $p_{B_{ik}} = t$.

The cost c_{ikt} for an arc whose batch B_{ik} has processing time t is defined as

$$c_{ikt} = t(n - i + 1)$$

and the complete model can then be written as follows.

$$\text{minimize } \sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} c_{ikt} x_{ikt} \quad (7.9)$$

$$\text{subject to } \sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} x_{ikt} - \sum_{\substack{(k,i) \in A \\ t \in \mathcal{T}}} x_{kit} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n + 1 \end{cases} \quad (7.10)$$

$$\sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} y_{jikt} = 1 \quad j \in N \quad (7.11)$$

$$\sum_{j \in N} s_j y_{jikt} \leq b \cdot x_{ikt} \quad \begin{matrix} (i,k) \in A \\ t \in \mathcal{T} \end{matrix} \quad (7.12)$$

$$\sum_{j \in N} y_{jikt} \leq (k - i) x_{ikt} \quad \begin{matrix} (i,k) \in A \\ t \in \mathcal{T} \end{matrix} \quad (7.13)$$

$$y_{jikt} \leq x_{ikt} \quad \begin{matrix} j \in N \\ (i,k) \in A \\ t \in \mathcal{T} \end{matrix} \quad (7.14)$$

$$x_{ikt}, y_{jikt} \in \{0, 1\} \quad \begin{matrix} j \in N \\ (i,k) \in A \\ t \in \mathcal{T} \end{matrix} \quad (7.15)$$

Objective function (7.9) requires to minimize the sum of total completion time for all selected arcs. Constraints in (7.10) represents a classical flow model with unitary flow, from source 1 to destination $n + 1$, counting as positive the flow to outgoing arcs and as negative the flow from incoming arcs. Exact partitioning for all jobs is enforced in (7.11), and capacity constraints (7.12) are defined for all selected arcs. Correct cardinality of a batch is enforced by constraints (7.13).

The number of variables for this model grows theoretically to $\mathcal{O}(n^4)$, but the actual number of variables can be sensibly trimmed since not all pairs i, k (with $i < k$) can correspond to a batch because of the machine capacity limit. Also, constraints (7.14) can be used as *lazy constraints*, to be dynamically separated when needed.

7.2 Variable Rounding heuristic

Once programs (7.1)–(7.8) or (7.9)–(7.15) has been solved to their continuous optimum, a lower bound is available. A simple strategy for getting an integral upper bound could be setting all variables back to the binary type and solve the integer version of the problem by Branch-and-Bound (B&B), truncating the process when

a limit on computation time is reached. The value of solution obtained with this approach will be called Branch-and Bound Upper Bound (BB-UB) in the following.

We also investigated a simple strategy based on rounding fractional variables in order to generate feasible solutions within shorter computation times. The value of such solution is called Variable Rounding Upper Bound (VR-UB) in the following. This approach is inspired by Mourgaya and Vanderbeck [53] where rounding heuristics are applied to vehicle routing problems, and is derived from the work described in Chapter 5 (Druetto and Grosso [20]) and Chapter 6 (Druetto et al. [22]).

7.2.1 Variable Rounding for the 7.1.3 model

Using a basic rounding approach, we build a partition path by rounding flow values, starting from the source and moving towards the sink node. The full procedure can be described as follows.

- Given the optimal fractional flow on the problem, we search for tuple (i, k, t) corresponding to arc (i, k) with processing time t and maximal nonzero flow that lies as near as possible to the source, and round its flow value to 1. This corresponds to the fixing of a variable x_{ikt} to 1.
- We search for all tuples (j, i, k, t) corresponding to jobs j associated to the fixed arc (i, k) with processing time t that present nonzero flow; potentially those jobs can be *more* than the required $k - i$ jobs to fill the arc.
- Amongst all subsets of the jobs found in the previous step, we select the *best* combination that can fit in a batch, that is of length $k - i$, and with *maximum flow value* amongst the sum of flow values for all included jobs; those jobs will have their flow value rounded to 1. This corresponds to the fixing of some variables y_{jikt} to 1. This step is admittedly combinatorial, but never involves more than a handful of jobs.
- The problem is optimized again to obtain new flow values for the remaining arcs and jobs.

We then iterate this rounding step until an arc reaching the sink node $n + 1$ is fixed, thus completing a feasible solution.

7.2.2 Variable Rounding for the 7.1.2 model

The underlying approach is similar to the one implemented for the other model: we build a partition path by rounding flow values, starting from the source and moving towards the sink node. Difference between this procedure and the one described in 7.2.1 model are as follows.

Since variables relative to arcs and jobs does not depend on a specific processing time, we have to fix the variable x_{ik} that corresponds to the arc (i, k) with maximal nonzero flow to 1, search for its associated jobs j with nonzero flow, select some of them with the same criteria as before, and fix those variables y_{ikj} to 1.

7.3 Computational results

For $1|p\text{-batch}, b, \sigma_j| \sum C_j$ very few heuristics and/or relaxations are available in literature; among them are both a relaxation and a greedy heuristic (Uzsoy [69], Azizoglu and Webster [5]) in the context of a B&B algorithm. Such procedures are labeled AW-LB and AW-UB respectively.

The following naming convention is valid for all tables, for the results of both models. All gaps in the following tables are evaluated by the relative difference between upper bound and lower bound, using the formula

$$\text{Gap} = \frac{\text{UB} - \text{LB}}{\text{UB}}.$$

Gap values for the AW-UB procedure by Azizoglu and Webster [5] are evaluated using AW-LB as the lower bound.

The values for the lower bound obtained by solving the continuous relaxation of the problem are denoted by CR-LB, and with BB-UB we refer to the upper bound found by running the integer version of the problem. The gap values in this case are evaluated using the best continuous lower bound returned by the solver after its termination (optimum found, or computation time limit reached). Time limit for the solver is set to 600 seconds.

The values for the upper bound obtained by Variable Rounding, as described in Section 7.2, are denoted by VR-UB. The gap values in this case are evaluated using CR-LB as the lower bound.

7.3.1 Testing environment

All the tests ran in a Linux environment equipped with Intel Core i7-6500U CPU @ 2.50GHz processor; all algorithms (AW-LB, AW-UB, VR-UB) have been implemented in Python 3.6; the Linear Program (LP) and MIP solver used was CPLEX 12.8, called directly from Python 3.6 environment using the callable library.

A number of test instances were generated following the de-facto standard for this type of parallel batching problems.

- The machine capacity was fixed to $b = 10$.

- The processing times p_j were randomly drawn from the uniform discrete distribution $[1, 100]$.
- The job sizes s_j were randomly drawn from four possible uniform discrete distributions, labeled by $\sigma \in \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

$$\begin{array}{ll} \sigma_1 : s_j \in [1, 10] & \sigma_3 : s_j \in [3, 10] \\ \sigma_2 : s_j \in [2, 8] & \sigma_4 : s_j \in [1, 5]. \end{array}$$

A batch with 10 instances for each σ class was generated, considering an increasing number of jobs $n \in [10, 20, 30, 40, 50]$. The following tables report average values evaluated over all (n, σ) combinations. The diciture `limit` is displayed for the cases where computational times of all instances in a particular combination exceeded the maximum allotted time of 600 seconds.

7.3.2 Results for the 7.1.2 model

Tab. 7.1 contains the average number of nodes opened during evaluation of BB-UB by the solver, and the number of optima found, all within the 600 seconds threshold.

Tab. 7.1: Number of opened BnB nodes and optima found for the 7.1.2 model.

Param		Avg Nodes	# Opt
n	σ	BB-UB	BB-UB
10	σ_1	7472	10
	σ_2	22256	10
	σ_3	3892	10
	σ_4	5440	10
20	σ_1	148518	0
	σ_2	613856	0
	σ_3	930158	0
	σ_4	102653	0

Unfortunately, performance of this model is poor: the B&B requires to explore a lot of nodes, and even with only 20 jobs the solver cannot find even an optimal result for all instances.

Comparison between upper bounds, in Tab. 7.2, confirms the poor performance of this model. Although the lower bound CR-LB is very fast to execute, even with 20 jobs where in less than one second the result is found, the integer optimum BB-UB is very hard to find. After the entire allotted time limit of 600 seconds, the gap obtained by the solver is, in the majority of the cases, even worse than the gap obtained using the polynomial bounds by by Azizoglu and Webster [5].

Since the quality of the lower bound CR-LB is bad, even the Variable Rounding upper bound VR-UB is of poor quality. Here, even for 10 jobs the obtained gap is worse than the gap obtained with the aforementioned polynomial bounds.

Tab. 7.2: Comparison of bounds quality and execution times for the 7.1.2 model.

Param		Avg Times (s)			Avg Gap		
n	σ	CR-LB	BB-UB	VR-UB	AW-UB	BB-UB	VR-UB
10	σ_1	0.02	3.27	0.02	0.37	0.00	0.67
	σ_2	0.01	3.14	0.01	0.26	0.00	0.48
	σ_3	0.01	0.72	0.01	0.22	0.00	0.36
	σ_4	0.03	3.60	0.02	0.37	0.00	0.70
20	σ_1	0.30	limit	0.61	0.33	0.46	0.70
	σ_2	0.20	limit	0.19	0.26	0.32	0.59
	σ_3	0.18	limit	0.11	0.21	0.16	0.44
	σ_4	0.31	limit	1.07	0.37	0.57	0.81

We decided to test this model only for instances with 10 and 20 jobs, given these poor performances.

7.3.3 Results for the 7.1.3 model

Tab. 7.3 contains the average number of nodes opened during evaluation of BB-UB by the solver, and the number of optima found, all within the 600 seconds threshold.

Tab. 7.3: Number of opened BnB nodes and optima found for the 7.1.3 model.

Param		Avg Nodes	# Opt
n	σ	BB-UB	BB-UB
10	σ_1	0	10
	σ_2	0	10
	σ_3	1	10
	σ_4	0	10
20	σ_1	753	10
	σ_2	1036	10
	σ_3	783	10
	σ_4	442	10
30	σ_1	648	7
	σ_2	6071	7
	σ_3	1846	10
	σ_4	143	2
40	σ_1	188	1
	σ_2	138	1
	σ_3	5189	3
	σ_4	2	0
50	σ_1	136	1
	σ_2	642	1
	σ_3	4752	4
	σ_4	0	0

The performance of this model is way better than the previous: the B&B requires to explore a very small number of nodes. For 10 jobs, in fact, the entire problem is solved in the root node, in the majority of the cases.

However, since the model is heavier than the previous one, even the computation of the root node is intensive. As we can see for 50 jobs, for the hardest distribution σ_4 the solver uses the entirety of its 600 seconds allotted time only for the root node evaluation.

Tab. 7.4: Comparison of bounds quality and execution times for the 7.1.3 model.

Param		Avg Times (s)			Avg Gap		
n	σ	CR-LB	BB-UB	VR-UB	AW-UB	BB-UB	VR-UB
10	σ_1	0.04	0.12	0.02	0.37	0.00	0.04
	σ_2	0.02	0.12	0.01	0.26	0.00	0.05
	σ_3	0.01	0.07	0.01	0.22	0.00	0.04
	σ_4	0.06	0.14	0.01	0.37	0.00	0.02
20	σ_1	0.93	41.26	1.04	0.33	0.00	0.09
	σ_2	0.57	14.03	0.57	0.26	0.00	0.07
	σ_3	0.33	4.38	0.29	0.21	0.00	0.04
	σ_4	1.60	100.97	1.51	0.37	0.00	0.04
30	σ_1	4.94	238.89	8.44	0.32	0.01	0.08
	σ_2	2.71	231.89	3.92	0.28	0.00	0.06
	σ_3	1.58	19.91	1.71	0.22	0.00	0.03
	σ_4	9.03	516.16	13.43	0.34	0.05	0.06
40	σ_1	24.64	576.59	55.11	0.37	0.06	0.08
	σ_2	16.58	567.71	48.09	0.27	0.05	0.09
	σ_3	8.12	531.04	14.13	0.24	0.01	0.07
	σ_4	43.00	limit	81.08	0.29	0.11	0.05
50	σ_1	63.40	583.14	123.19	0.35	0.09	0.07
	σ_2	26.33	591.03	37.28	0.24	0.01	0.07
	σ_3	15.49	367.34	11.06	0.19	0.00	0.03
	σ_4	74.65	limit	213.89	0.27	0.16	0.05

Comparison between upper bounds, in Tab. 7.4, shows the interesting performance of this model. The lower bound CR-LB is slightly slower to execute than in the previous case, but in return we have a way better Variable Rounding upper bound VR-UB in all cases, having a gap lower than 0.10 and performing way better than the gap evaluated with the polynomial bounds by Azizoglu and Webster [5]. When the upper bound BB-UB starts breaking the 600 seconds threshold in its evaluation, for 40 and 50 jobs, the gap evaluated by VR-UB is very competitive in comparison, even surpassing the B&B for distribution σ_4 that is known to be hard.

It is worth noting that the time required for the Variable Rounding upper bound is way lower than the B&B one, with an order of magnitude very similar to the continuous lower bound.

Increasing the number of jobs for the instances, it is clear that VR-UB will be even better in time performance than BB-UB, and will always retain its good gap quality.

7.4 Final remarks

Our analysis for the parallel batching problem $1|p\text{-batch}, b, \sigma_j| \sum C_j$ over two arc-flow models shows that model (7.9)–(7.15) is capable to deliver a good continuous relaxation lower bound CR-LB and a good Variable Rounding upper bound VR-UB, leading to excellent performance in gap quality within an acceptable amount of time.

Tab. 7.5: Comparison between lower bound CR-LB obtained by the 7.1.3 model, and lower bound CG-LB obtained by the CG approach described in Section 4.1.2.

Param		Avg Ratio
n	σ	$\frac{\text{CR-LB}}{\text{CG-LB}}$
10	σ_1	0.98
	σ_2	0.96
	σ_3	0.97
	σ_4	0.99
20	σ_1	0.95
	σ_2	0.96
	σ_3	0.98
	σ_4	0.99
30	σ_1	0.96
	σ_2	0.97
	σ_3	0.98
	σ_4	0.99
40	σ_1	0.97
	σ_2	0.96
	σ_3	0.97
	σ_4	0.99
50	σ_1	0.97
	σ_2	0.97
	σ_3	0.99
	σ_4	0.99

In Tab. 7.5 we compared the lower bound CR-LB obtained with the best performing model of Section 7.1.3, with the current state-of-the-art lower bound known in literature; that is, the Column Generation Lower Bound (CG-LB) described in Section 4.1.2 (Alfieri et al. [3]). The comparison ratio is evaluated by dividing the two lower bounds,

$$\text{Ratio} = \frac{\text{CR-LB}}{\text{CG-LB}};$$

the higher this fraction is, the better model (7.9)–(7.15) performs.

As it can be seen, the quality of our lower bound CR-LB is excellent, and stays tightly close to the state-of-the-art lower bound delivered by CG; especially for the hardest distribution σ_4 .

Thus we claim that model (7.9)–(7.15) is promising and deserves further study, in order to improve its performances; in particular we are studying *sifting* techniques for speeding up the solution of the continuous relaxation, reducing as well the computational time required by the Variable Rounding heuristic. Also, the extension to multiple parallel machines (identical or unrelated) variant of the problem would be easily handled. Extension of this approach also to the *weighted* total completion time case is under investigation.

Part II

Process scheduling in embedded systems

Process scheduling and memory mapping: multi-step optimization approach



THE second part of my PhD was dedicated to the development of ad-hoc assignment and scheduling algorithms for specific problems that arise in automotive embedded systems, in particular in the presence of shared variables constraints.

