CONVEX CONIC RELAXATIONS OF THE SCHEDULING PROBLEM

by

Angela Liliana Montenegro Villota

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Mathematics
The University of Birmingham
January 2013

ABSTRACT

Scheduling has been used to describe a resources optimization problem since the early 1950's and plays an important role in domains such as manufacturing, transportation, distribution, construction, engineering, and management. A scheduling problem is defined by determining a production environment, job characteristics, scheduling constraints and an objective function which yields to different cases of the problem. As a result a variety of complexity arises among scheduling problems ranging from polynomially solvable to $\mathcal{NP}$-hard problems.

In this thesis two scheduling problem formulations are studied. The scheduling problem in a single machine environment, with equal-length jobs and release dates, with the objective function of minimizing total weighted tardiness $(1|p_j = p, r_j|\sum w_j T_j)$, whose complexity was still unknown, is shown to be solvable to optimality in polynomial time. The same is concluded for the related case with similar characteristics but minimizing weighted tardiness together with the maximum completion time, also known as the makespan.

The second problem studied occurs in a single machine environment, with release dates and interruption of the processing allowed, with the objective of minimizing weighted completion time $(1|r_j, pmtn|\sum w_j C_j)$. For this case three convex conic relaxations, based on semidefinite and linear programming, to use in conjunction with a customised branch and bound algorithm, are developed. The algorithm can be used to solve efficiently any size of the problem provided the maximum number of jobs at any time is 7.

ACKNOWLEDGEMENTS

Firstly I would like to express my thanks to my supervisor Professor Michal Kočvara. His advice, guidance and support have made the completion of this project possible.

A large debt of gratitude is also due to my family, especially to my mother, brothers and sisters, for their invaluable encouragement and support throughout my whole life.

I would like also thank my husband Tom with whom I share unforgettable moments, for being part of this exciting journey.

Finally, I would like to acknowledge the School of Mathematics of the University of Birmingham for their financial support throughout my studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOTATION AND TERMINOLOGY

Notation and terminology are introduced here as a reference only. Some of these will be introduced later as definitions when they are needed.

Sets

$A, B,$ etc	Sets
$\mathbb{R}$	Real numbers
$\mathbb{R}^n$	Real n-dimensional vector space
$\mathbb{R}^n_+$	Positive orthant of $\mathbb{R}^n$
$\mathbb{R}^{n\times n}$	Real $n \times n$ matrices
$\mathbb{C}$	Complex numbers
$\mathbb{F}$	Field or set of scalars ($\mathbb{R}$ or $\mathbb{C}$)
$\mathbb{Z}$	Integer numbers
$\mathbb{Z}_+$	Positive integer numbers
$\mathbb{S}^n$	$= \left\{ \mathsf{A} \mid \mathsf{A} \in \mathbb{R}^{n\times n},\, \mathsf{A} = \mathsf{A}^T \right\}$, symmetric matrices
$\mathbb{S}^n_+$	$= \left\{ \mathsf{A} \mid \mathsf{A} \in \mathbb{S}^n,\, \boldsymbol{x}^T\mathsf{A}\boldsymbol{x} \geqslant 0 \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \right\}$, positive semidefinite matrices
$\mathbb{S}^n_{++}$	$= \left\{ \mathsf{A} \mid \mathsf{A} \in \mathbb{S}^n,\, \boldsymbol{x}^T\mathsf{A}\boldsymbol{x} > 0 \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \right\}$, positive definite matrices
$\mathcal{A}$	Linear mapping: $\mathbb{R}^n \to \mathbb{S}^m$
$\mathcal{A}^*$	Adjoint of $\mathcal{A}$: Linear mapping $\mathbb{S}^m \to \mathbb{R}^n$
$\mathcal{F}$	Feasible set

Innequalities

$\geqslant$	$\geqslant_{\mathbb{R}^n_+}$
$>$	$>_{\mathbb{R}^n_+}$
$\succeq$	$\geqslant_{\mathbb{S}^m_+}$
$\succ$	$>_{\mathbb{S}^m_{++}}$

Vector notation

$\boldsymbol{v}$	$= (v_1, v_2, \ldots, v_n)$, vector with n entries
e_i	Zero vector with 1 in the ith entry
$\|\boldsymbol{v}\|$	$= \left(\|v_1\|^2 + \ldots + \|v_n\|^2\right)^{1/2}$, length of a vector $\boldsymbol{v}$ or Euclidean norm
$\| \cdot \|$	Vector norm
$\langle \boldsymbol{v}, \boldsymbol{w} \rangle$	$= v_1 w_1 + \ldots + v_n w_n$, inner product of vectors $\boldsymbol{v}$ and $\boldsymbol{w}$

Matrix notation

$\mathsf{A}, \mathsf{B}, \ldots$	Matrices
I	Identity matrix
$a_{ij}, (\mathsf{A})_{ij}$	Elements of the matrix A
A^{-1}	Inverse of the matrix A
A^T	Transpose of the matrix A
A^*	Conjugate transpose of the matrix A
A_k	Any $k \times k$ principal submatrix of A
$\det(\mathsf{A})$	Determinant of the matrix A
$\operatorname{rank}(\mathsf{A})$	Rank of the matrix A
$\operatorname{nullity}(\mathsf{A})$	Nullity of the matrix A
$\operatorname{tr}(\mathsf{A})$	Trace of the matrix A
$E_k(\mathsf{A})$	Sum of principal minors of the matrix A
λ_i	Eigenvalues of a matrix for $i = 1, \ldots, n$
$e_k(\lambda_1, \ldots, \lambda_n)$	Elementary symmetric polynomials in variables λ_i, for $i = 1, \ldots, n$
$\rho(\mathsf{A}), \rho_{\mathsf{A}}$	Spectral radius of A
$e(\mathsf{A})$	Number of nonzero eigenvalues of the matrix A
$\operatorname{diag}(x_1, \ldots, x_n)$	$n \times n$ matrix with entries $x_1, \ldots, x_n$ in the main diagonal
$\|\!\| \cdot \|\!\|$	Matrix norm
$\operatorname{svec}(\mathsf{A})$	$= (a_{11}, \ldots, a_{nn})$, matrix vectorisation operator
$\mathsf{A} \succeq \mathsf{B}$	$\mathsf{A} - \mathsf{B}$ is positive semidefinite
$\mathsf{A} \succ \mathsf{B}$	$\mathsf{A} - \mathsf{B}$ is positive definite

Convex programming

K	Proper cone: convex, pointed, closed, and with nonempty interior
K_*	Dual of a proper cone
L	Lorentz cone
CP	Primal convex programming problem

Production environments in scheduling

1	Single machine
S_m	m machines in series
P_m	m identical parallel machines
Q_m	m related (uniform) parallel machines
R_m	m unrelated parallel machines
F	Flow shop with m machines
J	Job shop with m machines
O	Open shop with m machines

Scheduling constraints

$chain$	Chain-like precedence constraints
d_j	Due date
$\hat{d}_j$	Deadline
$intree$	Intree-like precedence constraints
nbr	Number of jobs
$outtree$	Outtree-like precedence constraints
$p_j = p$	Equal processing times
$prec$	Precedence constraints
$pmtn$	Preemption
r_j	Release dates
$tree$	Tree-like precedence constraints

Objective functions in scheduling

C_{max}	Makespan
$\sum C_j$	Total completion time
$\sum w_j C_j$	Weighted completion time
L_{max}	Maximum lateness
$\sum w_j L_j$	Weighted lateness
T_{max}	Maximum tardiness
$\sum w_j T_j$	Weighted tardiness
$\sum w_j T_j^2$	Minimum weighted squared tardiness
E_{max}	Maximum earliness
$\sum w_j E_j$	Weighted earliness
$\sum U_j$	Unit penalty for tardy jobs
$\sum w_j U_j$	Weighted unit penalty for tardy jobs

PART I

PRELIMINARIES

CHAPTER 1

INTRODUCTION

1.1 About the project

Convex conic relaxations of the scheduling problem refers to the relaxations given for two selected scheduling problem formulations. The first is the scheduling problem in a single machine environment, with equal-length jobs and release dates, with the objective function of minimizing total weighted tardiness (i.e. $1|p_j = p, r_j| \sum w_j T_j$). A classification for this problem is provided.