Multicore architectures provide the increased performance required by modern embedded real-time systems. Most platforms exhibit a Non-Uniform Memory Access (NUMA). In NUMA, memory banks with different access time can be explicitly addressed. Such an architecture, however, is challenging predictability given the significant impact of the allocation of variables on the execution times.

At software level, real-world embedded applications (for example, automotive) are composed by thousands of functions often communicating through shared variables stored in memory, with a variable access time because of NUMA.

This chapter addresses the mapping of complex embedded applications onto NUMA multicore architectures. The developed problem formulation offers a solution to the following problems:

1. allocating variables (called *labels* in the automotive context) over memories of different characteristics;
2. mapping functionalities (called *runnables*) onto Central Processing Units (CPUs);
3. creating Operating System (OS) tasks from runnables;
4. assigning priorities to tasks.

Our developed implementation is capable to handle an application composed by 1K+ runnables, all sharing 10K+ labels and finds a solution in at most 3 minutes on a standard laptop, enabling interactive design space exploration.

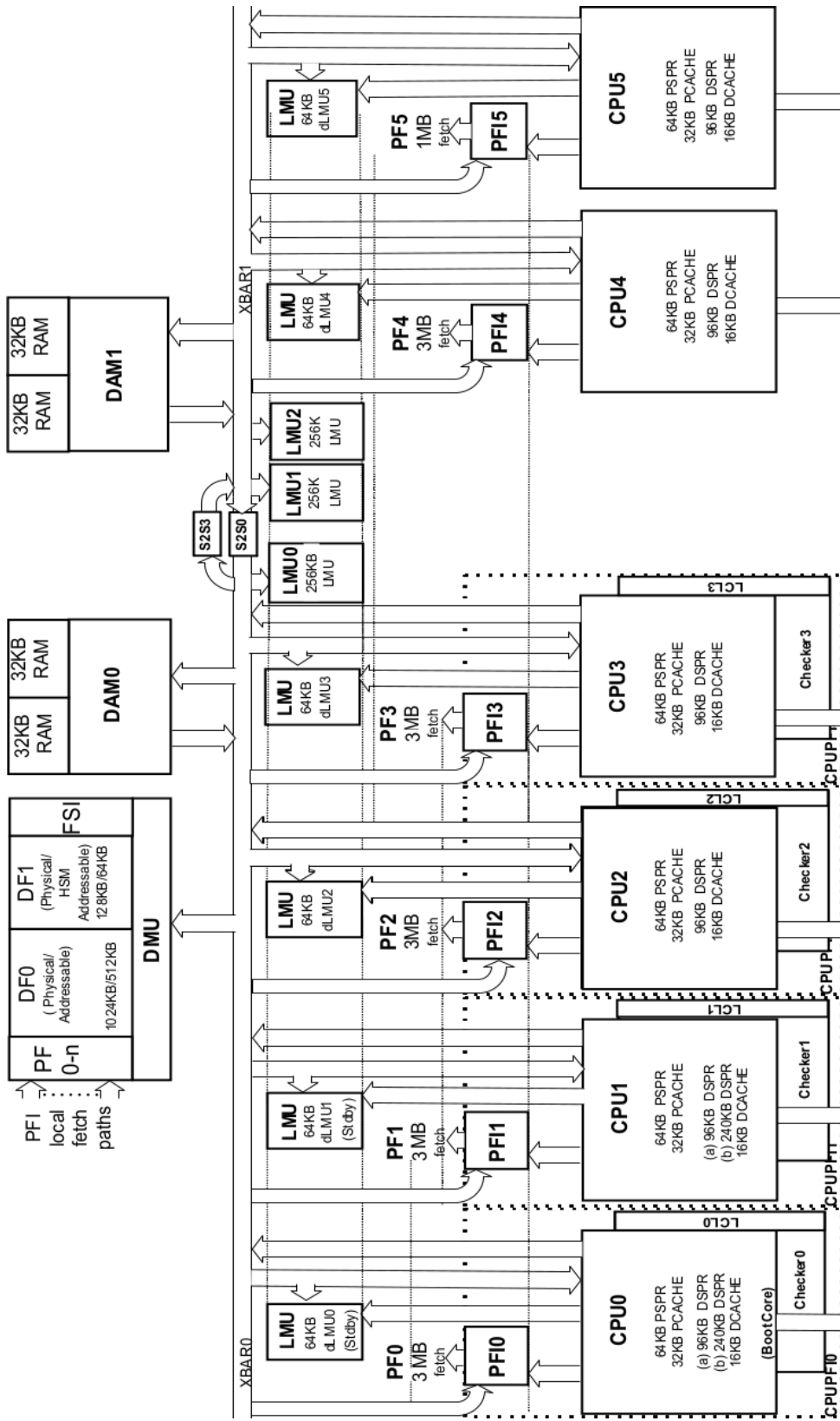


Fig. 8.1: AURIX TriCore TC39x: CPUs and memories (no I/O is reported).

8.1 CPUs and memories in NUMA

In this section, we illustrate a realistic NUMA architecture, widely used for large embedded applications, especially in the automotive context. Later, in Section 8.2.1, we will be borrowing an abstract model of the hardware (Wolff et al. [130], *Eclipse APP4MC Web Page* [88]), which generalizes the architectural features shared by several architectures.

Our reference architecture is Infineon AURIX™ TriCore™ TC3xx (*AURIX TriCore TC3xx Web Page* [75]). Fig. 8.1 describes the architecture of CPUs, memories, and the corresponding interconnections.

Specifically, the TC39x architecture offers the following memory areas for data, listed by decreasing proximity to the CPUs.

- DCACHE is the two-way set associative caches with Least Recently Used (LRU) replacement algorithm. It is protected by Error Correction Codes (ECCs) and, if properly configured, may be accessed from other CPUs.
- Each core has a dedicated Data ScratchPad RAM (DSPR) with fast access.
- Blocks of Local Memory Units (LMUs) are available to offer ECC-protected volatile storage.
 - Also, each core has a block of Distributed Local Memory Unit (dLMU), which grants a fast access to the CPU it is directly connected to. Other CPUs may also access through the available interconnection.
- Default Application Memory (DAM) may be used for both code or data, but has no hardware safety mechanisms such as ECCs.

The architecture also has memory areas to store code (PCACHE, PSPR, PF). These memories, however, are not further described because program storage is normally not a stringent constraint.

The above listed memories and the CPUs are connected through the high bandwidth System Resource Interconnect (SRI) Fabric. Accesses by CPUs to the available memory blocks are regulated by two crossbars (indicated by *XBAR0* and *XBAR1* in Fig. 8.1), as follows.

- Accesses to different memory blocks are executed in parallel.
- Accesses to the same memory block are arbitrated according to two round-robin groups (high and low priority), depending on the configuration. By setting all requestors to the same priority, standard round robin is achieved with a constant access time.

- Accesses from a CPU over one crossbar to a memory block connected to the other crossbar need to go through bridges ($S2S3$ and $S2S0$ in Fig. 8.1), and take a longer time.

Depending on the size of the data, more than one SRI bus transaction may be needed to complete a single load/store instruction.

8.2 System model

In this section, we describe the abstract model we referred to, in the analysis and development of our approach. This abstraction generalizes the relevant and real features shared by several architectures.

AUTomotive Open System ARchitecture (AUTOSAR) (*AUTOSAR Web Page* [76]) is a standardized software architecture for the definition of automotive components and for providing the foundation platform for their execution (Fürst et al. [93]). The essential element of AUTOSAR that is relevant for our problem is the concept of *runnables*, that are functions to be executed in response to events. For the sake of our work, we are interested in capturing the nature of these activation events, assumed as periodic or sporadic. AUTOSAR runnables communicate by means of data ports or by client-server interactions. In both cases, memory locations shall be identified (in a stage called Run-Time Environment (RTE) generation) to store the values communicated over the ports or the arguments of the call and its result.

In the AMALTHEA Projects (Model Based Open Source Development Environment for Automotive Multi Core Systems) and its follow-up APP4MC (*Eclipse APP4MC Web Page* [88]), the problem is abstracted by analyzing the results of the RTE generation stage directly. The model contains the elements that provide the minimal level of details necessary to setup a partitioning problem (Wolff et al. [130]). The AMALTHEA/APP4MC metamodel addresses also the hardware and the access time pertaining to it. The APP4MC platform was used to introduce the model of the 2017 WATERS Challenge (Kramer et al. [101]), which is used as our reference use case.

8.2.1 Hardware model

The AMALTHEA/APP4MC hardware model is depicted in Fig. 8.2. It provides the key features with an impact on timing analysis, without delving into too fine-grained details, which have little impact on the mapping problem. According to this view:

- a set of the identical CPUs, denoted here by \mathcal{M} , is available for processing instructions;
- each CPU $k \in \mathcal{M}$ has a directly connected *local memory* called Local RAM (LRAM) of size S_k^{CPU} , which can be accessed at higher speed;

- a *generic memory* called Global RAM (GRAM) is available (we assume its size is large enough to accommodate data as needed);
- a crossbar switch enables all CPUs to access both the GRAM and the LRAM of the others with dedicated virtual channels;
- all LRAMs and the GRAM are mapped to a unique address space, making them accessible from any CPU.

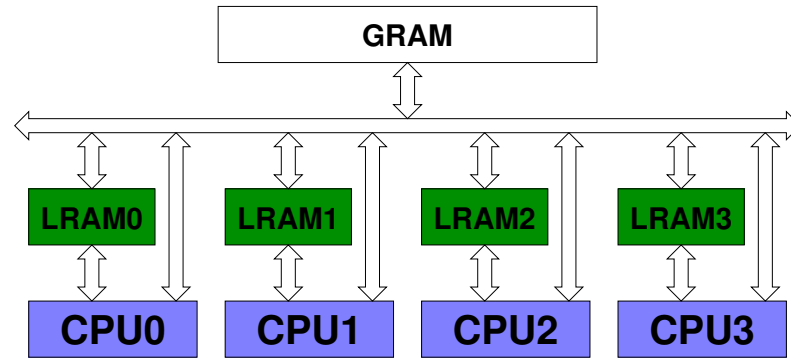


Fig. 8.2: Abstract hardware model.

8.2.2 Software model

The software model used in AMALTHEA, which is also relevant to our purposes, includes the following terms.

- A *label* is a data element used by the application code; it has a type, which determines its memory size. The set of labels is denoted by \mathcal{L} and s_ℓ is the size of label $\ell \in \mathcal{L}$.
- A *runnable* is a function implemented by sequential code. The set of runnables is denoted by \mathcal{N} . A runnable may read or write labels with a given frequency. Communication between runnables is implemented by writing/reading shared labels. As illustrated later in Section 8.4, a subset of labels $\mathcal{L}_i \subseteq \mathcal{L}$ is attached to the runnable $i \in \mathcal{N}$.
- T_i denotes the *period* or *minimum interarrival time* of runnable i . Later, we may use the notation $f_i = \frac{1}{T_i}$ to denote the *maximum frequency* of activation.
- A *task* corresponds to the OS notion of thread. Its code corresponds to a sequence of runnables invoked sequentially. The set of tasks is denoted by \mathcal{T} .
- Other notions will be used in this chapter such as the *gain* $g_{i,\ell}$ for runnable i to have fast access to label ℓ or the *execution cycles* C_i^0 of runnable i . However, we postpone their precise definitions to the context when they will be needed (Section 8.4 and Section 8.5, respectively).

Tab. 8.1 reports the size of the 2017 WATERS Challenge (Kramer et al. [101]) embedded application, which we extensively use in this paper.

Tab. 8.1: Key data of 2017 WATERS Challenge reference application.

property	value
Number of labels in \mathcal{L}	10000
Total memory of labels [bytes]	27363
Number of runnables in \mathcal{N}	1250
Number of tasks in \mathcal{T}	21
Number of different release patterns	19
Number of accesses by runnables to any label	15255

Finally, tasks execute on statically assigned CPU (partitioned scheduling) and are scheduled by Fixed Priority.

8.3 Problem description

The problem addressed in this work is the mapping of an application modeled by runnables \mathcal{N} and labels \mathcal{L} over the CPUs \mathcal{M} and their associated LRAMs, respectively (as depicted in Fig. 8.4).

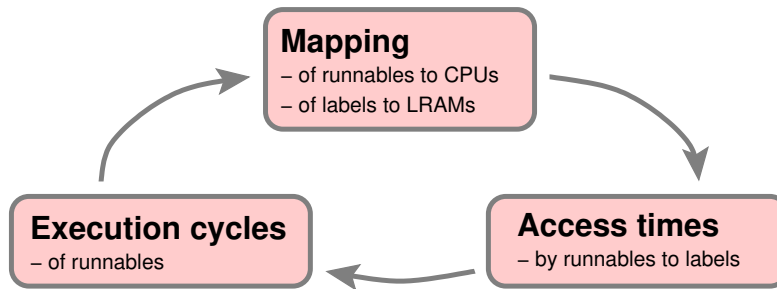


Fig. 8.3: The cyclic dependency of the mapping problem.

The main difficulty of the problem is due to the accesses that runnables make to labels. Depending on the mapping, these accesses may happen either locally (from a CPU to its directly connected LRAM, such as LRAM2 from CPU2 in Fig. 8.2) or remotely. Since access times vary by one order of magnitude (please refer to Tab. 8.2), they do affect the overall execution cycles of runnables.

In turn, the execution cycles of runnables do affect the mapping as any knapsack problem is affected by the size of the items to be packed. Such a cyclic dependency is represented in Fig. 8.3.

Finally, the size of real-world problem (please refer to Tab. 8.1) is about two orders of magnitude above the size of tractable problems of this kind.

Hence, we decompose the mapping in the following stages.

1. First, we address the problem of binding labels to runnables (in Section 8.4). This is the key enabler of the significant reduction in complexity. In Fig. 8.6, labels bound to a runnable are represented by a thick link.
2. Then runnables are mapped to CPUs (Section 8.5). As illustrated in Fig. 8.5, every runnable carries the bound labels which are then implicitly mapped to the corresponding LRAM.
3. Finally, runnables are assigned to tasks (Section 8.6) and then tasks are assigned a priority (Section 8.7).

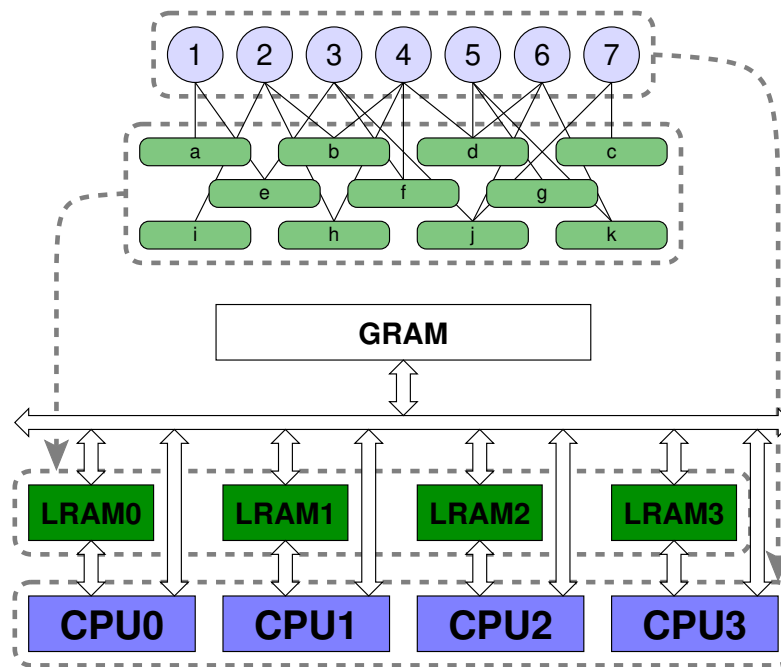


Fig. 8.4: The mapping problem. Runnables are azure circles (“1”, “2”, ...), labels are mint green rounded boxes (“a”, “b”, ...). The mapping (of runnables to CPUs and labels to LRAMs) is represented by thick dashed gray arrows.