The second scheduling problem studied occurs in a single machine environment, with release dates and interruption of the processing allowed, with the objective of minimizing weighted completion time ($1|r_j, pmtn| \sum w_j C_j$). For this case three relaxations, based on linear and semidefinite programming and an efficient customised branch and bound approximation algorithm using these relaxations are developed.

1.2 Methodology of the project

This document is divided in three parts. In Part I a general introduction is given. In Chapter 2 scheduling is defined as part of the production planning hierarchy. The concept scheduling as such is analysed in detail in Chapter 3, which also includes the characteristics of a scheduling problem, namely production environments, job characteristics, scheduling constraints and an objective function. Aspects about convex conic relaxations and the scheduling problem are discussed in Chapter 4.

In Part II the selected problem formulations are addressed. In Chapter 5 the scheduling problem $1|p_j = p, r_j|\sum w_j T_j$ is studied. Two cases are considered: the general case of the problem and the case of minimizing weighted tardiness together with the maximum completion time, also known as makespan. Initially, the problems are defined as integer programs with binary variables, in $\{0, 1\}$, then by allowing the variables to belong to the interval $[0, 1]$, linear programming relaxations are derived. For both cases the constraint matrix of the relaxations is shown to be totally unimodular and as such the optimal solutions are also in $\{0, 1\}$. Therefore, the linear programming relaxations for both cases solve the original integer programs to optimality. Since efficient polynomial-time algorithms for linear programming are known (see e.g. Wright (1997)), it is concluded that the scheduling problem $1|p_j = p, r_j|\sum w_j T_j$ and when minimizing makespan, can be solved to optimality in polynomial time.

In Chapter 6 the scheduling problem $1|r_j, pmtn|\sum w_j C_j$ is studied. The problem is formulated first as an integer program and then three relaxations, two linear programming based and a semidefinite programming based, are developed. Using these relaxations lower bounds for the optimal solution of the problem can be calculated. The obtained bounds are used in conjunction with a customised branch and bound algorithm, which by taking advantage of the properties of the problem, splits the problem into appropriate

subproblems for which enumeration trees are created. Each enumeration tree analyses only those nodes that are worth exploring. The algorithm provides solutions for this $\mathcal{NP}$-hard problem in an efficient time for any number of jobs provided the number of available jobs at a particular time is at most 7. Chapter 7 summarises the main conclusions of the project.

Part III comprises useful extra material included in the form of appendices. Appendix A includes matrix analysis theory involving eigenvalues and eigenvectors and also the characterisation of positive definite and semidefinite matrices. The Appendix B contains the basics and main aspects of conic programming theory.

CHAPTER 2

PRODUCTION PLANNING

2.1 Complexity

Industries are generally comprised of complex subsystems such as production, marketing, financial and others. They may also have several factories and depots and produce a wide variety of products for which they require people, machines, accessories and raw materials. Moreover, industries deal with decisions such as the construction of new plants, acquisition of new equipment, introduction of new products, preparation and reparation of machines.

One way of dealing with the complexity that arises from such subsystems is to develop production planning in a hierarchy. This reduces the global complexity of the problem, decomposing it into different levels with similar characteristics.

An efficient hierarchy planning focuses on balancing three contradicting objectives (Morton and Pentico (1993)): maximize throughput, which is the efficiency rate for a system to achieve its goal; maximize customer satisfaction and minimize operative costs and capital invested in the whole system.

To produce an appropriate solution with respect to these objectives, planning involves three key aspects (Baker (1974)):

1. Selection of products.

2. Quantification of products.

3. Specification of the required resources to produce the selected products.

2.2 Hierarchy planning

Hierarchy planning takes the above mentioned factors into consideration. Having made a decision in one level of the hierarchy, such a decision imposes physical restrictions for the next level in the hierarchy. The different levels of the hierarchy planning, according to Morton and Pentico (1993) are long-term planning, middle-term planning, short-term planning, scheduling, and reactive scheduling. See Figure 2.1.

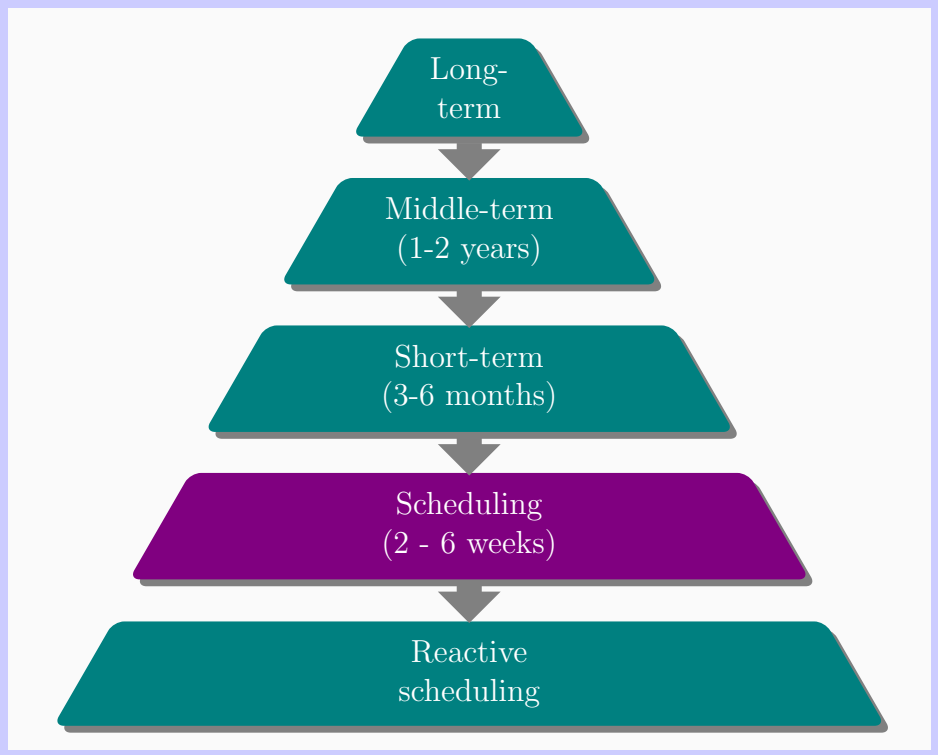


Figure 2.1: Hierarchy planning

Examples of *long-term planning* decisions are the design, building, layout, or expansion of plants, and warehouses.

Middle-term planning includes the reconfiguration of resources on a time scale of 1 to 2 years, when comparing the capacity of the plant versus the predetermined demand. The resources required in the next lower levels of planning are specified at this stage.

Among *short-term planning* decisions is the material requirements planning (MRP). In the MRP the amount of raw material which will be used in the subsequent 3 to 6 months is decided. MRP receives, as an input, the levels of production established in the middle-term planning and uses that information to estimate the resources that are required to satisfy such requirements.

Once the planning in the long, middle and short term has been done, the problem of *scheduling* can be solved. Scheduling is the production planning for the next 2 to 6 weeks. MRP provides specific information about jobs, such as the number of jobs, arrival times and due dates. A schedule is generated providing specific information about the activities to be performed.

Reactive Scheduling deals with emergencies that occur in a stochastic way but that require immediate attention, such as machine breakdowns and material failure.

CHAPTER 3

SCHEDULING PROBLEM

3.1 General remarks

3.1.1 The term *scheduling*

Scheduling takes place after the long, middle and short-term decisions in the hierarchy planning have been made. It establishes the specific tasks to be performed in a period of time from 2 to 6 weeks to reflect the decisions made in all the mentioned upper levels in the hierarchy. It is essentially a decision making process to determine a production plan considering the availability of resources and constraints. The latter may be physical constraints, like the number of machines for example; or constraints imposed by production policies, like the decision to offer a product in two different colours.