As it will be illustrated in greater details in the next section, the guiding principles that drive all optimization stages are the **minimization of the resource utilization**, and the **maximization of the slack** so that further upgrades or extensions may be accommodated more easily.

Without loss of generality, we identify the elements in any set \mathcal{N} by the integers $1, \dots, |\mathcal{N}|$. Also, to lighten the presentation, we use the same notation of any set \mathcal{N} to denote the number of elements as well. In short, we consider correct to write $\mathcal{N} = \{1, 2, \dots, \mathcal{N}\}$.

8.4 Binding labels to runnables

The key phase that makes the overall methodology feasible for the large scale automotive use case, is the binding of labels to runnables. In fact, given the size of realistic applications (of 1K+ runnables and 10K+ labels, please refer to Tab. 8.1 for details), a unique Integer Program (IP) formulation for the joint mapping of the labels and runnables is not tractable with the computing capacity available at time of writing.

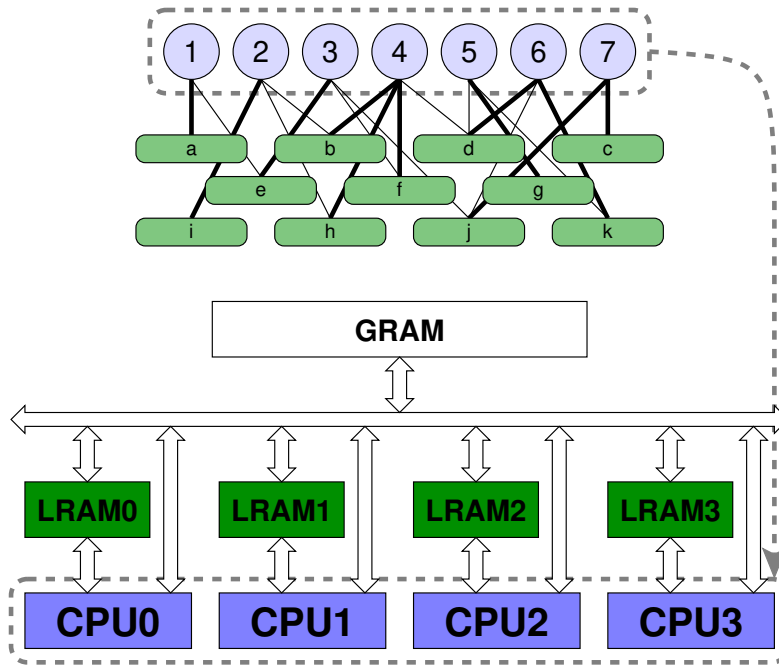


Fig. 8.5: In our methodology, labels are first bound to the runnable which benefits the most (represented by a thick link), then runnables only are mapped to a CPU; the bound labels will follow to the linked LRAM.

Hence, we bind labels to runnables (as represented by thick black links between runnables and labels in Fig. 8.6). When a runnable i is mapped to CPU k , then all the labels \mathcal{L}_i bound to it are mapped to the LRAM directly linked to CPU k .

We formulate the binding problem as follows. For each pair $(i, \ell) \in \mathcal{N} \times \mathcal{L}$:

- if the label ℓ is used by the runnable i , we define the *gain* $g_{i,\ell}$ as the saved execution cycles by one invocation of the runnable i when the label ℓ is allocated to the LRAM linked to the CPU where the runnable i is mapped;
- we set $g_{i,\ell} = 0$ if the label ℓ is not used by runnable i .

The gain $g_{i,\ell}$ is expressed in clock cycles. Its calculation depends on many factors: type of access, size s_ℓ of ℓ , frequency of access, and so on. Later, in Section 8.8.1, we illustrate the gain models used in the experiments. We remark that our proposed methodology is independent of such a choice.

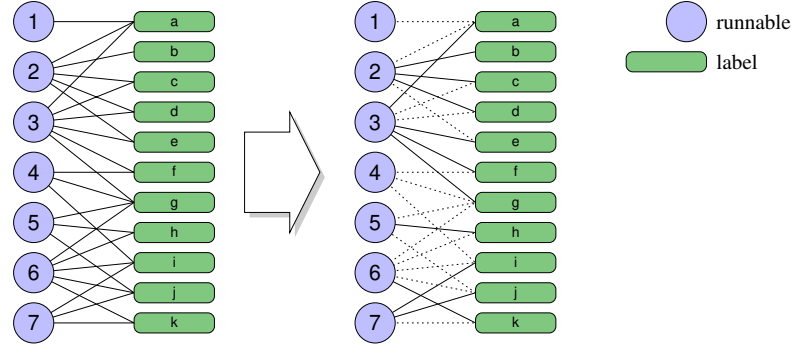


Fig. 8.6: Binding labels to runnables.

Variables

For the purpose of partitioning labels among runnables, we introduce the following variables

$$x_{i,\ell} = \begin{cases} 1 & \text{label } \ell \text{ is bound to runnable } i \\ 0 & \text{otherwise} \end{cases} \quad i \in \mathcal{N}, \ell \in \mathcal{L} \quad (8.1)$$

and we define the partition of labels by

$$\mathcal{L}_i = \{\ell \in \mathcal{L} : x_{i,\ell} = 1\} \quad i \in \mathcal{N}.$$

Constraints

If we denote by S_i the amount (unknown) of LRAM assigned to labels in \mathcal{L}_i , then the following constraint

$$\sum_{\ell \in \mathcal{L}} s_\ell x_{i,\ell} \leq S_i \quad i \in \mathcal{N} \quad (8.2)$$

ensures that the total memory local to the runnable i is not exceeded, while the next one

$$\sum_{i \in \mathcal{N}} S_i \leq \sum_{k \in \mathcal{M}} S_k^{\text{cpu}} \quad (8.3)$$

is needed not to exceed the total LRAM. If needed, our formulation can also include a constraint on the maximum amount of memory S_i needed by runnable i .

Finally, it is certainly needed to assign a label to at most one runnable; that is,

$$\sum_{i \in \mathcal{N}} x_{i,\ell} \leq 1 \quad \ell \in \mathcal{L}. \quad (8.4)$$

Notice that the subsets \mathcal{L}_i are not a partition (that is, constraint (8.4) is not written with the “=” sign but with the “ \leq ” one) as there may be some labels that are not bound to any runnable.

Objective function

The natural aim of the binding is to minimize the resource usage by runnables. In fact, wherever every runnable i is mapped, we are certain that **accesses to labels in \mathcal{L}_i are through local links**. Since each runnable executes with frequency f_i , the metric to be maximized is

$$\sum_{i \in \mathcal{N}} f_i \sum_{\ell \in \mathcal{L}} (g_{i,\ell} \cdot x_{i,\ell}). \quad (8.5)$$

The rationale of the cost of equation (8.5) is to bind label ℓ to the runnable i that can benefit the most in terms of saving CPU utilization, since it has the largest utilization gain represented by execution time saving $g_{i,\ell}$ multiplied by the frequency of execution f_i .

Size of the problem

For this problem, the number of variables $x_{i,\ell}$ (8.1) is $\mathcal{N} \times \mathcal{L}$, while the number of constraints is $\mathcal{N} + \mathcal{L} + 1$ following from (8.2)–(8.4).

8.4.1 Polynomial-time algorithms

If the overall amount of LRAM is sufficient to store all labels, then constraint (8.3) is never active. This means that the only active constraint remains the one in equation (8.4) and that the optimal solution is

$$x_{i,\ell} = 1 \Leftrightarrow i = \operatorname{argmax}_{j \in \mathcal{N}} \{f_j \cdot g_{j,\ell}\} \quad \ell \in \mathcal{L} \quad (8.6)$$

which is found in $\mathcal{O}(\mathcal{N} \times \mathcal{L})$ time. For such a solution, every label ℓ brings the maximum saving of resource usage, which is $G_\ell = \max_j \{f_j \cdot g_{j,\ell}\}$.

If instead the constraint of equation (8.3) is active, then the problem becomes the knapsack problem. The continuous relaxation of this problem is solved exactly in $\mathcal{O}(\mathcal{L} \times \log(\mathcal{L}))$ time as suggested by Dantzig [85] and described below.

1. Labels are sorted by decreasing *gain density* as follows:

$$\frac{G_\ell}{s_\ell} \geq \frac{G_{\ell'}}{s_{\ell'}} \quad \ell, \ell' \in \mathcal{L}, \ell < \ell'. \quad (8.7)$$

2. Labels are selected, and assigned to the runnable of equation (8.6), following the ordering of equation (8.7) until the memory capacity constraint (8.3) is not violated.
3. Let $z \in \mathcal{L}$ be the *critical item*, which is the first label that, according to the ordering of equation (8.7), violates the capacity constraint (8.3).

If we set

$$S^{\text{slack}} = \sum_{k \in \mathcal{M}} S_k^{\text{cpu}} - \sum_{\ell=1}^{z-1} s_\ell,$$

which is the remaining memory capacity after allocating the first $z - 1$ labels, then the optimum is found by taking a fraction $\frac{S^{\text{slack}}}{s_z}$ of the label z . Such an optimal solution has the following (see equation (8.5)) maximal metric:

$$\sum_{\ell=1}^{z-1} G_\ell + \frac{S^{\text{slack}}}{s_z} G_z.$$

In our polynomial time greedy algorithm:

1. we drop the critical item “label z ” and bind labels up to the one in position $z - 1$ in the ordering of equation (8.7), and
2. we fill up the remaining capacity S^{slack} with any label that fits.

Distance to optimality

The maximum penalty of this solution is

$$\frac{S^{\text{slack}}}{s_z} \cdot G_z \leq G_z \leq \max_{\ell \in \mathcal{L}} G_\ell = \max_{\ell \in \mathcal{L}, i \in \mathcal{N}} \{f_i \cdot g_{i,\ell}\}.$$

If the granularity $f_i \cdot g_{i,\ell} \ll 1$, as in our use case (Kramer et al. [101]), the penalty is negligible. Our experiments confirm the quality of the polynomial greedy algorithm, since its solution and the optimal one are nearly indistinguishable.

8.5 Mapping runnables to CPUs

The mapping of the runnables in \mathcal{N} over the available CPUs can be formalized as an IP with binary variables.

Variables

We model the mapping over the CPUs by $\mathcal{N} \times \mathcal{M}$ variables $y_{i,k}$ with the following interpretation:

$$y_{i,k} = \begin{cases} 1 & \text{runnable } i \text{ is mapped to CPU } k \\ 0 & \text{otherwise} \end{cases} \quad i \in \mathcal{N}, k \in \mathcal{M}. \quad (8.8)$$

We remind that if $y_{i,k} = 1$, then the bound labels in \mathcal{L}_i will be mapped to the LRAM associated to CPU k .

The distinguishing feature that needs to be captured in the problem of mapping runnables to CPUs is whether or not any pair of runnables is mapped onto the same core. In fact, if runnable i is on the same CPU as runnable j , it may save processing time if it uses labels in \mathcal{L}_j . Motivated by this observation, we add the following variables to the problem formulation:

$$x_{i,j}^{\text{same}} = \begin{cases} 1 & \text{runnables } i \text{ and } j \text{ mapped on same CPU} \\ 0 & \text{otherwise} \end{cases} \quad i, j \in \mathcal{N}, i < j. \quad (8.9)$$

Finally, an additional continuous *slack* variable z is added. Such a variable represents the “extensibility” of software for future updates and it is going to be maximized. A negative value of z indicates an infeasible design.

Constraints

If runnables i and j are bound to the same CPU k , then the corresponding variable $x_{i,j}^{\text{same}}$ must be equal to one:

$$x_{i,j}^{\text{same}} = \max_{k \in \mathcal{M}} (y_{i,k} + y_{j,k}) - 1 \quad i, j \in \mathcal{N}, i < j, \quad (8.10)$$

which can also be written as a linear constraint, by adding extra variables $x_{i,j,k}^{\text{same}}$ for each $k \in \mathcal{M}$.

Each runnable is mapped over one CPU only, that is

$$\sum_{k \in \mathcal{M}} y_{i,k} = 1 \quad i \in \mathcal{N}. \quad (8.11)$$

Notice that if a runnable i must be necessarily mapped to some specific CPU k (for example, some of the available CPUs offer some hardware features or accelerator which are necessary to the runnable), it is possible to encode such a constraint by setting $y_{i,k} = 1$.

The variables $x_{i,j}^{\text{same}}$ as defined by equation (8.9) clearly imply an equivalence relation. Hence, we enforce the following properties.

- **Reflexivity** is enforced implicitly by omitting the variables $x_{i,i}^{\text{same}}$, as it would always be $x_{i,i}^{\text{same}} = 1$.
- **Symmetry** is enforced implicitly by having only one variable $x_{i,j}^{\text{same}}$ for both ordered pairs (i, j) and (j, i) . To have a more convenient notation, we may be using $x_{i,j}^{\text{same}}$ with $i > j$. When this happens, we mean $x_{j,i}^{\text{same}}$.
- **Transitivity** that is

$$(x_{i,j}^{\text{same}} = 1) \wedge (x_{j,\ell}^{\text{same}} = 1) \Rightarrow (x_{i,\ell}^{\text{same}} = 1).$$

Transitivity is not explicitly enforced because implied by equations (8.10) and (8.11). In fact, if $x_{i,j}^{\text{same}} = 1$, from constraint (8.10) it must exist $k_1 \in \mathcal{M}$ such that $y_{i,k_1} = y_{j,k_1} = 1$. For the same reason, $x_{j,\ell}^{\text{same}} = 1$ implies that it must exist $k_2 \in \mathcal{M}$ such that $y_{j,k_2} = y_{\ell,k_2} = 1$. Constraint (8.11) implies that $k_1 = k_2$ and then from $y_{i,k_1} = y_{\ell,k_1} = 1$ we have $x_{i,\ell}^{\text{same}} = 1$.

Furthermore, applications may require two or more runnables to be scheduled together over the same CPU. For example, in the automotive AUTOSAR standard, runnables may belong to Software Components (SWCs), which need to be mapped to the same core. This can be easily encoded constraining $x_{i,j}^{\text{same}} = 1$ for all the runnables i and j belonging to the same SWC.

Before formulating the constraints on the CPU capacity, let us introduce the following notation.