According to Baker (1974) there are three prevalent goals in the scheduling problem: the efficient utilisation of resources, prompt response to demand and conformity with deadlines.

Therefore, solving a scheduling problem implies making a decision about three variables: *assignment* of orders, equipment and personnel; *sequencing* of activities and estab-

lishment of the *starting and completion dates* of the operations.

Some scheduling problems are related with assignment only. In that case the complexity is reduced and optimal solutions are easily found, using Smith's rule (Smith (1956)) for example.

3.1.2 Basic definitions

In a production environment a given number of jobs need to be scheduled on a specified number of machines.

Definition 3.1.1. (Jobs) *Let n be the total number of activities, tasks, or products, identified as jobs, which need to be schedule in a production environment. The jobs are associated with a number j as the jth job for $j = 1 \dots n$.*

Definition 3.1.2. (Machines) *Let m be the number of machines that are available in a scheduling environment. The machines are associated with a number i to identify the ith machine for $i = 1 \dots m$.*

The information known a priori constitutes an input for planning and generally consists of the parameters defined in the following four definitions (Morton and Pentico (1993)).

Definition 3.1.3. (Processing time) *The processing time, denoted by p_{ij}, is the time required to process the job j on the machine i. If there is only one machine, or if the job requires the same amount of processing on all the machines, then the subscript i is omitted, i.e. p_j.*

Definition 3.1.4. (Release date) *The release date, denoted by r_{ij}, is the time at which the job j is available to be processed in the machine i. If there is only one machine, or if the job becomes available to all machines simultaneously then the subscript i is omitted, i.e. r_j.*

Definition 3.1.5. (Due date) *The due date, denoted by d_j, is the time by which the job j is expected to be completed.*

Definition 3.1.6. (Weight) *The weight, denoted by w_j, represents the importance of the job j. The weights are non-negative, i.e. $w_j \geqslant 0$, and sometimes the sum of the weights for all the jobs equals one, i.e.*

$$\sum_{j=1}^{n} w_j = 1. \tag{3.1}$$

Planning gives primary output measures, some of them are explained below (Morton and Pentico (1993) and Leung (2004)).

Definition 3.1.7. (Completion time) *The completion time, denoted by C_j, is the time at which the job j is completed.*

Definition 3.1.8. (Lateness) *Lateness, denoted by L_j, is the amount of time that the completion time of job j differs from the due date of the same job j,*

$$L_j = C_j - d_j. \tag{3.2}$$

Remark The value L_j can be positive or negative.

Definition 3.1.9. (Tardiness) *Tardiness, denoted by T_j, corresponds to the positive values of lateness. If the value of lateness for the job j is negative, then tardiness is equal to zero,*

$$T_j = \max \{ 0 , +L_j \} . \tag{3.3}$$

Definition 3.1.10. (Earliness) *Earliness, denoted by E_j, corresponds to the negative values of lateness. If the value of lateness for the job j is positive, then earliness is equal to zero,*

$$E_j = \max \{ 0 , -L_j \} . \tag{3.4}$$

Remark Let L_j, T_j, and E_j be the lateness, tardiness and earliness of the job j, for all $j = 1, \ldots n$, respectively. Then

$$L_j = T_j - E_j. \tag{3.5}$$

Definition 3.1.11. (Unit penalty for late completion) *The unit penalty, denoted by U_j, is the penalty of job j for a completion that exceeds its due date. And so $U_j = 1$ if and only if $C_j > d_j$ (i.e. $T_j \neq 0$) otherwise is equal to zero.*

3.2 Characteristics

In order to characterise scheduling problems, it is necessary to identify the production environment, the scheduling constraints, and the objective function.

3.2.1 Production environment

The production environment establishes the relation, and therefore the complexity, between the stages of the process and the production requirements of each item. Production environments differ in accordance with the unit of production (discrete or continue), products variety, volume, between others.

In order to analyse the behaviour of a production environment, it is important to understand how its components work. For this reason extensive research has been done around specific configurations of complex or real production environments. The importance of studying simple cases relies on identifying special characteristics to facilitate the study of more complex models. The commonly studied theoretical environments are both single machine and two machines. Real environments include flow shop, job shop and open shop.

Single machine (1)

In the environment with a single machine, n jobs need to be processed on a single machine configuration. Thus the order of the jobs needs to be specified. See Figure 3.1.

Series environment: n jobs - 2 machines in series

In this environment n jobs need to be processed by 2 machines sequentially. The notation used to describe this environment is S_2. In this case the order of the jobs in each of the

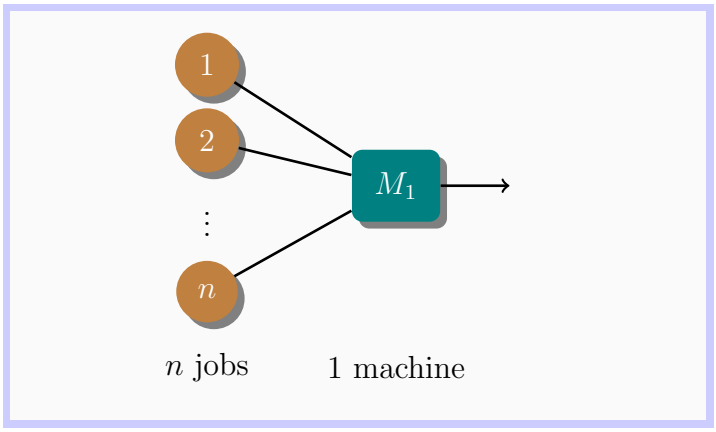


Figure 3.1: Single machine environment

machines will be given by the schedule. See Figure 3.2.

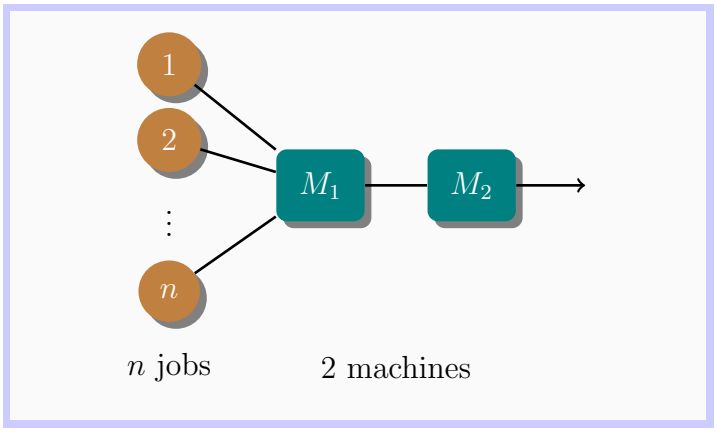


Figure 3.2: Two machines in series environment

Series environment: n jobs - m machines in series

In this configuration n jobs need to be processed on each of m machines in series, denoted as S_m. As in the previous case at each machine the order of the jobs has to be determined. See Figure 3.3.

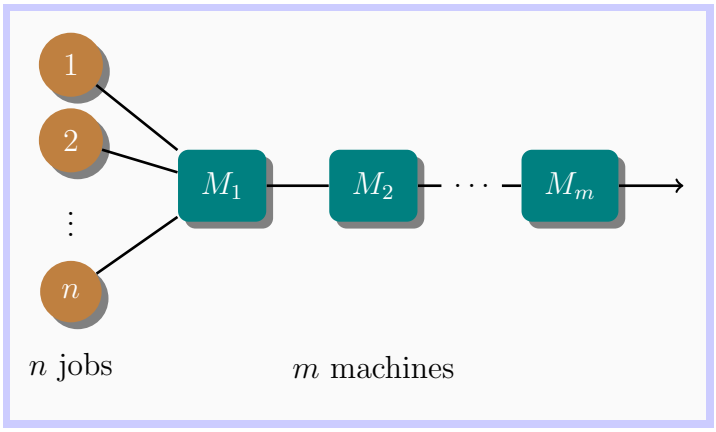


Figure 3.3: m machines in series environment

Parallel Machines: n jobs - m machines in parallel

In this environment n given jobs can be processed on any of the m available machines. See Figure 3.4. There are some variations of this environment in accordance with the relation of the machines. Namely P_m: m parallel identical machines, Q_m: m related parallel machines and R_m: m unrelated parallel machines.