- C_i^0 denotes the execution cycles of runnable i , assuming that:
 - all labels in \mathcal{L}_i , bound to runnable i as described in Section 8.4, are stored in LRAM and then enjoy a faster access;
 - all other labels are stored in GRAM.
- $\Delta C_{i,j}$ denotes the execution cycles saved by one invocation of runnable i if runnable j executes over the same CPU. $\Delta C_{i,j}$ is non-zero, if runnable i happens to use any label in \mathcal{L}_j . $\Delta C_{i,j}$ is written as function of the gains $g_{i,\ell}$ introduced in Section 8.4, as follows:

$$\Delta C_{i,j} = \sum_{\ell \in \mathcal{L}_j} g_{i,\ell} \quad i \in \mathcal{N}. \quad (8.12)$$

- The variable $C_{i,k} \geq 0$ represents the number of execution cycles required by runnable i over CPU k , that is

$$C_{i,k} \geq C_i^0 \cdot y_{i,k} - \sum_{j \in \mathcal{N}, j \neq i} (\Delta C_{i,j} \cdot x_{i,j}^{\text{same}}) \quad k \in \mathcal{M}. \quad (8.13)$$

It is worth noting that, if the solver does not map runnable i to CPU k , the value of $C_{i,k}$ is set to zero.

As stated in Section 8.4, when runnable i is partitioned it also carries an amount of needed local memory S_i . The constraint of limited size of the local memory is formulated as

$$\frac{1}{S_k^{\text{cpu}}} \sum_{i \in \mathcal{N}} (y_{i,k} \cdot S_i) \leq 1 - \alpha^{\text{mem}} z \quad k \in \mathcal{M} \quad (8.14)$$

while the constraint on the CPU capacity is

$$\sum_{i \in \mathcal{N}} (f_i \cdot C_{i,k}) \leq 1 - \alpha^{\text{cpu}} z \quad k \in \mathcal{M}. \quad (8.15)$$

The weights α^{mem} and α^{cpu} represent the relevance of the *slack* in each constraint and can be freely chosen by the designer. A large value of α^{mem} or α^{cpu} encodes the goal of having much slack in the constraint, while a value of zero informs the solver that the constraint can also hold tightly.

Objective function

The goal of the design is to maximize the “extensibility” of software for future updates, that is

$$\text{maximize } z. \quad (8.16)$$

If the optimal z^* found is negative then the problem is not feasible. If $z^* \geq \frac{(1-U_{\text{LL}})}{\alpha_{\text{cpu}}}$, with U_{LL} equals to the utilization upper bound $\log 2 \approx 0.693$ by Liu and Layland [105], the problem is feasible. Otherwise, schedulability is ensured by the next step of the design (the assignment of priority described in Section 8.7). Notice that the proposed design goal generalizes the typical optimization objective borrowed from the literature (Pazzaglia et al. [119]).

Size of the problem

Summarizing, the total number of variables is $2(\mathcal{N} \times \mathcal{M})$, counting $y_{i,k}$ of equation (8.8) and $C_{i,k}$ of equation (8.13). The number of constraints is $\mathcal{O}(\mathcal{N}^2)$, dominated by (8.10). In real-world scenarios, $\mathcal{N} \approx 1000$ as shown in Tab. 8.1, making the number of constraints in the order of millions.

This is true even without considering the linearization of constraint (8.10), that would amplify the magnitude of the number of x^{same} variables and the numerosity of constraint (8.10) by a factor \mathcal{M} (the number of cores).

When such a problem becomes intractable, a different approach, illustrated next, is required.

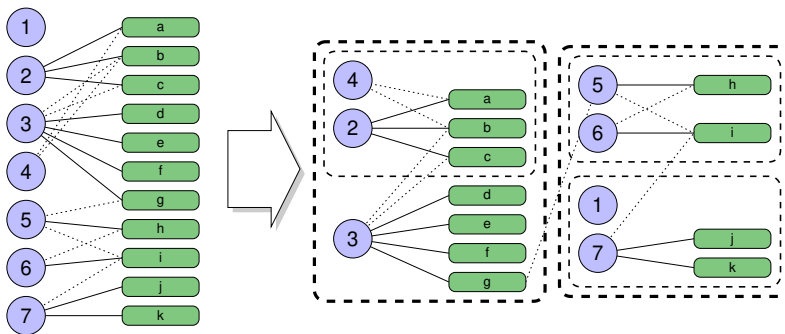


Fig. 8.7: Merging runnables in clusters.

8.5.1 Hierarchical Clustering

To mitigate the issues due to the size of the problem, we borrow the methods of Hierarchical Clustering (HC) from the literature (Murtagh and Contreras [114]). With HC, runnables are aggregated into clusters (Fig. 8.7) of tunable size. Then the mapping described earlier in Section 8.5 is applied to the fewer clusters, rather than to all runnables. An advantage of HC is that the size of clusters can be set by the designer to trade accuracy versus tractability. Also, HC is particularly well suited for partitioning very many “small” items, as in our use case.

Algorithm 10 Hierarchical Clustering.

```
1: function HC( $\mathcal{N}, \mathcal{U}$ )                                ▷ runnables, their utilizations
2:   clusters  $\leftarrow \{\{i\} : i \in \mathcal{N}\};$            ▷ initialize clusters
3:   utils  $\leftarrow \mathcal{U};$                                ▷ utilization of singleton clusters
4:   mergeTree  $\leftarrow [];$                              ▷ tracking of cluster merges
5:   for  $n$  from  $|\mathcal{N}| - 1$  to 1 do                       ▷ need  $|\mathcal{N}| - 1$  merges
6:      $c_{\min} \leftarrow \operatorname{argmin}\{\text{utils}\};$        ▷ min cluster utilization
7:      $U_{\text{avg}} \leftarrow (\sum\{\text{utils}\} - \text{utils}[c_{\min}])/n;$   ▷ average cluster utilization
8:     maxGain  $\leftarrow -1;$ 
9:     for  $c$  in clusters  $\setminus \{c_{\min}\}$  do           ▷ cannot merge a cluster with itself
10:      if  $\text{utils}[c] > U_{\text{avg}}$  then                   ▷ skip if cluster utilization is above average
11:        continue;
12:      end if
13:       $g \leftarrow \text{utilGain}(c_{\min}, c);$ 
14:      if  $g > \text{maxGain}$  then                         ▷ selecting cluster with maximum gain
15:        maxGain  $\leftarrow g;$ 
16:         $c_{\text{best}} \leftarrow c;$ 
17:      end if
18:    end for
19:    clusters  $\leftarrow \text{clusters} \setminus \{c_{\min}\} \setminus \{c_{\text{best}}\};$   ▷ remove min and best clusters
20:     $c_{\text{new}} \leftarrow c_{\min} \cup c_{\text{best}};$            ▷ merge clusters
21:    clusters  $\leftarrow \text{clusters} \cup \{c_{\text{new}}\}$        ▷ add new cluster
22:     $\text{utils}[c_{\text{new}}] \leftarrow \text{utils}[c_{\min}] + \text{utils}[c_{\text{best}}] - \text{maxGain};$   ▷ update utilization
23:    mergeTree[ $n$ ]  $\leftarrow \text{clusters};$                ▷ save found  $n$  clusters
24:  end for
25:  return mergeTree;
26: end function
```

The full procedure of HC is outlined in Algorithm 10. The initial clustering set clusters is initialized (line 2) by considering all runnables as singleton clusters. The array utils contains the CPU utilization of all clusters (line 3) and for any cluster c we denote its utilization of $\text{utils}[c]$. The array of clusters mergeTree (initialized at line 4) is meant to contain the set of all found clusters. More specifically, mergeTree[n] reports the solution of how all runnables in \mathcal{N} are partitioned in n clusters.

The loop from line 5 to line 24 picks a pair of clusters and merges them until a unique cluster with all runnables is created. At every iteration, our algorithm first

picks the cluster c_{\min} with the lowest utilization (line 6). The second cluster c_{best} to be merged with c_{\min} satisfies two properties:

1. it has utilization no greater than the average utilization of clusters (enforced by the condition at line 10);
2. it has the highest gain maxGain if paired with c_{\min} (enforced by the condition at line 14).

The choice of this merging rule was proven to keep a very balanced utilization of the clusters, while leading towards the best possible decrease in utilization for all merge operations.

Once the joining pair of clusters is chosen, they are removed from the set (line 19), all runnables that they contain are merged in a new cluster c_{new} (at line 20), which is then added to the set (line 21). At line 22, the utilization of the new cluster c_{new} is computed accounting for the utilization of each of the merged clusters and the gain maxGain they have from being together.

Finally, the clustering obtained for level n is added into the clustering tree (line 23) as the element with key n ; the procedure stops when, at level 1, all runnables are merged in a single cluster.

Considerations

An immediate advantage of HC is the availability of the entire hierarchical tree recording the history of the merges. Such a tree enables the designer to choose the desired level of granularity. Once a level of granularity has been chosen, the newly formed clusters are considered as “runnables” for the model described in Section 8.5 and the problem of mapping runnables to CPUs is solved with an orders of magnitude smaller set \mathcal{N} .

In the classical HC, at each step, the merge is always done by joining the two clusters with the *best possible similarity*, hence in our case the *highest possible gain*; but that is not the approach followed in this work, for two specific reasons. Firstly, it is a costly approach, involving a potentially quadratic search at every step, and since the number of runnables is very high the time required for the clustering can increase and impact hugely on the performance of the entire procedure. Secondly, and more importantly, in this specific problem blindly joining at every step the two clusters with highest reciprocal gain leads eventually to extremely unbalanced clustering; this can be easily seen considering the fact that if a cluster contains runnables with a lot of labels required also by other runnables in unmerged clusters, this kind of approach will untimely merge at every step this favorable cluster with all others. This leads to a situation where in the clustering tree at each level n there is one huge cluster growing, and $n - 1$ clusters of (probably) runnable singletons.

8.6 Aggregation of runnables into tasks

Runnables, we remind, are equivalent to functions to be properly invoked. The division of software in runnables responds to specifications and principles at application design level. At the lower OS level, instead, it may be infeasible (due to the potentially large number of runnables) and it is certainly inefficient (due to context switches) to dedicate an OS task to each runnable. For example, in the 2017 WATERS Challenge (Kramer et al. [101]), there are 1250 runnables, but only 21 tasks executing them, as reported in Tab. 8.1. It is then necessary to establish criteria to aggregate runnables.

Our approach to aggregate runnables in tasks is applied after the optimal mapping of runnables (or clusters of runnables) to CPUs is performed as described in Section 8.5. We assume then to have a solution to the mapping represented by the variables $y_{i,k}$ and $x_{i,j}^{\text{same}}$ defined in (8.8) and (8.9), respectively. Let us now formalize the aggregation of runnables.

- The set of tasks is denoted by \mathcal{T} .
- The set of runnables in \mathcal{N} to form task t is denoted by $\mathcal{N}_t \subseteq \mathcal{N}$. The subsets in $\{\mathcal{N}_t\}_{t \in \mathcal{T}}$ form a partition of \mathcal{N} such that every runnable $i \in \mathcal{N}$ belongs to one and only one subset \mathcal{N}_t .
- The *equivalence relation* \sim over the pairs of runnables $\mathcal{N} \times \mathcal{N}$ encodes the aggregation of runnables. In our case, $i \sim j$ if “the two runnables i and j have the same release period and none of them self-suspends”. We remark that our methodology works for any other choice \sim providing the properties of equivalence relations.
- The runnables in \mathcal{N}_t belonging to the same task t are defined as

$$i, j \in \mathcal{N}_t \Leftrightarrow (i \sim j) \wedge x_{i,j}^{\text{same}} = 1 \quad (8.17)$$

with $x_{i,j}^{\text{same}}$ being the variables representing the optimal mapping found in Section 8.5.

From our definition of \sim above, it follows that

$$i \sim j \Rightarrow T_i = T_j, \quad (8.18)$$

meaning that two runnables with different period cannot be aggregated together in the same task. Such assumption holds for the 2017 WATERS Challenge (Kramer et al. [101]) and is recommended in software design. Still, having runnables with different period in the same task is possible. In such a case, the task implementation simulates the different periods by counting the invocations of each runnable and the execution

pattern of the task becomes analogous to the multi-frame task model (Mok and Chen [113]). The analysis of this case, however, is left to future investigations.

The definition of equation (8.17) partitions runnables to tasks. We can now define the parameters of tasks, starting from the parameters of the runnables.

- From (8.13), the execution cycles of task $t \in \mathcal{T}$ are

$$C_t = \sum_{i \in \mathcal{N}_t} \left(C_i^0 - \sum_{j \in \mathcal{N}, x_{i,j}^{\text{same}}=1} \Delta C_{i,j} \right). \quad (8.19)$$

- Because of (8.18), all runnables of a task have the same period (or minimum interarrival time). Hence, $\forall t \in \mathcal{T}$, we set $T_t = T_i$, picking any $i \in \mathcal{N}_t$.
- The deadline of task $t \in \mathcal{T}$ is

$$D_t = \min_{i \in \mathcal{N}_t} D_i.$$

- Finally, it is useful to introduce the partition of tasks over the CPUs in \mathcal{M} . We define

$$\mathcal{T}_k = \{t \in \mathcal{T} : i \in \mathcal{N}_t, y_{i,k} = 1\}.$$

Notice that this is a good definition because if $y_{i,k} = 1$ for some runnable $i \in \mathcal{N}_t$, then $y_{j,k} = 1$ for all $j \in \mathcal{N}_t$.

8.7 Assigning priorities to tasks

For each CPU k , the priorities of the tasks in \mathcal{T}_k are assigned based on the Robust Priority Assignment (RPA) by Davis and Burns [86], recalled in Algorithm 11. The only adaptation with reference to the original RPA is the computation of the maximum slack z_i at line 8. Rather than using binary search as originally proposed (Davis and Burns [86]), we borrow the *sensitivity analysis* by Bini et al. [79] to find the exact expression of the per-task slack z_i . More precisely, from the exact schedulability condition (Lehoczky et al. [103]) properly modified to account for the per-task weighted slack z_i

$$\exists t \in \mathcal{P}_i : \frac{1}{t} \left(C_i + B_i + \sum_{j \in \mathcal{T}_{\text{hp}}} \left\lceil \frac{t}{T_j} \right\rceil C_j \right) \leq 1 - \alpha_i^{\text{sched}} z_i$$

with

$$\mathcal{P}_i = \{D_i\} \cup \{kT_j : j \in \mathcal{T}_{\text{hp}}, 0 < kT_j < D_i, k \in \mathbb{N}\}, \quad (8.20)$$

we find

$$z_i \leq \max_{t \in \mathcal{P}_i} \frac{t - \left(C_i + B_i + \sum_{j \in \mathcal{T}_{\text{hp}}} \left\lceil \frac{t}{T_j} \right\rceil C_j \right)}{\alpha_i^{\text{sched}} t} \quad (8.21)$$

which is the expression used at line 8 of Algorithm 11 for computing z_i .

Algorithm 11 Robust Priority Assignment.

```

1: function RPA( $\mathcal{T}_k$ ) ▷ tasks mapped to CPU  $k$ 
2:   pri  $\leftarrow$  lowest priority available;
3:    $\mathcal{T}_{\text{unassigned}} \leftarrow \mathcal{T}_k$ ;
4:   while  $\mathcal{T}_{\text{unassigned}}$  not empty; do
5:      $z_{\text{best}} \leftarrow -\infty$ ;
6:     for  $i$  in  $\mathcal{T}_{\text{unassigned}}$  do
7:        $\mathcal{T}_{\text{hp}} \leftarrow \mathcal{T}_{\text{unassigned}} \setminus \{i\}$ ;
8:        $z_i \leftarrow$  slack of task  $i$ ; ▷ RHS of (8.21)
9:       if  $z_i > z_{\text{best}}$  then ▷ found a better task
10:         $z_{\text{best}} \leftarrow z_i$ ;
11:         $i_{\text{best}} \leftarrow i$ ;
12:       end if
13:     end for
14:     priority[ $i_{\text{best}}$ ]  $\leftarrow$  pri;
15:     pri  $\leftarrow$  next priority higher than pri;
16:      $\mathcal{T}_{\text{unassigned}} \leftarrow \mathcal{T}_{\text{unassigned}} \setminus \{i_{\text{best}}\}$ ;
17:   end while
18:   return priority;
19: end function

```

The blocking time B_i in equation (8.21) is the *waiting* time spent by task i for the execution of lower priority tasks on the same CPU or any task on other CPUs. This may happen because of:

- an attempt to access to a shared resource locked by
 - a lower priority task within the same CPU, or
 - a task executing on a different CPU;
- the invocation of a blocking system call such as a remote procedure call (allowed by the AUTOSAR standard, for example).