Flow shop (F)

In a flow shop the production flow is linear. The jobs are processed following the same sequence of activities and in the same machines. In more complex flow shops a machine in the series can be replaced for a set of parallel machines. An example of flow shop is an assembly line. See Figure 3.5.

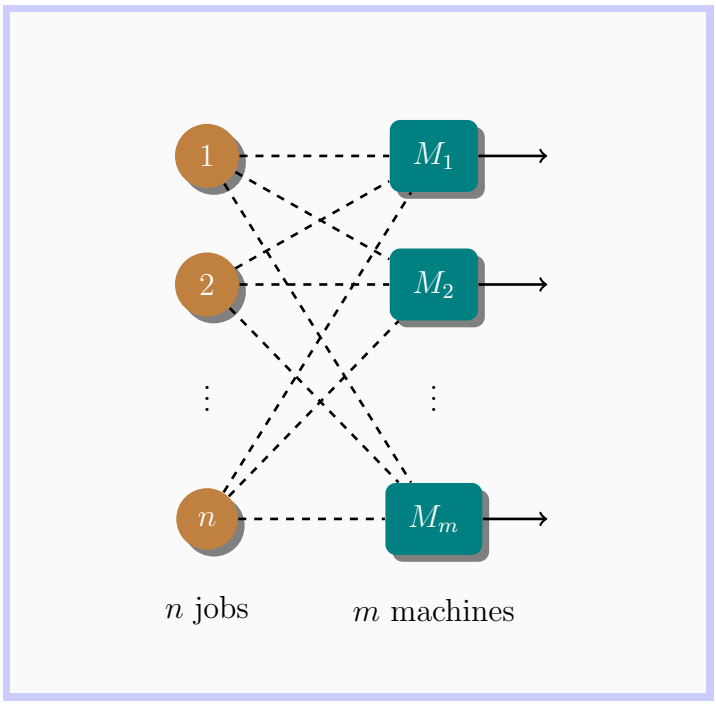


Figure 3.4: m machines in parallel environment

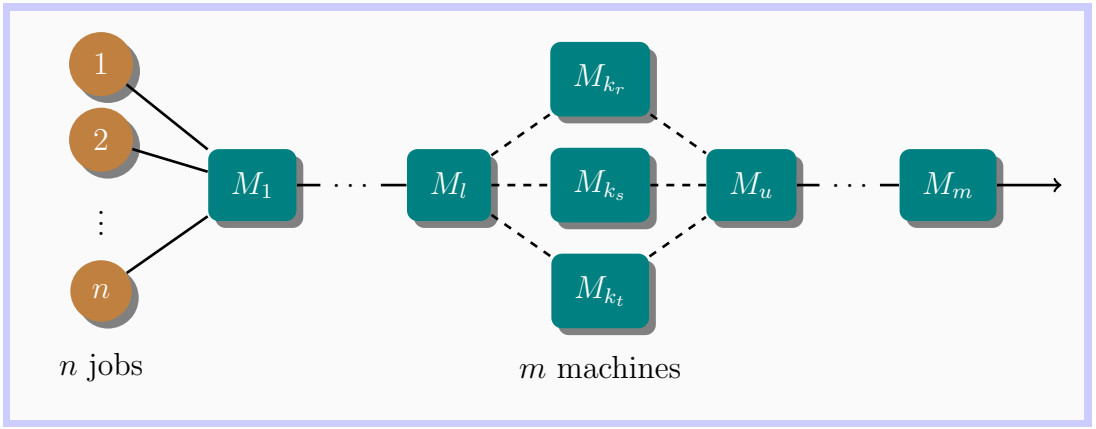


Figure 3.5: Flow shop environment

Job shop (J)

In this environment each job has a specific flow of production. Examples of job shop are fabrication tools shops, where each tool visits different machines according to the specific requirements of the job. A hammer and a screwdriver would be elaborated by different machines in the same shop, for instance.

Open shop (O)

In this environment, each job needs to visit every machine exactly once no matter the order. An example of an open shop can be painting parts of a product using different colours.

3.2.2 Scheduling constraints

Scheduling constraints are determined by the characteristics of the jobs and scheduling relations.

Preemption vs Nonpreemption

When jobs that are being processed on a machine can be interrupted to finish their processing later, then preemption is allowed.

Definition 3.2.1. (Preemption) *Preemption of the jobs $j \in J$, denoted by pmtn, indicates that the processing time p_j, for $j = 1, \ldots, n$, can be split into p_{j_r}, for $r = 1, \ldots, s$, such that*

$$p_j = \sum_{r=1}^{s} p_{j_r} \qquad j = 1, \ldots, n. \tag{3.6}$$

Release dates

Recall Definition 3.1.4 of release dates given in page 10. When the values r_j are given the schedule should consider release dates as an additional constraint.

Precedence constraints

Certain jobs may require the completion of other jobs before starting their own processing. Such relations are known as precedence constraints.

Definition 3.2.2. (Precedence constraints) *The set of all precedence constraints for a job j is represented by P_j, for $j = 1, \ldots, n$. Then, the release date of the job j is affected by the relation $r_j \geqslant C_i, \forall i \neq j \in P_j$.*

Precedence relations, denoted as *prec*, are represented in a precedence graph, where each node represents a job and each edge represent a relation between the jobs. When the precedence graph is a *tree*, a more specific problem can be defined. Such tree could be an *intree* in which the job with the no successors is called root. The jobs immediately preceding the root are in a level below and so on. On the other hand in an *outree* the

jobs with no successors are located in the first level. The next level is filled in with jobs that have successors in the levels immediately above. Additionally when the precedence graph is formed by nodes with disjoint paths then the precedence is called *chain*.

Number of jobs

When the maximum number of jobs allowed to be processed is restricted to a fixed quantity the number of jobs becomes a scheduling constraint.

Definition 3.2.3. (Number of jobs) *The number of jobs, denoted by n_j, specifies the maximum number of jobs that can be processed on a given environment.*

Processing times

When all the jobs require exactly the same processing time then this characteristic establishes a scheduling constraint.

Definition 3.2.4. (Equal processing times) *If given the jobs j the processing time is the same for all $j = 1, \ldots, n$, then this quantity is denoted then by p, thus $p_j = p$.*

Deadlines

When a deadline for a job cannot be negotiated, then such deadline can be added to the schedule as a constraint.

Definition 3.2.5. (Deadlines) *The deadline of the job j, denoted by $\hat{d}_j$, specifies the latest day by which the job should be finished.*

3.2.3 Objective function

The objective function represents the measure that will be used to evaluate the generated scheduling. There are two types of criteria that can be considered, cost and performance. The associated costs with a particular schedule include setup of machine costs, after working hour's costs, storage costs and costs for not having enough products to sell. Once a schedule has been generated performance criteria can be derived. Some of these performance criteria are explained below.

Completion time

The completion time indicates the maximum time at which the system changes from busy to idle. If a system can be left free as soon as possible then it will become available to process new jobs. Therefore these measures are generally minimised. There are several performance measures based on completion time. In this study makespan, total completion time, and total weighted completion time are explained.

Definition 3.2.6. (Makespan) *Makespan, denoted by C_{max}, is the maximum completion time among n jobs, i.e.*

$$C_{max} = \max_{1 \leqslant j \leqslant n} \{ C_j \} . \tag{3.7}$$

Remark The completion time, C_j, is relative to the release date, if the latter exists.