Several protocols can be used to protect resources shared globally. In the case of the AUTOSAR standard, a discussion on their applicability to AUTOSAR and blocking times can be found in Wieder and Brandenburg [129] and in Yang et al. [131], where linear formulations for lock-based protocols that admit a bounded blocking time are presented. AUTOSAR requires that a task can be terminated at any time, even when waiting (in a spin lock) for a global resource. This can be in principle solved by using the protocols described in Craig [83] and in Takada and Sakamura [125]. We remark that shared resource protocols for multiprocessor systems are outside the

scope of this work and that our methodology can nevertheless integrate any protocol listed above.

Complexity

Algorithm 11 is pseudo-polynomial because it depends upon the cardinality of \mathcal{P}_i from equation (8.20). In our experiments, the entire execution of RPA algorithm completed in a matter of tenths of a second.

8.8 Experiments

Our experiments are based on the use case of an automotive application provided by the WATERS 2017 Challenge (Kramer et al. [101]), whose size was reported earlier in Tab. 8.1. Our methodology is relevant for those applications which make an intensive use of labels in memory. Applications with CPU bound work and few memory operations do not particularly benefit by our approach since the impact of the mapping of labels is negligible.

8.8.1 The use case

In our experiments, we borrowed the values of the memory access times from the datasheet of TC39x architectures. Tab. 8.2 reports the stall cycles to access a 32-bit word.

- The cycles for read and write accesses are reported.
- Since write operations are buffered, stall cycles “5, 3” mean:
 - 5 cycles for the first write of a 32-bit word;
 - 3 cycles for the next consecutive writes.
- The column “Local CPU” reports the access time from a CPU to the directly connected LRAM (direct accesses to GRAM are not possible as shown in the architecture of Fig. 8.2).
- The column “Remote CPU” reports the time for accesses from a CPU to a LRAM or to the GRAM which need to traverse the crossbar (shown as a horizontal arrow in Fig. 8.2).

Access times for fetching instructions are not reported because memory for storing program code is normally not a stringent constraints and for this reason is not addressed.

Tab. 8.2: Stall cycles for memory accesses in TC39x. LRAM denotes: Data ScratchPad RAM (DSPR) and distributed Local Memory Units (dLMUs). GRAM denotes: Local Memory Units (LMUs) and Default Application Memory (DAM).

Type	Local CPU		Remote CPU	
	read	write	read	write
LRAM	0	0	7	5, 3
GRAM	n.a.	n.a.	7	5, 3

Given these characteristics, we model the gain $g_{i,\ell}$ of the cycles saved by one invocation of runnable i when accessing the label ℓ in the directly connected LRAM, by

$$g_{i,\ell} = \begin{cases} 7 \times \text{num}_i \times \lceil \frac{s_\ell}{4} \rceil & \text{read accesses} \\ 5 + 3 \times (\lceil \frac{s_\ell}{4} \rceil - 1) & \text{write accesses} \end{cases}$$

with:

- s_ℓ denoting the size of the label ℓ in bytes;
- $\lceil \frac{s_\ell}{4} \rceil$ accounting for accesses made by 4-bytes words;
- num_i denoting the number of reads by the one invocation of runnable i ;
- the expression of the write cycles accounting for the different number of write cycles to the first word and the following ones in presence of write buffers.

The model of the gain $g_{i,\ell}$ assumes that a runnable writes to a label only once per invocation. This assumption originates from an inspection of the WATERS 2017 use case, which does not contain any information about the access statistics for the labels to be written. Clearly, our approach can account for labels to be written more than once whenever this information is available. Moreover, we underline that our approach is compatible with any model of the memory access times such as the one proposed in the WATERS 2017 use case (which is 1 cycle to access to the directly linked LRAM, 9 cycles for other accesses) or others. In fact, the gains $g_{i,\ell}$ are fed as input to our problem.

To avoid trivial solutions (in which, for example, all runnables fit onto the same core or the mapping is infeasible), we scaled the execution cycles of all runnables such that:

- if no label is in LRAM, then the total utilization $\sum_{i \in \mathcal{N}} (f_i \cdot C_i)$ is equal to 2.110 (still fitting on 4 CPUs);
- if all runnables have their used labels in the directly linked LRAM, then the total utilization is equal to 1.479 (not fitting on a single CPU).

These values allow exploring rich scenarios for the mapping over 4 identical CPUs, and set upper and lower bounds to the total utilization of any solution.

8.8.2 A Simulated Annealing approach

To evaluate the quality of the IP solution, we implemented a Simulated Annealing (SA) mapping optimizer (Aarts and Korst [73]) as baseline. The choice upon a SA algorithm in spite of other meta-heuristics or genetic algorithms is twofold. Firstly, SA is a generally applicable and easy-to-implement stochastic approximation approach, and it is able to produce good solutions for an optimization problem even if the underlying structure of the problem is not obvious nor easily understandable. Moreover, SA algorithms in the past history (Bertsimas and Tsitsiklis [78]) have outperformed the best known heuristics for several problems, while for other problems their performance was comparable to specialized heuristics finely-crafted to solve exactly those specific problems.

SA belongs to the category of *randomized optimization* techniques and aims at optimizing a given objective function by performing a sequence of random changes to the system configuration, generating at every iteration a new mapping solution. At each step, the new configuration is evaluated and retained if its performance is better than the previous solution. If the new solution is worse, it can still be conditionally accepted with a probability P computed by

$$P = e^{\frac{-\Delta V}{T}}$$

which exponentially decreases with “temperature” T ; this prevents being stuck in a local optimum. ΔV is the difference between the current and the new performance values.

The performance metric is the same as the IP (maximization of the slack, as in equation (8.16)). The constraints are not explicit, to let the algorithm explore unfeasible regions of the solution space and guarantee the reachability of any possible configuration. However, unfeasible configurations are penalized in the objective function, hence the SA moves towards feasible solutions in the end.

The random modifications to the system configuration are realized by 4 *transition operators*.

- *Task priority assignment*: this operator picks a task of the system and sets its priority to a different value.
- *Runnable-to-task mapping*: this operator chooses a runnable of the system and moves it to another task. The runnable can be moved to an existing task, or a newly created one. If the task where the runnable was originally allocated contains no runnables after the transition, it is removed from the system.

- *Task-to-CPU mapping*: this operator picks a task of the system and moves it to a different CPU.
- *Label-to-memory mapping*: this operator chooses a label and places it to a different memory where it can fit according to the currently available space.

At every step, the algorithm randomly chooses the number of consecutive transition operators to apply to the current solution (between 1 and 3) and then randomly picks the operators. Applying more than one operator at a time helps the SA escape local optima during the search.

The algorithm was programmed to run with an initial temperature of 20000, a final temperature of 0.0001, a cooling rate of 0.94, a maximum number of temperature values (MAXNUMCHAINS) of 4000000, a maximum number of iterations for each temperature value (MAXTRY) of 400000, a maximum number of acceptable configurations for each temperature value (MAXCHANGE) of 20000 and a penalty multiplier 10 (in cost) for unfeasible configurations.

8.8.3 Setup for our approach

The executions of the solver are performed under a Linux environment over one of the cores available in a machine equipped with Intel Core i7-11700 CPU @ 2.50GHz.

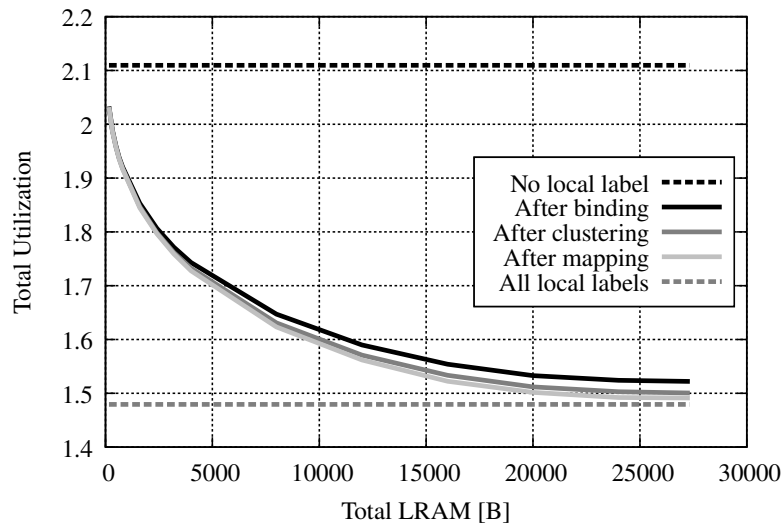


Fig. 8.8: Utilization as function of the allocated memory, with 40 clusters.

To ease the process of data extraction and analysis of the 2017 WATERS Challenge (Kramer et al. [101]), described in XML format, we implemented the parsing process, the two polynomial algorithms to bind labels to runnables (Section 8.4.1) and to aggregate runnables in clusters (Section 8.5.1), and the pseudo-polynomial algorithm to assign priorities (described in Section 8.7) using Python 3 programming language.

The solver of the IP problem to map clusters to CPUs of Section 8.5 is COIN-OR Cbc (Lougee-Heimer [106]). Cbc is one of the best open-source integer optimization solvers, and it is developed and maintained explicitly for research by the non-profit COIN-OR Foundation. The communication between the main Python 3 code and Cbc solver is made by Python’s library Pyomo (Hart et al. [96]). The solver was invoked as a single thread.

We underline that we purposely targeted a not-so-performing implementation (that is, an interpreted language and a single-thread solver) to focus exclusively on the optimization problems and leave room to further performance improvements by those willing to make an industrial product out of our research prototype.

The weights to the slack α^{mem} , α^{cpu} , and α_i^{sched} from (8.14), (8.15) and (8.21) are set as follows:

- $\alpha^{\text{mem}} = 0$ and $\alpha^{\text{cpu}} = 1$ meaning that we aim at balancing the total utilization among CPUs;
- $\alpha_i^{\text{sched}} = 1$ meaning that we equally weight all tasks.

We tried different weights, which did not highlight any different behavior.

8.8.4 Computational results

The experiment of Fig. 8.8, made with 40 clusters, shows the impact of the amount of allocated LRAM onto the total utilization of the application. For reference, we also plot the upper and lower bounds to the utilization found as described in Section 8.8.1.

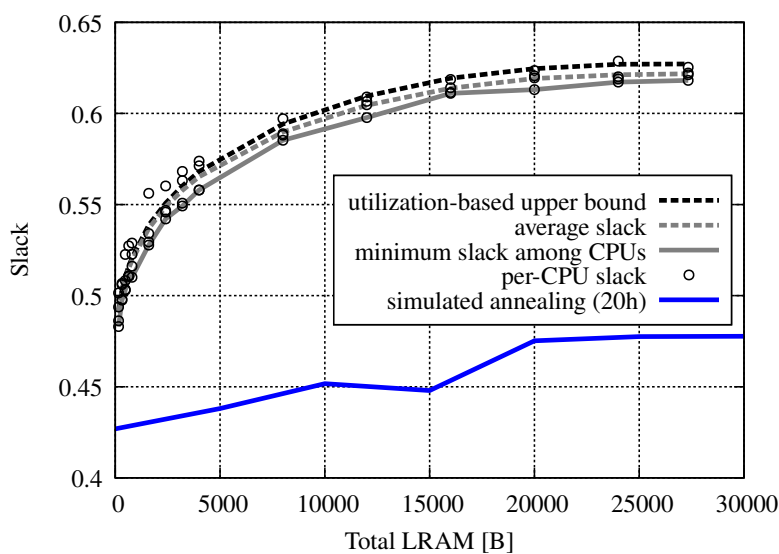


Fig. 8.9: Slack of the mapping, with 40 clusters.

The “After binding” plot corresponds to the total utilization of the whole application after labels are bound to runnables (as described in Section 8.4.1). Then, “After clustering” accounts for the extra utilization savings achieved by HC (Section 8.5.1). Finally, “After mapping” is the utilization after the clusters are mapped to CPUs.

As expected, the more LRAM is available for storing labels, the lower the final utilization it is. We also observe that a much steeper descent is achieved for the low values of LRAM; our explanation is that storing in LRAM a very frequently used label has a greater impact on utilization than storing another one with the same size, but used less frequently.

Fig. 8.9 shows the achieved slack. As the IP optimization targets the maximization of the minimum slack among CPUs, we observe that all CPU slacks are quite balanced. It is very striking to observe that SA, despite running about 1500 times longer than the proposed IP approach, is always achieving significantly inferior results.

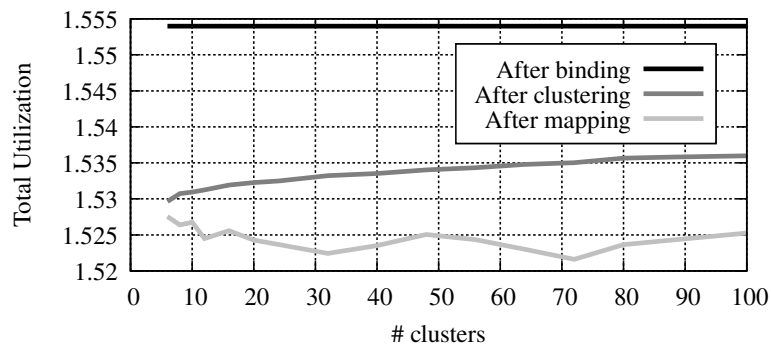


Fig. 8.10: Utilization as function of the number of clusters.

The impact of the number of clusters on the utilization can be seen in Fig. 8.10, and is as expected. As clusters get merged, their total utilization decreases, because the merged pair takes advantage of the commonly used labels.

On the other hand, if clusters are too few, then the mapping has really little maneuvering margin to allocate clusters which are then very coarse grained. To our experience, a number between 20 and 40 demonstrated to be a good compromise when mapping over 4 CPUs.

Tab. 8.3: Run-time of binding, clustering, and priority assignment.

Phase	Time (milliseconds)	
	average	std deviation
Binding labels	10.7	8.1
Clustering runnables	338.6	40.0
Priority assignment	17.4	12.1

Let us now examine the run-time of the whole method. The run time taken by the binding (Section 8.4.1), the clustering (Section 8.5.1), and the priority assignment

(Section 8.7) is negligible compared to the time taken to map clusters over CPUs. Tab. 8.3 reports the average and standard deviation of their run-times. Fig. 8.11 shows the run-time of the mapping of clusters (of runnables); not surprisingly, it grows rapidly with the number of clusters. We also observe that if the amount of allocated LRAM is smaller (3.2K instead of 16K) then the run-time is also smaller. This happens because if the LRAM is large, then there are more pairs of clusters with potential utilization savings by staying together.

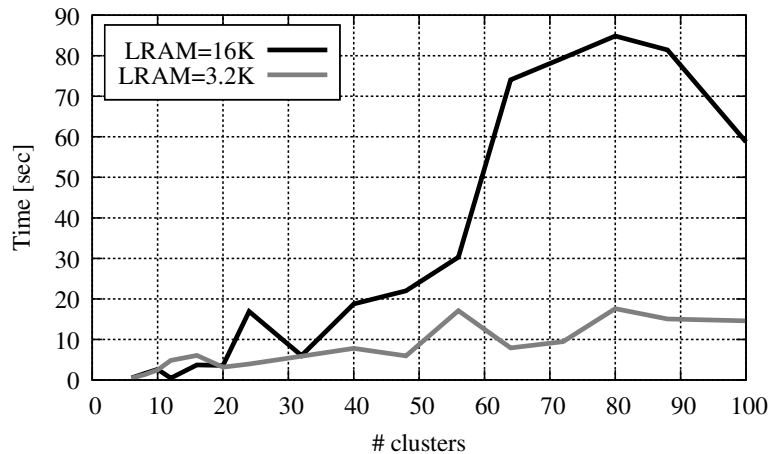


Fig. 8.11: Run-time of mapping of clusters.

Our explanation for the observed decrease of the run time when the number of clusters approaches 100 is that for such a value, the clusters gets smaller and smaller. Hence, the pruning rules of the solver operate very effectively.

8.9 Final remarks

In this chapter we have illustrated a whole methodology for mapping complex embedded software over NUMA architectures, exploiting the features of the different memory areas. The key innovation of our approach resides in the “binding” phase that reduces the complexity of the problem by orders of magnitude.

In the future, we may exploit the efficiency of our method by integrating it with other tools including measurement-based timing analysis tools. Also, we may be investigating the adaptation of the mapping in response to variations in the application features or in the processing capacity. Finally, a valuable direction of further investigation is the possibility to add end-to-end constraints, which are very typical of automotive applications.



lot of work has been done in the study of these problems, but even more work is still underway, both in the adaption of our succesful approaches to other problem families and in the analysis of new problems with this accumulated knowledge at hand. I am very happy of what we have managed to achieve during my PhD, and research is quickly moving forward to provide new state-of-the-art results and analysis over new problems.

In the following, a brief highlight of current research directions is given, with focus on aspects derived and relevant to my PhD thesis.

9.1 Flow models for parallel batching

With the work done in Part I, we managed to give an innovative and extremely tight lower bound for the parallel batching problem that aims to minimize the total completion time. This has become the new state-of-the-art for that family of problems, being able to even solve to optimality a lot of instances and giving excellent heuristic solutions for different variants of the main problem: in the weighted objective function version, in the case with incompatibilities, in the case with multiple job sizes (and in the case with both incompatibilities and multiple job sizes).

We are considering a third type of reformulation that can be able to cope with other objective functions, an application of our Column Generation (CG) strategy to the polynomial-size models, and implementation of our flow formulation approach on different scheduling problems that exhibit other types of complicating constraints.

9.1.1 Time-indexed formulation for exponential-size models

The flow models described in Chapter 4 and Chapter 6 can be considered built upon *job-indexed* graphs, since the number of nodes in the graph is always equal to the number of jobs (plus one). For a similar reason, the flow model described in Chapter 5 can be considered built upon a *weight-indexed* graph, since the number of nodes in the graph is always equal to the sum of weights of all jobs (plus one).

Unfortunately, these approaches are not able to consider several objective functions (like the total tardiness) and problem structures (like job release dates) because of the underlying evaluation of the cost of each arc that impacts because the way arcs are generated by the CG algorithm. To cope with these additional constraints and

other objective functions, we are working on a formulation built upon a *time-indexed* graph, with the number of nodes in the graph equal to the sum of processing times of all jobs (plus one). Obviously this can result in slower execution times with respect to the *job-indexed* and *weight-indexed* approaches, since the resulting model is undoubtedly bigger (processing times are typically higher than weights); but luckily in the preliminary tests we have done this impact is not too big that becomes untractable.

This modeling approach is currently under further investigations, especially with respect to the way new arcs can be efficiently generated with an adapted CG algorithm, and with respect to the objective functions that we are able to integrate with this technique.

9.1.2 Application of Column Generation to polynomial-size models

One reason our CG approach have excellent time performances in the exponential-size models is the fact that only very few columns are actually generated and added to the model, way less (by orders of magnitude) than the actual number of all possible columns, that is exponential. This is the main strength of good CG algorithms, and the principal fact that enables the writing and efficient solving of exponential-size models.

This specific approach is not, in fact, applied only where the number of variables is untractable, but can be implemented even when there is only a polynomial (but still high) number of variables. This is the case of the models described in Chapter 7, especially the stronger model of Section 7.1.3 where the number of variables is in the order of $\mathcal{O}(n^4)$ and for higher number of jobs leads to slow computational times.

Our aim in this direction is to modify the CG approach already implemented, and proven to be effective, for the exponential-size models, and apply them to the stronger of the two polynomial-size models, hoping that computational times will decrease significantly while generating only the columns relevant to the optimal (relaxed) solution. When CG techniques are implemented for polynomial-size models, one can simply *sift* all the possible columns by enumeration if a clever pricing formula does not exist; this solution is not at all viable for exponential-size models, obviously. In fact, the name of the approach changes, and becomes *Column Sifting* in this specific case.

9.1.3 Flow formulations for Common Server problems

The last research direction we have undertaken with regards to flow formulations applied to the scheduling world is related to the (unbatched) parallel machines problem in the presence of a common server. In the basic version of the problem, the aim is to minimize the makespan across all machines under the extra requirement that all jobs needs a specific *load* operation, done by a single *common server*, before the machine can start processing the job.

Following the modeling scheme already envisioned for Chapter 7, hence a flow model of polynomial (or pseudo-polynomial) size, we started to develop and analyze the performance of this kind of modeling technique for this Common Server problem.

Preliminary results are very promising, and will hopefully be published soon, defining a new state-of-the-art for this problem. In the immediate future we are already trying to adapt this approach to more difficult variants; for example, with more than one Common Server (adding an *unload* operation) or with extra complicating constraints over jobs.

9.2 Process scheduling in embedded systems

With the work done in Part II, we studied a challenging problem in process scheduling for specific embedded systems, obtaining excellent results both in term of solution quality and computational times. This has been done via the development of a specific, ad-hoc and fine-tuned algorithm for the entire labels/runnables mapping procedure that renders the problem tractable via *splitting* it in subsequent phases and solving each single step separately.

We now are thinking about the flexibility of our approach, in which cases it can still produce good results in presence of run-time modifications on the underlying system, and on which extent this approach can be adapted to cope with other specific problems of the same kind.

9.2.1 Handle modifications over runnables

As we stated in Section 8.9, one possible further investigation path could be the adaption of the mapping procedure in response to various modifications that could happen at run-time.

Consider we have a full mapping scheme of labels to runnables (hence to Local RAMs (LRAMs)) and of runnables to Central Processing Units (CPUs), and something changes in the underlying architecture or in the utilization required by runnables. For example, some runnables may have been modified and would require less (or more)

computational time to execute their job, or something changes in the communication speed between CPUs and the “distant” LRAMs or the Global RAM (GRAM).

If our aim is to strike a *balance* between utilization on all CPUs, the situation becomes unfavorable if those changes impact the utilization over CPUs in such a way that the slack over some of them is much higher than the slack over others. In this case, we can re-run our procedure in order to distribute more equally all labels and runnables between LRAMs and CPUs and achieve a balanced utilization again. But this approach can be potentially costly in terms of system management, requiring if modifications are mandatory even changes over the clustering of runnables, their mapping to the CPUs, their grouping in tasks and their priorities. One can decide to postpone such evaluation only after a certain *threshold* ratio over the reciprocal difference between slacks has been reached; for example, we can say that if the highest slack strays from the lowest slack less than 25% we consider that still acceptable and don't re-run our entire procedure.

If, instead, a change has happened in utilizations of runnables only because of some of them changed their used labels, a simpler thing can be done. One can try to run only the mapping of labels to runnables (Section 8.4.1) to bind labels to the new-best runnables, and see how (and if) the slack improves; if it is not the case, hence a full run of the entire procedure could be required.

9.2.2 Application to different architectures

In several parts of Chapter 8 we said that a specific evaluation or a peculiar consideration has been done for the application over our case-study.

For example, the gain in utilization $g_{i,\ell}$ that a runnable i that runs on a CPU exhibits while having a specific label ℓ assigned to the relative LRAM (Section 8.8.1) can be whatever the underlying architecture allows it to be. It is problem-specific data, and our method is entirely independent of such a choice.

This leads us to the possibility of exploration over a lot of different architectures that conforms to the basic abstract scheme described in Section 8.2, with little to no structural modifications to our approach.

Bibliography

Part I

- [1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993 (cit. on pp. 33, 63).
- [2] A. Alfieri, A. Druetto, A. Grosso, and F. Salassa. “Column generation for minimizing total completion time on a single machine with parallel batching”. In: *IFAC-PapersOnLine* 52.13 (2019), pp. 969–974 (cit. on pp. 14, 57, 70).
- [3] A. Alfieri, A. Druetto, A. Grosso, and F. Salassa. “Column generation for minimizing total completion time in a parallel-batching environment”. In: *Journal of Scheduling* 24 (Oct. 2021), pp. 569–588 (cit. on pp. 15, 57, 70, 82, 83, 86, 87, 95, 97, 98, 106).
- [4] J.E.C. Arroyo and J.Y.-T. Leung. “An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times”. In: *Computers & Industrial Engineering* 105 (Mar. 2017), pp. 84–100 (cit. on p. 11).
- [5] M. Azizoglu and S. Webster. “Scheduling a batch processing machine with non-identical job sizes”. In: *International Journal of Production Research* 38.10 (July 2000), pp. 2173–2184 (cit. on pp. 12, 27, 37, 50–52, 69, 74, 75, 102, 103, 105).
- [6] M. Azizoglu and S. Webster. “Scheduling a batch processing machine with incompatible job families”. In: *Computers & Industrial Engineering* 39.3–4 (Apr. 2001), pp. 325–335 (cit. on pp. 13, 87).
- [7] P. Baptiste. “Batching identical jobs”. In: *Mathematical Methods of Operations Research* 52.3 (Dec. 2000), pp. 355–367 (cit. on p. 10).
- [8] A. Bellanger, A. Janiak, M.Y. Kovalyov, and A. Oulamara. “Scheduling an unbounded batching machine with job processing time compatibilities”. In: *Discrete Applied Mathematics* 160.1 (Jan. 2012), pp. 15–23 (cit. on p. 11).
- [9] J.F. Benders. “Split-merge: Using exponential neighborhood search for scheduling a batching machine”. In: *Numerische Mathematik* 4 (Dec. 1962), pp. 238–252 (cit. on p. 12).
- [10] M. Boudhar. “Scheduling a batch processing machine with bipartite compatibility graphs”. In: *Mathematical Methods of Operations Research* 57.3 (Aug. 2003), pp. 513–527 (cit. on p. 10).
- [11] P. Brucker, A. Gladky, H. Hoogeveen, et al. “Scheduling a batching machine”. In: *Journal of Scheduling* 1.1 (Dec. 1998), pp. 31–54 (cit. on pp. 9–13).

- [12] M. Cabo, J.L. González-Velarde, E. Possani, and Y.Á. Ríos Solís. “Bi-objective scheduling on a restricted batching machine”. In: *Computers & Operations Research* 100 (Dec. 2018), pp. 201–210 (cit. on p. 12).
- [13] M. Cabo, E. Possani, C.N. Potts, and X. Song. “Split-merge: Using exponential neighborhood search for scheduling a batching machine”. In: *Computers & Operations Research* 63 (Nov. 2015), pp. 125–135 (cit. on p. 12).
- [14] G. Cachon and C. Terwiesch. *Matching Supply with Demand: An Introduction to Operations Management*. McGraw-Hill Education, 2012 (cit. on p. 2).
- [15] P. Damodaran, P. Kumar Manjeshwar, and K. Srihari. “Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms”. In: *International Journal of Production Economics* 103.2 (Oct. 2006), pp. 882–891 (cit. on pp. 10, 12).
- [16] S. Dauzère-Pérès and L. Mönch. “Scheduling jobs on a single batch processing machine with incompatible job families and weighted number of tardy jobs objective”. In: *Computers & Operations Research* 40.5 (May 2013), pp. 1224–1233 (cit. on p. 13).
- [17] J. Desrosiers and M. Lübbecke. “A Primer in Column Generation”. In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M.M. Solomon. Springer, Boston, MA, 2005 (cit. on p. 62).
- [18] J. Desrosiers and M.E. Lübbecke. “Branch-Price-and-Cut Algorithms”. In: *Wiley Encyclopedia of Operations Research and Management Science*. Ed. by John Wiley & Sons. American Cancer Society, Jan. 2011 (cit. on p. 27).
- [19] G. Dobson and R.S. Nambimadom. “The Batch Loading and Scheduling Problem”. In: *Operations Research* 49.1 (Feb. 2001), pp. 52–65 (cit. on p. 13).
- [20] A. Druetto and A. Grosso. “Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine”. In: *Computers & Operations Research* 139 (Mar. 2022) (cit. on pp. 15, 82, 86, 87, 95, 97, 101).
- [21] A. Druetto and A. Grosso. “Polynomial-Size Models to Minimize Total Completion Time in a Parallel Batching Environment”. In: *IFAC-PapersOnLine* 55.10 (2022), pp. 2173–2178 (cit. on p. 15).
- [22] A. Druetto, E. Pastore, and E. Renner. “Parallel batching with multi-size jobs and incompatible job families”. In: *TOP* 31 (July 2023), pp. 440–458 (cit. on pp. 15, 97, 101).
- [23] J. Du and J.Y.-T. Leung. “Minimizing Total Tardiness on One Machine Is NP-Hard”. In: *Mathematics of Operations Research* 15.3 (Aug. 1990), pp. 483–495 (cit. on p. 13).
- [24] L. Dupont and C. Dhaenens-Flipo. “Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure”. In: *Computers & Operations Research* 29.7 (June 2002), pp. 807–819 (cit. on p. 10).
- [25] S. Emde, L. Polten, and M. Gendreau. “Logic-based benders decomposition for scheduling a batching machine”. In: *Computers & Operations Research* 113 (Jan. 2020) (cit. on p. 12).