Definition 3.2.7. (Total completion time) *The total completion time is equal to the sum of the completion times of the n jobs, i.e.* $\sum_{j=1}^{n} C_j$*, where* C_j *is the completion time of* j*,* $j = 1, \ldots, n$*, in a particular schedule.*

The time that a job spends in the system should be relative to the importance or weight of each particular job. In this manner jobs with higher importance in criteria would have a priority to be finished before those jobs with lower importance. Specially considering criteria related with cost and customer service.

Definition 3.2.8. (Total weighted completion time) *The total weighted completion time is equal to the weighted sum of the completion time for n jobs, i.e.* $\sum_{j=1}^{n} w_j C_j$*, where* C_j *is the completion time of* j *and* w_j *is the weight of the job* j *for* $j = 1, \ldots, n$*.*

Lateness

Lateness gives information about delayed jobs, and so if the schedule is optimal under lateness generally produces efficient values in other objectives as well. Furthermore, the utilisation of lateness facilitates the calculus of other performance measures like flowtime for example.

Definition 3.2.9. (Maximum lateness) *The maximum lateness, denoted by L_{max}, is the maximum L_j for $j = 1, \ldots, n$,*

$$L_{max} = \max_{1 \leqslant j \leqslant n} \{ L_j \}. \tag{3.8}$$

Definition 3.2.10. (Weighted lateness) *The weighted lateness, denoted by L_{wt}, is the weighted sum of the lateness, for $j = 1, \ldots, n$, $\sum_{j=1}^{n} w_j L_j$.*

Tardiness

When a delay in finishing a job is acceptable, tardiness can be used as an objective function.

Definition 3.2.11. (Maximum tardiness) *The maximum tardiness, denoted by T_{max}, is the maximum T_j for $j = 1, \ldots, n$,*

$$T_{max} = \max_{1 \leqslant j \leqslant n} \{ T_j \}. \tag{3.9}$$

Weighted tardiness is an appropriate alternative when a job is accepted by a customer neither before nor after its due date.

Definition 3.2.12. (Weighted tardiness) *The weighted tardiness is the weighted sum of the tardiness for $j = 1, \ldots, n$, $\sum_{j=1}^{n} w_j T_j$.*

If both maximum tardiness and weighted tardiness are considered, then weighted square tardiness can also be calculated.

Definition 3.2.13. (Minimum weighted squared tardiness) *The minimum weighted square tardiness is the lowest possible value for the weighted square tardiness for* $j = 1, \ldots, n$, $\min_{1 \leqslant j \leqslant n} \{ w_j T_j^2 \}$.

Earliness

Earliness has applications in management techniques, such as just-in-time, that tend to control the costs for storing units that are finished before their due date.

Definition 3.2.14. (Maximum earliness) *The maximum earliness, denoted by* E_{max}*, is the maximum negative value of Lateness for* $j = 1, \ldots, n$,

$$E_{max} = \max_{1 \leqslant j \leqslant n} \{ E_j \}. \tag{3.10}$$

Definition 3.2.15. (Weighted earliness) *The weighted earliness is the weighted sum of* E_j *for* $j = 1, \ldots, n$, $\sum_{j=1}^{n} w_j E_j$.

Tardy jobs

The unit penalty for tardy jobs and the weighted unit penalty for tardy jobs are used when the customers cannot receive a job after its due date.

Definition 3.2.16. (Unit penalty for tardy jobs) *The total unit penalty for tardy jobs is* $\sum_{j=1}^{n} U_j$.

Definition 3.2.17. (Weighted unit penalty for tardy jobs) *The weighted number of tardy jobs, is the weighted sum of unit penalty for tardy jobs for* $j = 1, \ldots, n$, $\sum_{j=1}^{n} w_j U_j$.

3.3 Definition of a scheduling problem

As explained before, a scheduling problem is defined determining the production environment, job characteristics, scheduling constraints and an objective function. Using the notation introduced by Graham et al. (1979) a scheduling problem is represented by

$$\alpha \mid \beta \mid \gamma \tag{3.11}$$

where α represents the production environment, β gives job characteristics and details of scheduling constraints and γ is the objective function.

CHAPTER 4

CONVEX PROGRAMMING IN SCHEDULING

4.1 Complexity and scheduling problems

Scheduling problems are defined by a production environment, job characteristics and an objective function. Therefore an specific problem is defined by a particular selection of attributes in each of these aspects, which yields an unlimited number of problem types. Such diversity is reflected in a range of problems with different levels of complexity.

There are polynomially solvable scheduling problems, for which a solution exists for every instance of the problem. For example $1 \,|\, prec \,|\, f_{max}$, where $f_{max} \in \{C_{max}, L_{max}\}$, which can be solved in $\mathcal{O}(n^2)$ time (Lawler (1973)); the problem $1 \,|\, prec, pmtn, r_j \,|\, f_{max}$, also solvable in $\mathcal{O}(n^2)$ time (Baker et al. (1983)); the problem $1 \,|\, | \sum w_j C_j$, which can be solved in $\mathcal{O}(n \log n)$ (Smith (1956)) and the scheduling problem $P \,|\, intree, p_j = p \,|\, L_{max}$, also in $\mathcal{O}(n \log n)$ (Brucker et al. (1977)).

There are pseudo-polynomially solvable scheduling problems, whose solvable time depends on the numeric value of the data rather than the size of the problem itself. For instance $1||\sum w_jU_j$, which can be solved using dynamic programming in $\mathcal{O}(n\sum p_j)$ (Lawler and Moore (1969)) or in $\mathcal{O}(n\sum w_j)$ (Sahni (1976)) and the problem $1||\sum T_j$, which is solvable in $\mathcal{O}(n^4\sum p_j)$ (Lawler (1977)).

Weakly $\mathcal{NP}$-hard scheduling problems include those for which algorithms are exponential functions of the size of the problem. The problems $1||\sum w_jU_j$ (Karp (1972)), $1||\sum T_j$ (Du and Leung (1990)) and $1\,|\,d_j = d\,|\sum w_jT_j$ (Jinjiang (1992)), are examples of this group.

In the case of $\mathcal{NP}$-hard problems in the strong sense, efficient polynomial time exact algorithms are highly unlikely to exist. Examples of this type include the problem studied here, $1\,|\,r_j, pmtn\,|\sum w_jCj$ (Labetoulle et al. (1984)), also the problem $1||\sum w_jT_j$ (Lenstra et al. (1977)), the problem $P_2\,|\,tree\,|\,C_{max}$ (Du et al. (1991)) and $P\,|\,prec, p_j = 1\,|\,C_{max}$ (Ullman (1975)).

For $\mathcal{NP}$-hard problems the use of approximation algorithms is common, since they produce solutions that are guaranteed to be within a performance ratio α of the actual optimum. Thus solutions obtained by approximation algorithms are known to be feasible, perhaps non-optimal but they are still acceptable.

4.2 Approximation algorithms

4.2.1 Approximation algorithms for optimization problems

Convex optimization has played an important role in the development of approximation algorithms. The widely studied convex linear programming relaxation has shown to be an efficient tool for solving $\mathcal{NP}$-hard problems and specifically scheduling problems (see

e.g. Hall (1997) in which approximation algorithms for different scheduling problems are summarised). There are also *nonlinear* cases that have made effective use of more general conic programs, namely conic quadratic and semidefinite programming.

Semidefinite programming was successfully used by Lovász (1979) on the shannon capacity of a graph and by Gröetschel et al. (1981) who proposed a polynomial-time algorithm to find a maximum independent set in a perfect graph. Goemans and Williamson (1995) showed the power of semidefinite programming relaxations for $\mathcal{NP}$-hard problems with their work on the maximum cut problem, MAXCUT. Other important results have been obtained in graph colouring by Karger et al. (1994), in betweenness by Chor and Sudan (1998) and in graph bisection by Ye (2001).

4.2.2 Approximation algorithms in scheduling

Approximation algorithms using linear programming relaxations have been used for example to schedule unrelated machines, with preemption allowed and subject to release dates to minimize the total weighted completion, $R \,|\, r_j \,|\, \sum w_j C_j$ independently by Skutella (1998), and Sethuraman and Squillante (1999a,b).

The preemptive version of above scheduling problem, denoted as $R \,|\, r_j, pmtn \,|\, \sum w_j C_j$, with a convex quadratic relaxation and a 3-approximation algorithm is given by Skutella (1999). Skutella (2001) considers the nonpreemptive version of the latter $R \,|\, r_j \,|\, \sum w_j C_j$, with a 2-approximation algorithm based on semidefinite programming. A 3/2-approximation algorithm using a convex quadratic relaxation, for a similar problem in the absence of release dates, $R \,|\, | \,\sum w_j C_j$, is also given in Skutella (2001).

In Skutella (2001) the two machine case, $R2 \,|\, | \,\sum w_j C_j$ a semidefinite programming relaxation is also studied with a 1.2752-approximation algorithm.

PART II

SPECIFIC CASES

CHAPTER 5

$1 \,|\, p_j = p, r_j \,|\, \sum w_j T_j$

On a single machine n jobs need to be processed. The processing time p_j of any given job j is the same, i.e., $p_j = p$ for $j = 1, \ldots, n$, with $p > 0$ and $p \in \mathbb{Z}_+$. Once the processing of a job has started, it must be finished, that is preemption is not permitted. The jobs are available from a given time $r_j \geqslant 0$ with $r_j \in \mathbb{Z}_+$ for $j = 1, \ldots, n$. A weight $w_j \geqslant 0$ for all $j = 1, \ldots, n$ is given. The task is to find a schedule that minimizes the total weighted tardiness of the jobs $\sum w_j T_j$, with

$$T_j = \max\{0, C_j - d_j\},$$

where C_j denotes the completion time of a job j, which is found with the schedule, and $d_j > 0$ is the due date with $d_j \in \mathbb{Z}_+$ for $j = 1, \ldots, n$. Using the notation introduced by Graham et al. (1979), this problem may be written as $1|p_j = p, r_j| \sum w_j T_j$.[1]

[1]Paper submitted to the Journal of Scheduling, currently under review.

5.1 Complexity of the problem

Hitherto the complexity of this problem was unknown (Brucker and Knust (2006) and van den Akker et al. (2010)). In van den Akker et al. (2010) the problem is formulated as a time-indexed integer program. In addition, its linear programming relaxation is solved using a branch-and-bound algorithm, for which a solution is integral or it can be converted into one in polynomial time for some special cases of the problem, namely when

(i) all due dates, weights or release dates are equal, or

(ii) all due dates and release dates are equally ordered.

The solution of their linear programming relaxation is either integral or it can be adjusted in polynomial time into an integral one.

5.2 Formulation of the problem

5.2.1 Integer program

The length of the schedule, m, is the total time required to process all the jobs with $m \in \mathbb{Z}_+$ (since $p_j, r_j \in \mathbb{Z}_+$).

Preemption is not permitted, therefore in order to generate a schedule, it suffices to indicate the times at which each and every job is due to start. Thus n start processing times need to be determined. Since the problem occurs in a single machine environment, then such start times are required to give enough time to process each job without overlapping.

Let S denote the set of times at which the machine can start processing any of the jobs. In order to determine the set S note that for each job arriving a decision can be made of whether to process such job immediately or after another job. If there are jobs available at the time t, then t is a candidate for a start time and if such t is chosen as the

start time for one of the jobs the machine would then be unavailable for p units of time. Note that the set S contains only those times t that are worth exploring. These times are either a release time or a multiple of p with integer coefficients between 1 and m. Thus to determine S let R_l, $l \in \{0, \ldots, p-1\}$, be the set containing those possible times such that the operation t mod p has a common value l, for $0 \leqslant l \leqslant p-1$. The elements of each R_l are found as follows

(i) For each r_j, $j = 1, \ldots, n$, calculate $l_j = r_j \bmod p$.

(ii) Find the corresponding R_l such that $l_j = l$ and add the element r_j to R_l, $j = 1, \ldots, n$.

(iii) For each $j = 1, \ldots, n$ and for each nonempty set R_k, $k = 0, \ldots p-1$, with $k \neq l_j$, determine the minimum $t \geqslant r_j$ such that $t \bmod p = k$, $t \notin R_k$, and add that element t to R_k.

The set S can be constructed as

$$S = R_0 \cup R_1 \cup \ldots \cup R_{p-1}.$$

Example 5.2.1. *In a single machine environment $n = 2$ jobs need to be processed. The processing time is $p = 3$ units of time. Release date r_j, due date d_j and weight w_j for each job is given in the next table*

Job	Data		
	r_j	d_j	w_j
1	1	3	1
2	3	5	100

The sets R_l for Example 5.2.1 can be found as follows. Since $l_1 = 1$ and $l_2 = 0$, then

initially $R_0 = \{3\}$ and $R_1 = \{1\}$. For $j = 1$ the only nonempty R_l set with $l \neq l_1 = R_1$ is R_0 for which the minimum $t \geqslant r_1$, $t \notin R_0$ satisfying $t \bmod 3 = 0$ is 6. For $j = 2$ the only nonempty R_l set with $l \neq l_2 = R_0$ is R_1 and the minimum $t \notin R_1$ satisfying $t \bmod 3 = 1$ is 4, then

$$R_0 = \{3, 6\},$$
$$R_1 = \{1, 4\},$$
$$R_2 = \emptyset,$$

and so

$$S = \{1, 3, 4, 6\}.$$

The solution of example 5.2.1 is given in Section 5.5.

Variables: Let x_{ij} for $j = 1, \ldots, n$, with $i \in S$, for $i \geqslant r_j$ (otherwise the job is not available) be the binary variable indicating the start processing time for the jobs,

$$x_{ij} = \begin{cases} 1, & \text{if the processing of } j \text{ starts at } i; \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, let y_{ij}, for $j = 1, \ldots, n$, with $i \in S$ for $i \geqslant r_j$, be the binary variable indicating when a job is being processed,

$$y_{ij} = \begin{cases} 1, & \text{if job } j \text{ is being processed at } i; \\ 0, & \text{otherwise.} \end{cases}$$

Also, let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the row vectors containing all the variables x_{ij} and y_{ij} respectively,

$$\boldsymbol{x} = \big(x_{S_1 1}^T, \ldots, x_{S_j j}^T, \ldots, x_{S_n n}^T\big),$$
$$\boldsymbol{y} = \big(y_{S_1 1}^T, \ldots, y_{S_j j}^T, \ldots, y_{S_n n}^T\big),$$

where

$$S_j = S \cap \big\{i \geqslant r_j\big\}, \qquad j = 1, \ldots, n.$$

Objective function: Recall the objective function

$$\sum_{j=1}^{n} w_j T_j,$$

with

$$T_j = \max\big\{0, C_j - d_j\big\},$$

where C_j denotes the completion time of a job j in a given schedule. Thus, jobs that have yet to be processed after their due dates will increase the objective function.

For instance, for a job j at the time $i = d_j$, a check to see if the job has been processed and finished during the interval $1 \leqslant i \leqslant d_j$ has to be made. Observe that $t = k - p$ is the maximum possible start time t at which the job j could have been processed and finished by the time k, for $k = d_j + 1, \ldots, m$, after job j was released. Note that

$$1 - \sum_{\substack{i \in S_j \\ i \leqslant d_j - p}} x_{ij},$$

is zero if the job has been processed and finished at the time $d_j + 1$ (that is one of the variables $x_{ij} = 1$ in the interval $1 \leqslant i \leqslant d_j$ with $i \in S_j$) and is equal to 1 otherwise. The

objective function can then be written as

$$\sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S_j \\ i \leqslant k-p}} x_{ij} \right) \right).$$

Constraints: An interval of p consecutive units of time is referred here as a slot. There is only one machine, then independently of which times t are in S the sum of those x_{tj} that are part of a slot should not exceed one. In this way, initially we could say that the start time x_{tj} can be selected only if none of the x_{ij} with $t - p + 1 \leqslant i < t$ is selected at the same time, otherwise the one machine constraint would be violated.