- [26] B.Q. Fan, T.C.E. Cheng, S.S. Li, and Q. Feng. “Bounded parallel-batching scheduling with two competing agents”. In: *Journal of Scheduling* 16.3 (June 2013), pp. 261–271 (cit. on p. 14).
- [27] B.A. Foster and D.M. Ryan. “An Integer Programming Approach to the Vehicle Scheduling Problem”. In: *Operational Research Quarterly (1970-1977)* 27.2 (1976), pp. 367–384 (cit. on p. 36).
- [28] J.W. Fowler and L. Mönch. “A survey of scheduling with parallel batch (p-batch) processing”. In: *European Journal of Operational Research* 298.1 (Apr. 2022), pp. 1–24 (cit. on pp. 7, 12).
- [29] R. Fu, J. Tian, S. Li, and J. Yuan. “An optimal online algorithm for the parallel-batch scheduling with job processing time compatibilities”. In: *Journal of Combinatorial Optimization* 34.4 (Nov. 2017), pp. 1187–1197 (cit. on p. 11).
- [30] Michel Gondran, Michel Minoux, and Steven Vajda. *Graphs and Algorithms*. John Wiley & Sons, Inc., 1984 (cit. on p. 69).
- [31] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. In: *Discrete Optimization II*. Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 287–326 (cit. on pp. 8, 27, 57, 82).
- [32] M. Hulett, P. Damodaran, and M. Amouie. “Scheduling non-identical parallel batch processing machines to minimize total weighted tardiness using particle swarm optimization”. In: *Computers & Industrial Engineering* 113 (Nov. 2017), pp. 425–436 (cit. on p. 81).
- [33] Y. Ikura and M. Gimple. “Efficient scheduling algorithms for a single batch processing machine”. In: *Operations Research Letters* 5.2 (July 1986), pp. 61–65 (cit. on p. 7).
- [34] F. Jolai. “Minimizing number of tardy jobs on a batch processing machine with incompatible job families”. In: *European Journal of Operational Research* 162.1 (Apr. 2005), pp. 184–190 (cit. on p. 13).
- [35] F. Jolai Ghazvini and L. Dupont. “Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes”. In: *International Journal of Production Economics* 55.3 (Aug. 1998), pp. 273–280 (cit. on p. 12).
- [36] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Berlin, Heidelberg, 2004 (cit. on pp. 34, 85).
- [37] I. Kucukkoc. “MILP models to minimise makespan in additive manufacturing machine scheduling problems”. In: *Computers & Operations Research* 105 (May 2019), pp. 58–67 (cit. on p. 11).
- [38] B.J. Lageweg, J.K. Lenstra, E.L. Lawler, and A.H.G. Rinnooy Kan. “Computer-Aided complexity classification of combinatorial problems”. In: *Communications of the ACM* 25.11 (Nov. 1982), pp. 817–822 (cit. on pp. 9, 11).
- [39] E.L. Lawler. “A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness”. In: *Studies in Integer Programming*. Ed. by P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser. Vol. 1. Annals of Discrete Mathematics. Elsevier, 1977, pp. 331–342 (cit. on p. 13).

- [40] C.-Y. Lee, R. Uzsoy, and L.A. Martin-Vega. “Efficient Algorithms for Scheduling Semiconductor Burn-In Operations”. In: *Operations Research* 40.4 (Aug. 1992), pp. 764–775 (cit. on pp. 9, 11).
- [41] C.-L. Li and C.-Y. Lee. “Scheduling with agreeable release times and due dates on a batch processing machine”. In: *European Journal of Operational Research* 96.3 (Feb. 1997), pp. 564–569 (cit. on p. 11).
- [42] S. Li. “Approximation algorithms for scheduling jobs with release times and arbitrary sizes on batch machines with non-identical capacities”. In: *European Journal of Operational Research* 263.3 (Dec. 2017), pp. 815–826 (cit. on p. 10).
- [43] S. Li and J. Yuan. “Minimizing Total Tardiness on One Machine Is NP-Hard”. In: *Journal of Scheduling* 15 (Oct. 2012), pp. 629–640 (cit. on p. 96).
- [44] J.J. Liu, Z.T. Li, Q.X. Chen, and N. Mao. “Controlling delivery and energy performance of parallel batch processors in dynamic mould manufacturing”. In: *Computers & Operations Research* 66 (Feb. 2016), pp. 116–129 (cit. on pp. 3, 81).
- [45] A. Malapert, C. Gueret, and L.-M. Rousseau. “A constraint programming approach for a batch processing problem with non-identical job sizes”. In: *European Journal of Operational Research* 221.3 (Sept. 2012), pp. 533–545 (cit. on p. 12).
- [46] M. Mathirajan and A.I. Sivakumar. “A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor”. In: *The International Journal of Advanced Manufacturing Technology* 29.9 (Jan. 2006), pp. 990–1001 (cit. on p. 7).
- [47] M. Mathirajan, A.I. Sivakumar, and V. Chandru. “Scheduling algorithms for heterogeneous batch processors with incompatible job-families”. In: *Journal of Intelligent Manufacturing* 15.6 (Dec. 2004), pp. 787–803 (cit. on p. 14).
- [48] S. Melouk, P. Damodaran, and P.-Y. Chang. “Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing”. In: *International Journal of Production Economics* 87.2 (Jan. 2004), pp. 141–147 (cit. on pp. 10, 11).
- [49] L. Mönch, H. Balasubramanian, J.W. Fowler, and M.E. Pfund. “Minimizing Total Weighted Tardiness on Parallel Batch Process Machines Using Genetic Algorithms”. In: *Operations Research Proceedings 2002*. Ed. by U. Leopold-Wildburger, F. Rendl, and G. Wäscher. Springer Berlin, Heidelberg, 2003, pp. 229–234 (cit. on p. 14).
- [50] L. Mönch, J.W. Fowler, S. Dauzère-Pérès, S.J. Mason, and O. Rose. “A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations”. In: *Journal of Scheduling* 14.6 (Jan. 2011), pp. 583–599 (cit. on p. 7).
- [51] L. Mönch, J.W. Fowler, and S.J. Mason. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*. Vol. 52. Operations Research/Computer Science Interfaces Series. Springer Science & Business Media, 2012 (cit. on pp. 3, 81).
- [52] L. Mönch and R. Unbehaun. “Decomposition heuristics for minimizing earliness–tardiness on parallel burn-in ovens with a common due date”. In: *Computers & operations research* 34.11 (Nov. 2007), pp. 3380–3396 (cit. on p. 81).

- [53] M. Mourgaya and F. Vanderbeck. “Column generation based heuristic for tactical planning in multi-period vehicle routing”. In: *European Journal of Operational Research* 183.3 (Dec. 2007), pp. 1028–1041 (cit. on pp. 70, 101).
- [54] I. Muter. “Exact algorithms to minimize makespan on single and parallel batch processing machines”. In: *European Journal of Operational Research* 285.2 (Sept. 2020), pp. 470–483 (cit. on pp. 10, 11).
- [55] O. Ozturk. “A truncated column generation algorithm for the parallel batch scheduling problem to minimize total flow time”. In: *European Journal of Operational Research* 286.2 (Oct. 2020), pp. 432–443 (cit. on p. 13).
- [56] O. Ozturk, M.A. Begen, and G.S. Zaric. “A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan”. In: *International Journal of Production Research* 55.6 (Nov. 2016), pp. 1815–1831 (cit. on p. 11).
- [57] O. Ozturk, M.-L. Espinouse, M. Di Mascolo, and A. Gouin. “Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates”. In: *International Journal of Production Research* 50.20 (Dec. 2011), pp. 6022–6035 (cit. on pp. 3, 81).
- [58] I.C. Perez, J.W. Fowler, and W.M. Carlyle. “Minimizing total weighted tardiness on a single batch process machine with incompatible job families”. In: *Operations Research* 32.2 (Feb. 2005), pp. 327–341 (cit. on p. 13).
- [59] C.N. Potts and M.Y. Kovalyov. “Scheduling with batching: A review”. In: *European Journal of Operational Research* 120.2 (Jan. 2000), pp. 228–249 (cit. on pp. 7, 10, 12, 81).
- [60] N. Rafiee Parsa, B. Karimi, and A. Husseinzadeh Kashan. “A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes”. In: *Computers & Operations Research* 37.10 (Oct. 2010), pp. 1720–1730 (cit. on p. 10).
- [61] N. Rafiee Parsa, B. Karimi, and S.M. Moattar Hussein. “Minimizing total flow time on a batch processing machine using a hybrid max-min ant system”. In: *Computers & Industrial Engineering* 99 (Sept. 2016), pp. 372–381 (cit. on pp. 12, 27, 28, 43, 44, 47, 48).
- [62] N. Rafiee Parsa, T. Keshavarz, B. Karimi, and S.M. Moattar Hussein. “A hybrid neural network approach to minimize total completion time on a single batch processing machine”. In: *International Transactions in Operational Research* 28.5 (Apr. 2019), pp. 2867–2899 (cit. on p. 47).
- [63] B. Shahidi-Zadeh, R. Tavakkoli-Moghaddam, A. Taheri-Moghaddam, and I. Rastgar. “Solving a bi-objective unrelated parallel batch processing machines scheduling problem: A comparison study”. In: *Computers & Operations Research* 88 (Dec. 2017), pp. 71–90 (cit. on p. 14).
- [64] O. Shahvari and R. Logendran. “A bi-objective batch processing problem with dual-resources on unrelated-parallel machines”. In: *Applied Soft Computing* 61 (Dec. 2017), pp. 174–192 (cit. on p. 14).

- [65] T. Takamatsu, I. Hashimoto, and S. Hasebe. “Optimal scheduling and minimum storage tank capacities in a process system with parallel batch units”. In: *Computers & Chemical Engineering* 3.1–4 (1979), pp. 185–195 (cit. on p. 81).
- [66] Y. Tan, L. Mönch, and J.W. Fowler. “A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines”. In: *Journal of Scheduling* 21.2 (Apr. 2018), pp. 209–226 (cit. on p. 14).
- [67] R.S. Trindade, O.C.B. de Araújo, and M.H.C. Fampa. “Arc-flow approach for single batch-processing machine scheduling”. In: *Computers & Operations Research* 134 (Oct. 2021) (cit. on pp. 10, 97, 98).
- [68] R.S. Trindade, O.C.B. de Araújo, M.H.C. Fampa, and F.M. Müller. “Modelling and symmetry breaking in scheduling problems on batch processing machines”. In: *International Journal of Production Research* 56.22 (Nov. 2018), pp. 7031–7048 (cit. on p. 11).
- [69] R. Uzsoy. “Scheduling a single batch processing machine with non-identical job sizes”. In: *International Journal of Production Research* 32.7 (1994), pp. 1615–1635 (cit. on pp. 12, 37, 43, 44, 47, 50, 52, 102).
- [70] R. Uzsoy, C.-Y. Lee, and L.A. Martin-Vega. “A review of production planning and scheduling models in the semiconductor industry part II: shop-floor control”. In: *IEEE Transactions* 26.5 (Sept. 1994), pp. 44–55 (cit. on p. 10).
- [71] J. Zhang, X. Yao, and Y. Li. “Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing”. In: *International Journal of Production Research* 58.8 (Apr. 2020), pp. 2263–2282 (cit. on pp. 57, 81).
- [72] H. Zhou, J. Pang, P.-K. Chen, and F.-D. Chou. “A modified particle swarm optimization algorithm for a batch-processing machine scheduling problem with arbitrary release times and non-identical job sizes”. In: *Computers & Industrial Engineering* 123 (Sept. 2018), pp. 67–81 (cit. on p. 12).

Part II

- [73] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley & Sons, 1989 (cit. on p. 132).
- [74] A. Alexandrescu, I. Agavriloaei, and M. Craus. “A genetic algorithm for mapping tasks in heterogeneous computing systems”. In: *15th International Conference on System Theory, Control and Computing*. 2011, pp. 1–6 (cit. on p. 16).
- [75] *AURIX TriCore TC3xx Web Page*. <https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/> (cit. on p. 113).
- [76] *AUTOSAR Web Page*. <https://www.autosar.org/> (cit. on p. 114).
- [77] M. Becker, D. Dasari, B. Nolic, et al. “Contention-free execution of automotive applications on a clustered many-core platform”. In: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. 2016, pp. 14–24 (cit. on p. 17).
- [78] D. Bertsimas and J. Tsitsiklis. “Simulated Annealing”. In: *Statistical Science* 8.1 (Feb. 1993), pp. 10–15 (cit. on p. 132).

- [79] E. Bini, M. Di Natale, and G. Buttazzo. “Sensitivity analysis for fixed-priority real-time systems”. In: *Real-Time Systems* 39.1–3 (Apr. 2008), pp. 5–30 (cit. on p. 128).
- [80] R. Bouaziz, L. Lemarchand, F. Singhoff, B. Zalila, and M. Jmaiel. “Multi-objective design exploration approach for ravenstar real-time systems”. In: *Real-Time Systems* 54 (Feb. 2018), pp. 424–483 (cit. on p. 16).
- [81] D. Casini, P. Pazzaglia, A. Biondi, and M. Di Natale. “Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration”. In: *Journal of Systems Architecture* 124 (Mar. 2022) (cit. on p. 18).
- [82] J. Chen, P. Han, Y. Liu, and X. Du. “Scheduling independent tasks in cloud environment based on modified differential evolution”. In: *Concurrency and Computation: Practice and Experience* (Mar. 2021) (cit. on p. 16).
- [83] T.S. Craig. “Queuing spin lock algorithms to support timing predictability”. In: *1993 Proceedings Real-Time Systems Symposium* (Dec. 1993), pp. 148–157 (cit. on p. 129).
- [84] P. Cuadra, L. Krawczyk, R. Höttger, P. Heisig, and C. Wolff. “Automated scheduling for tightly-coupled embedded multi-core systems using hybrid genetic algorithms”. In: *International Conference on Information and Software Technologies*. 2017, pp. 362–373 (cit. on p. 16).
- [85] G.B. Dantzig. “Discrete-variable extremum problems”. In: *Operations Research* 5.2 (Apr. 1957), pp. 266–288 (cit. on p. 120).
- [86] R.I. Davis and A. Burns. “Robust priority assignment for fixed priority real-time systems”. In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. 2007, pp. 3–14 (cit. on p. 128).
- [87] A. Druetto, E. Bini, A. Grosso, et al. “Task and Memory Mapping of Large Size Embedded Applications over NUMA Architecture”. In: *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS)*. Association for Computing Machinery, 2023, pp. 166–176 (cit. on p. 20).
- [88] *Eclipse APP4MC Web Page*. <https://www.eclipse.org/app4mc/> (cit. on pp. 113, 114).
- [89] H.R. Faragardi, B. Lisper, K. Sandström, and T. Nolte. “An efficient scheduling of AUTOSAR runnables to minimize communication cost in multi-core systems”. In: *7th International Symposium on Telecommunications (IST’2014)*. 2014, pp. 41–48 (cit. on p. 16).
- [90] F. Fauberteau and S. Midonnet. “Robust Partitioned Scheduling for Static-Priority Real-Time Multiprocessor Systems with Shared Resources”. In: *18th International Conference on Real-Time and Network Systems*. 2010, pp. 217–225 (cit. on p. 15).
- [91] G. Fernandez, J. Abella, E. Quinones, et al. “Seeking time-composable partitions of tasks for cots multicore processors”. In: *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. 2015, pp. 208–217 (cit. on p. 18).
- [92] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. “Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.6 (May 2010), pp. 911–924 (cit. on p. 16).