Recall that variables x_{tj} indicate only the start time of a job. However the machine is busy for the whole time that is processing any of the jobs, which is not indicated by variables x_{tj} alone. Therefore variables y_{ij} are brought into consideration. In order to satisfy the one machine constraint, x_{tj} can only be one if and only if y_{tj} is also equal to one and, since preemption is not allowed, if a different job was being processed in the machine before, in any of the times belonging to the previous slot. The latter means that any of the y_{ij} with $t - p + 1 \leqslant i < t$ are equal to one. This condition holds unless the schedule has just started, that is $t = 1$.

None of the x_{ij} with $i \in [t - p + 1, t)$ must be equal to one if x_{tj} is equal to one. Thus considering only those i and t that belong to the same R_l the assumption reads as x_{ij} can only be selected if and only if x_{tj} with $t = i + p$ is not selected. Thus

$$\sum_{j=1}^{n} y_{tj} - \sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} x_{tj} = -1, \qquad t = i + p,$$
$$i, t \in R_l,\ l = 0, \ldots, p - 1. \tag{5.1}$$

If the processing of a job starts at x_{ij} then the variables y_{ij} are equal to one. Such a

constraint is modelled as

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} y_{ij} \leqslant 0, \qquad i \in S. \tag{5.2}$$

In order to avoid choosing the same start time for more than one job, the constraints

$$\sum_{j=1}^{n} x_{ij} \leqslant 1, \qquad i \in S, \tag{5.3}$$

and

$$\sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S, \tag{5.4}$$

must be added. Finally, the processing of all jobs is ensured with the constraint

$$\sum_{i \in S_j} x_{ij} = 1, \qquad j = 1, \ldots, n. \tag{5.5}$$

Note that constraint (5.4) follows from equations (5.2) and (5.5).

If $x_{1j} = 1$ for example, then $x_{1l} = 0$ for all $l \neq j$ is ensured by constraint (5.3). In constraint (5.2) one of the variables y_{1k} can be one. We want exactly the same y_{1j} belonging to our job with $x_{1j} = 1$.

Assume by contradiction that $y_{1l} = 1$, with $j \neq l$ for a different job to the one whose processing started at the time 1, with $x_{1j} = 1$. In that case constraint (5.2) is satisfied. However in equation (5.1), for R_1, a job that is being processed immediately after job j, will generate the following values for such equation $1 - 0 - 1 \neq -1$. Thus (5.1) is not satisfied which means that $y_{1j} = 1$.

The restriction in constraint (5.1) is applied to the following job which is going to be processed in the machine. For such new job to be processed any other job that has been previously processed need to be completed before allowing a new job to be processed.

This is enforced by constraint (5.1).

Note that when the schedule is starting is assumed that the machine is free before the time 1 and so no other job has to be considered when deciding if job j starts at one, $x_{1j} = 1$.

Recall Example 5.2.1, where $S = \{1, 3, 4, 6\}$. The set of constraints is

$$
\begin{aligned}
y_{61} + y_{62} - y_{31} - y_{32} - x_{61} - x_{62} &= -1, \\
y_{41} + y_{42} - y_{11} - x_{41} - x_{42} &= -1 \\
x_{31} + x_{32} &\leqslant y_{31} + y_{32}, \\
x_{61} + x_{62} &\leqslant y_{61} + y_{62}, \\
x_{11} &\leqslant y_{11}, \\
x_{41} + x_{42} &\leqslant y_{41} + y_{42}, \\
x_{31} + x_{32} &\leqslant 1, \\
x_{61} + x_{62} &\leqslant 1, \\
x_{11} &\leqslant 1, \\
x_{41} + x_{42} &\leqslant 1, \\
y_{31} + y_{32} &\leqslant 1, \\
y_{61} + y_{62} &\leqslant 1, \\
y_{11} &\leqslant 1, \\
y_{41} + y_{42} &\leqslant 1, \\
x_{11} + x_{31} + x_{41} + x_{61} &= 1, \\
x_{32} + x_{42} + x_{62} &= 1.
\end{aligned}
$$

Note that the number of constraints using the above definitions is

$$\sum_{l=0}^{p-1} \big(|R_l| - 1\big) + 3|S| + n,$$

where $|R_l|$ and $|S|$ are the number of elements of the sets R_l and S, respectively.

Remark The matrix of constraints is sparse.

The integer programming formulation for the problem is introduced next.

Problem 5.2.2. *The formulation of the scheduling problem* $1|p_j = p, r_j|\sum w_j T_j$ *as an integer program* $(\mathrm{IP}_{5.2.2})$ *is given by*

$$\begin{aligned}
\tau^{(\mathrm{IP}_{5.2.2})} = \min_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S_j \\ i \leqslant k-p}} x_{ij} \right) \right) \\
s.t. \quad & \sum_{j=1}^{n} y_{tj} - \sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} x_{tj} = -1, \quad t = i + p, \\
& \qquad i, t \in R_l,\ l = 0, \ldots, p-1, \\
& \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} y_{ij} \leqslant 0, \qquad i \in S, \\
& \sum_{j=1}^{n} x_{ij} \leqslant 1, \qquad i \in S, \\
& \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S, \\
& \sum_{i \in S_j} x_{ij} = 1, \qquad j = 1, \ldots, n, \\
& x_{ij}, y_{ij} \in \{0, 1\}, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

5.3 Relaxation of the problem

5.3.1 Linear programming relaxation

A relaxation of Problem 5.2.2 is given by replacing the non-convex constraints x_{ij} and $y_{ij} \in \{0,1\}$ by the convex relaxations $0 \leqslant x_{ij} \leqslant 1$ and $0 \leqslant y_{ij} \leqslant 1$, respectively.

Problem 5.3.1. *A relaxation for* Problem 5.2.2*, denoted* $\mathrm{LP}_{5.3.1\mathrm{r}}$*, is*

$$
\begin{aligned}
\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})} = \min_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S_j \\ i \leqslant k-p}} x_{ij} \right) \right) \\
s.t. \quad & \sum_{j=1}^{n} y_{tj} - \sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} x_{tj} = -1, \quad t = i + p, \\
& \qquad i, t \in R_l,\ l = 0, \ldots, p-1, \\
& \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} y_{ij} \leqslant 0, \qquad i \in S, \\
& \sum_{j=1}^{n} x_{ij} \leqslant 1, \qquad i \in S, \\
& \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S, \\
& \sum_{i \in S_j} x_{ij} = 1, \qquad j = 1, \ldots, n, \\
& 0 \leqslant x_{ij} \leqslant 1, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& 0 \leqslant y_{ij} \leqslant 1, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}
$$

In order to show that Problem 5.3.1 is in fact a relaxation let $\mathcal{F}^{(\mathrm{IP}_{5.2.2})}$ and $\mathcal{F}^{(\mathrm{LP}_{5.3.1\mathrm{r}})}$

denote the feasible sets of Problems 5.2.2 and 5.3.1, respectively.

Lemma 5.3.2. Problem 5.3.1 *is a relaxation of the scheduling problem* $1\,|\,p_j = p, r_j\,|\,\sum w_j T_j$ *given in* Problem 5.2.2, *i.e.*

(i) $\mathcal{F}^{(\mathrm{IP}_{5.2.2})} \subseteq \mathcal{F}^{(\mathrm{LP}_{5.3.1\mathrm{r}})}$.

(ii) $\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})} \leqslant \tau^{(\mathrm{IP}_{5.2.2})}$, *for all* x_{ij} *and* $y_{ij} \in \mathcal{F}^{(\mathrm{IP}_{5.2.2})}$.