- [93] S. Fürst, J. Mössinger, S. Bunzel, et al. “AUTOSAR – A Worldwide Standard is on the Road”. In: *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*. 2009 (cit. on pp. 3, 114).
- [94] M. Guan and T. Tong. “Ant colony algorithm based optimization method for real-time task scheduling of multi-core system”. Pat. CN105487920. 2016 (cit. on p. 19).
- [95] A. Hamann, D. Ziegenbein, S. Kramer, and M. Lukasiewicz. “Demonstration of the FMTV 2016 timing verification challenge”. In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2016 (cit. on p. 19).
- [96] W.E. Hart, J.-P. Watson, and D.L. Woodruff. “Pyomo: modeling and solving mathematical programs in Python”. In: *Mathematical Programming Computation* 3 (Aug. 2011), pp. 219–260 (cit. on p. 134).
- [97] F. Hebbache, F. Brandner, M. Jan, and L. Pautet. “Work-conserving dynamic time-division multiplexing for multi-criticality systems”. In: *Real-Time Systems* 56 (July 2020), pp. 124–170 (cit. on p. 19).
- [98] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. “Giotto: a time-triggered language for embedded programming”. In: *Lecture Notes in Computer Science* 2211 (Sept. 2001), pp. 84–99 (cit. on p. 17).
- [99] R. Höttinger, L. Krawczyk, and B. Igel. “Model-based automotive partitioning and mapping for embedded multicore systems”. In: *International Conference on Parallel, Distributed Systems and Software Engineering*. 2015 (cit. on p. 17).
- [100] Y. Kobayashi, K. Honda, S. Kojima, et al. “Mapping Method Usable with Clustered Many-core Platforms for Simulink Model”. In: *Journal of Information Processing* 30 (Feb. 2022), pp. 141–150 (cit. on p. 17).
- [101] S. Kramer, D. Ziegenbein, and A. Hamann. *Automotive application model based on APP4MC (WATERS17)*. <https://waters2017.inria.fr/> (cit. on pp. 114, 116, 121, 127, 130, 133).
- [102] C. Lameter. “NUMA (Non-Uniform Memory Access): An Overview”. In: *Queue* 11.7 (July 2013), pp. 40–51 (cit. on p. 4).
- [103] J.P. Lehoczky, L. Sha, and Y. Ding. “The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior”. In: *Proceedings of the 10th IEEE Real-Time Systems Symposium*. 1989, pp. 166–171 (cit. on p. 128).
- [104] L. Liping. “Central Processing Unit performance optimization method and device based on NUMA (Non-uniform Memory Access) architecture”. Pat. CN107346267. 2017 (cit. on p. 19).
- [105] C.L. Liu and J.W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. In: *Journal of the Association for Computing Machinery* 20.1 (Jan. 1973), pp. 46–61 (cit. on p. 124).
- [106] R. Lougee-Heimer. “The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community”. In: *IBM Journal of Research and Development* 47.1 (Jan. 2003), pp. 57–66 (cit. on p. 134).
- [107] M. Lowinski, D. Ziegenbein, and S. Glesner. “Splitting tasks for migrating real-time automotive applications to multi-core ecus”. In: *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*. 2016, pp. 1–8 (cit. on p. 18).

- [108] F. Lupp, S. Aldegheri, H.D. Patel, and N. Bombieri. “Task Mapping and Scheduling for OpenVX Applications on Heterogeneous Multi/Many-Core Architectures”. In: *IEEE Transactions on Computers* 70.8 (Feb. 2021), pp. 1148–1159 (cit. on p. 16).
- [109] G.M. Mancuso, E. Bini, and G. Pannocchia. “Optimal priority assignment to control tasks”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.5s (Oct. 2014), pp. 1–17 (cit. on p. 18).
- [110] M. Maspoli, M. Knauss, and M. Nowacki. “Method and device for operating a many-core system”. Pat. US2017090820. 2017 (cit. on p. 19).
- [111] S.D. McLean, S.S. Craciunas, E.A.J. Hansen, and P. Pop. “Mapping and Scheduling Automotive Applications on ADAS Platforms using Metaheuristics”. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2020, pp. 329–336 (cit. on p. 15).
- [112] R. Mirosanlou, D. Guo, M. Hassan, and R. Pellizzoni. “MCsim: An extensible dram memory controller simulator”. In: *IEEE Computer Architecture Letters* 19.2 (July 2020), pp. 105–109 (cit. on p. 19).
- [113] A.K. Mok and D. Chen. “A multiframe model for real-time tasks”. In: *IEEE Transactions on Software Engineering* 23.10 (Oct. 1997), pp. 635–645 (cit. on p. 128).
- [114] F. Murtagh and P. Contreras. “Methods of Hierarchical Clustering”. In: *International Encyclopedia of Statistical Science* (Dec. 2011), pp. 633–635 (cit. on p. 125).
- [115] S. Noriaki, E. Masato, and S. Junji. “Real time system task configuration optimization system for multi-core processors, and method and program”. Pat. US2012331474. 2012 (cit. on p. 19).
- [116] A.J. Page, T.M. Keane, and T.J. Naughton. “Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system”. In: *Journal of Parallel and Distributed Computing* 70.7 (July 2010), pp. 758–766 (cit. on p. 16).
- [117] R. Pathan, P. Voudouris, and P. Stenström. “Scheduling parallel real-time recurrent tasks on multicore platforms”. In: *IEEE Transactions on Parallel and Distributed Systems* 29.4 (Nov. 2017), pp. 915–928 (cit. on p. 16).
- [118] S. Paul, N. Chatterjee, P. Ghosal, and J.-P. Diguët. “Adaptive Task Allocation and Scheduling on NoC-based Multicore Platforms with Multitasking Processors”. In: *ACM Transactions on Embedded Computing Systems* 20.1 (Dec. 2020), pp. 1–26 (cit. on p. 18).
- [119] P. Pazzaglia, A. Biondi, and M. Di Natale. “Optimizing the functional deployment on multicore platforms with logical execution time”. In: *2019 IEEE Real-Time Systems Symposium (RTSS)*. 2019, pp. 207–219 (cit. on pp. 17, 124).
- [120] Q. Perret, P. Maurère, É. Noulard, et al. “Mapping hard real-time applications on many-core processors”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 2016, pp. 235–244 (cit. on p. 18).
- [121] J. Reineke, I. Liu, H.D. Patel, S. Kim, and E.A. Lee. “PRET DRAM controller: Bank privatization for predictability and temporal isolation”. In: *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. 2011, pp. 99–108 (cit. on p. 19).

- [122] S.E. Saidi, S. Cotard, K. Chaaban, and K. Marteil. “An ILP approach for mapping AUTOSAR runnables on multi-core architectures”. In: *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. 2015, pp. 1–8 (cit. on p. 18).
- [123] D. Senapati, A. Sarkar, and C. Karfa. “PRESTO: A Penalty-aware Real-time Scheduler for Task Graphs on Heterogeneous Platforms”. In: *IEEE Transactions on Computers* 71.2 (Feb. 2021), pp. 421–435 (cit. on p. 16).
- [124] M.H. Shirvani and R.N. Talouki. “A novel hybrid heuristic-based list scheduling algorithm in heterogeneous cloud computing environment for makespan optimization”. In: *Parallel Computing* 108 (Dec. 2021) (cit. on p. 17).
- [125] H. Takada and K. Sakamura. “Predictable spin lock algorithms with preemption”. In: *Proceedings of 11th IEEE Workshop on Real-Time Operating Systems and Software* (May 1994), pp. 2–6 (cit. on p. 129).
- [126] H. Topcuoglu, S. Hariri, and M.-Y. Wu. “Performance-effective and low-complexity task scheduling for heterogeneous computing”. In: *IEEE transactions on parallel and distributed systems* 13.3 (Mar. 2002), pp. 260–274 (cit. on p. 17).
- [127] S. Voronov, S. Tang, T. Amert, and J.H. Anderson. “AI meets real-time: Addressing real-world complexities in graph response-time analysis”. In: *2021 IEEE Real-Time Systems Symposium (RTSS)*. 2021, pp. 82–96 (cit. on p. 18).
- [128] W. Wang, S. Cotard, F. Gravez, and B. Miramond. “Optimizing application distribution on multi-core systems within AUTOSAR”. In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. 2016 (cit. on p. 18).
- [129] A. Wieder and B. Brandenburg. “On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks”. In: *Proceedings of the IEEE 34th Real-Time Systems Symposium*. 2013, pp. 45–56 (cit. on p. 129).
- [130] C. Wolff, L. Krawczyk, R. Höttger, et al. “AMALTHEA – Tailoring tools to projects in automotive software development”. In: *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. 2015, pp. 515–520 (cit. on pp. 16, 113, 114).
- [131] M. Yang, A. Wieder, and B. Brandenburg. “Global real-time semaphore protocols: A survey, unified analysis, and comparison”. In: *Proceedings of the IEEE 36th Real-Time Systems Symposium*. 2015, pp. 1–12 (cit. on p. 129).
- [132] Y. Zhao and H. Zeng. “The concept of unschedulability core for optimizing real-time systems with fixed-priority scheduling”. In: *IEEE Transactions on Computers* 68.6 (June 2018), pp. 926–938 (cit. on p. 18).

List of Publications

Publications related to the topics of the thesis

A. Alfieri, A. Druetto, A. Grosso and F. Salassa, *Column generation for minimizing total completion time on a single machine with parallel batching*. IFAC-PapersOnLine, vol 52, n 13, pages 969-974. Elsevier, 2019.

Basis for Chapter 4, Part I

A. Druetto and A. Grosso, *Column generation bounds on a network flow model to minimize the total weighted completion time for a single parallel batching machine*. 31st European Conference on Operational Research (EURO), July 2021.

Basis for Chapter 5, Part I

E. Renier, A. Druetto and E. Pastore, *Parallel batching with multi-size jobs*. 31st European Conference on Operational Research (EURO), July 2021.

Basis for Chapter 6, Part I

A. Alfieri, A. Druetto, A. Grosso and F. Salassa, *Column generation for minimizing total completion time in a parallel-batching environment*. Journal of Scheduling, vol 24, pages 569-588. Springer Nature, 2021.

Seminal paper of Chapter 4, Part I

A. Druetto and A. Grosso, *Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine*. Computers & Operations Research, vol 139. Elsevier, 2022.

Seminal paper of Chapter 5, Part I

A. Druetto and A. Grosso, *Polynomial-Size ILP formulations for the Total Completion Time Problem on a Parallel Batching Machine*. 15th Workshop on Models and Algorithms for Planning and Scheduling (MAPSP), June 2022.

Basis for Chapter 7, Part I

A. Druetto and A. Grosso, *Polynomial-Size Models to Minimize Total Completion Time in a Parallel Batching Environment*. IFAC-PapersOnLine, vol 55, n 10, pages 2173-2178. Elsevier, 2022.

Seminal paper of Chapter 7, Part I

A. Druetto, E. Pastore and E. Renier, *Parallel batching with multi-size jobs and incompatible job families*. TOP, vol 31, pages 440-458. Springer Nature, 2023.

Seminal paper of Chapter 6, Part I

A. Druetto, E. Bini, A. Grosso, S. Puri, S. Bacci, M. Di Natale and F. Paladino, *Task and Memory Mapping of Large Size Embedded Applications over NUMA architecture*. Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS), June 2023.
Patent Pending: *SOFTWARE OPTIMIZATION METHOD AND DEVICE FOR NUMA ARCHITECTURE*, International Patent Application PCT/EP2022/063829 (not published yet).

Seminal paper of Chapter 8, Part II

Other publications during the PhD period

A. Druetto, M. Roberti, R. Cancelliere, D. Cavagnino and M. Gai, *A Deep Learning Approach to Anomaly Detection in the Gaia Space Mission Data*. Lecture Notes in Computer Science, vol 11507, pages 390-401. Springer Nature, 2019.

M. Roberti, A. Druetto, D. Busonero, R. Cancelliere, D. Cavagnino and M. Gai, *Anomaly Detection Techniques in the Gaia Space Mission Data*. Journal of Signal Processing Systems, vol 93, pages 1339-1357. Springer Nature, 2021.

R. Aringhieri, S. Bigharaz, A. Druetto, D. Duma, A. Grosso and A. Guastalla, *The daily swab test collection problem*. Annals of Operations Research, In Press. Springer Nature, 2022.

List of Acronyms (Part I)

Notation	Description
B&B	Branch-and-Bound.
B&P	Branch-and-Price.
BB-UB	Branch-and Bound Upper Bound.
CG	Column Generation.
CG-LB	Column Generation Lower Bound.
CG-UB	Column Generation Upper Bound.
DP	Dynamic Programming.
ER-UB	Early Rounding Upper Bound.
LP	Linear Program.
LPT	Longest Processing Time.
MIP	Mixed-Integer Program.
P&B	Price-and-Branch.
PR	Parallel Relaxation.
RMP	Restricted Master Problem.
SPT	Shortest Processing Time.
VR-UB	Variable Rounding Upper Bound.

List of Acronyms (Part II)

Notation	Description
AUTOSAR	AUTomotive Open System ARchitecture.
CPU	Central Processing Unit.
DAG	Direct Acyclic Graph.
DRAM	Dynamic Random Access Memory.
GA	Genetic Algorithm.
GRAM	Global RAM.
HC	Hierarchical Clustering.
IP	Integer Program.
LRAM	Local RAM.
NUMA	Non-Uniform Memory Access.
OS	Operating System.
RPA	Robust Priority Assignment.
RTE	Run-Time Environment.
SA	Simulated Annealing.

List of Figures

4.1	Batch sequence as a path on a graph.	31
4.2	Example of a full graph.	31
4.3	Batch sequences on machines as a collection of paths on a graph.	39
5.1	Weighted batch sequence as a path on a graph.	60
8.1	AURIX TriCore TC39x: CPUs and memories (no I/O is reported).	112
8.2	Abstract hardware model.	115
8.3	The cyclic dependency of the mapping problem.	116
8.4	The mapping problem.	117
8.5	Binding of labels to runnables, then runnables to CPUs.	118
8.6	Binding labels to runnables.	119
8.7	Merging runnables in clusters.	124
8.8	Utilization as function of the allocated memory, with 40 clusters.	133
8.9	Slack of the mapping, with 40 clusters.	134
8.10	Utilization as function of the number of clusters.	135
8.11	Run-time of mapping of clusters.	136

List of Tables

2.1	Graham three-field notation.	8
3.1	Notation summary for Parallel Batching.	23
4.1	Results for CG-UB and CG-LB with $b = 10$	45
4.2	Results for CG-UB and CG-LB with $b = 30$	46
4.3	Results for CG-UB and CG-LB with $b = 50$	47
4.4	Comparison between HMMAS and CG-UB algorithms.	48
4.5	Comparison between CPLEX-UB and CG-UB.	49
4.6	Results for CG-UB and CG-LB with $b = 10$ and 2 parallel machines.	50
4.7	Results for CG-UB and CG-LB with $b = 10$ and 3 parallel machines.	51
4.8	Results for CG-UB and CG-LB with $b = 10$ and 5 parallel machines.	52
4.9	Comparison of exact approaches.	53
4.10	Comparison between CG-UB and real optima with $b = 10$	54
4.11	Results for CG-UB and CG-LB with $b = 50$ and $\sigma = \sigma_5$	55
5.1	Times and size for arc-based model vs path-based model.	73
5.2	Results for CG-LB and VR-UB on the weighted model.	75
5.3	Comparison between VR-UB and ER-UB on the weighted model.	76
5.4	Comparison between VR-UB and ER-UB over the Extended Set.	77
5.5	Comparison between VR-UB and ER-UB with different b	78
6.1	Results for the multi-size case.	89
6.2	Results for the incompatible families case.	90
6.3	Results for the multi-size (2) and incompatible families case.	91
6.4	Results for the multi-size (3) and incompatible families case.	92
6.5	Results for all cases over the extra instances.	94
7.1	Open nodes and optima found for the 7.1.2 model.	103
7.2	Results for the 7.1.2 model.	104
7.3	Open nodes and optima found for the 7.1.3 model.	104
7.4	Results for the 7.1.3 model.	105
7.5	Comparison between the 7.1.3 and the 4.1.2 lower bounds.	106
8.1	Key data of 2017 WATERS Challenge reference application.	116
8.2	Stall cycles for memory accesses in TC39x.	131
8.3	Run-time of binding, clustering, and priority assignment.	135

Colophon

This thesis was typeset with $\text{\LaTeX}2_{\epsilon}$. It uses a slightly adapted version of the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at <http://cleanthesis.der-ric.de/>.