Proof. *(i)* In Problem 5.2.2 variables x_{ij} and y_{ij} belong to $\{0, 1\}$ and in Problem 5.3.1 the same values for those variables are considered and some more, $0 \leqslant x_{ij} \leqslant 1$ and $0 \leqslant y_{ij} \leqslant 1$. Thus, it holds that $\mathcal{F}^{(\mathrm{IP}_{5.2.2})} \subseteq \mathcal{F}^{(\mathrm{LP}_{5.3.1\mathrm{r}})}$.

(ii) The objective function of both problems is the same and since $\mathcal{F}^{(\mathrm{IP}_{5.2.2})} \subseteq \mathcal{F}^{(\mathrm{LP}_{5.3.1\mathrm{r}})}$ then $\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})} \leqslant \tau^{(\mathrm{IP}_{5.2.2})}$ for all x_{ij} and $y_{ij} \in \mathcal{F}^{(\mathrm{IP}_{5.2.2})}$.

□

The following Theorem follows immediately.

Theorem 5.3.3. *The optimal value of* Problem 5.3.1 *is a lower bound for the optimal value in* Problem 5.2.2.

Proof. See Lemma 5.3.2 part *(ii)*.

□

5.4 Quality of the relaxation

In the linear programming relaxation given above the binary variables in $\{0, 1\}$ are changed to belong to $[0, 1]$. If a binary solution to the relaxation (with the constraint in $[0, 1]$) can be ensured, the solution is also optimal for the integer program. Let us see why this is the case and when it can take place.

Consider the polyhedron

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \mathsf{A}\boldsymbol{x} \geqslant \boldsymbol{b}, \boldsymbol{x} \geqslant 0\}, \tag{5.6}$$

where $\boldsymbol{b} \in \mathbb{R}^m$ is a vector and $\mathsf{A} \in \mathbb{R}^{m \times n}$ is a matrix of coefficients.

Let $\mathsf{A} = (\mathsf{B}\,\mathsf{N})$, where B is comprised of the linearly independent columns of A and N of the linearly dependent. A unique solution $\tilde{\boldsymbol{x}}$ of $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$, found as

$$\tilde{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_N \end{pmatrix} = \begin{pmatrix} \mathsf{B}^{-1}\boldsymbol{b} \\ 0 \end{pmatrix},$$

is a basic solution. If $\tilde{\boldsymbol{x}}$ is also feasible (i.e., it satisfies all the conditions in (5.6)) it is called a basic feasible solution. The set of basic feasible solutions coincide with the set of vertices or extreme points of the polyhedron described by the feasible set of (5.6). The polyhedron is called integral if all the vertices are in $\mathbb{Z}^n$ (see Schrijver (1986) chapter 19).

An integral polyhedron satisfies certain conditions that can be determined by analysing the properties of the matrix of coefficients. Here, the case when the matrix is totally unimodular is considered.

Definition 5.4.1. (Totally unimodular matrix) *Let* A *be an* $m \times n$ *real matrix. Then* A *is totally unimodular if every square submatrix of* A *has determinant equal to* $+1$, -1 *or* 0.

Some important facts related to totally unimodular matrices are explained in the following theorems.

Theorem 5.4.2. *Let* A *be an* $m \times n$ *real matrix. Then* A *is totally unimodular if and only if any of the following matrices are totally unimodular:*

(i) A^T,

(ii) (A, I),

(iii) $(\mathsf{A}, \boldsymbol{e}_i)$, *where* $\boldsymbol{e}_i$ *denotes a column of the identity matrix.*

Proof. *(i)* The matrices A and A^T have the same determinant and similarly the determinant of any square submatrix of A is the same as its corresponding transpose.

(ii) For a proof see Schrijver (1986) chapter 19.

(iii) The determinant of a submatrix of A formed by an intersection of the row or column $\boldsymbol{e}_i$ including the nonzero entry can be found by expansion of such row or column and is equal to the determinant of such submatrix. If the vector $\boldsymbol{e}_i$ is not intercepted by the square submatrix then its determinant remains the same. If the submatrix intercepts only the zero part of the row or column its determinant is equal to zero.

□

Definition 5.4.3. (Eulerian matrix) *Let* A *be an* $m \times n$ *real matrix with entries* $a_{ij} \in \{0, 1, -1\}$. *Then* A *is Eulerian if the sum of the elements per row and column is even.*

Theorem 5.4.4. *Let* $\mathsf{A} = (a_{ij})$ *be an* $m \times n$ *matrix such that* $a_{ij} \in \{0, 1, -1\}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. A *is totally unimodular if and only if for every Eulerian square submatrix it holds that the sum of the elements is a multiple of* 4.

Proof. Camion (1965). □

Checking the determinant of a matrix can be difficult when the matrix is large. The following Theorem provides sufficient conditions to verify that a matrix is totally unimodular.

Theorem 5.4.5. *Let* $\mathsf{A} = (a_{ij})$ *be an* $m \times n$ *matrix such that*

(i) The entries $a_{ij} \in \{0, 1, -1\}$*, for all* $i = 1, \ldots, m$ *and* $j = 1, \ldots, n$*.*

(ii) In each column j *there are at most two nonzero coefficients, i.e.,*

$$\sum_{i=1}^{m} |a_{ij}| \leqslant 2, \qquad j = 1, \ldots, n.$$

(iii) Let M *represent the set of rows. Then* M *can be partitioned into two sets, say* M_1 *and* M_2*, such that for each column* j *the following holds*

$$\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0, \qquad j = 1, \ldots, n.$$

Then A *is totally unimodular.*

Proof. Tamir (1976). □

A totally unimodular matrix is a sufficient criterion for the set of constraints of a linear program to have vertices in $\mathbb{Z}^n$ as shown in Theorem 5.4.6.

Theorem 5.4.6. *Let* A *be an* $m \times n$ *totally unimodular matrix and let* $\boldsymbol{b} \in \mathbb{Z}^m$*. Then*

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \mathsf{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geqslant 0\},$$

is an integral polyhedron.

Proof. Veinoot and Dantzig (1968).

□

The quality of the relaxation is now demonstrated.

Lemma 5.4.7. *The constraint matrix of* Problem 5.3.1 *is totally unimodular.*

Proof. Using slack variables $u_i^{(k)} \geqslant 0$, $k = 1, 2, 3$; $i \in S$ and $v_{ij}^{(l)} \geqslant 0$, $l = 1, 2$; $i \in S_j$, $j = 1, \ldots, n$, the inequality constraints of Problem 5.3.1 can be written as equalities. The feasible set is then equivalent to

$$\sum_{j=1}^{n} y_{tj} - \sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} x_{tj} = -1, \qquad t = i + p,$$

$$i, t \in R_l, l = 0, \ldots, p-1,$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} y_{ij} + u_i^{(1)} = 0, \qquad i \in S,$$

$$\sum_{j=1}^{n} x_{ij} + u_i^{(2)} = 1, \qquad i \in S,$$

$$\sum_{j=1}^{n} y_{ij} + u_i^{(3)} = 1, \qquad i \in S,$$

$$\sum_{i \in S_j} x_{ij} = 1, \qquad j = 1, \ldots, n,$$

$$x_{ij} + v_{ij}^{(1)} = 1, \qquad i \in S_j, \qquad j = 1, \ldots, n,$$

$$y_{ij} + v_{ij}^{(2)} = 1, \qquad i \in S_j, \qquad j = 1, \ldots, n,$$

$$
\begin{aligned}
&x_{ij}, y_{ij} \geqslant 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
&u_i^{(k)} \geqslant 0, \qquad k = 1, \ldots, 3, \qquad i \in S, \\
&v_{ij}^{(l)} \geqslant 0, \qquad l = 1, 2, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}
$$

After deleting a row or a column from a matrix, the number of possible square submatrices of said matrix is different, but the characteristics of the remaining ones, including the determinants, are not affected by this change. Therefore, without loss of generality suppose that $i \in S$ for $i = 1, \ldots, m$ and suppose that $S_j = S$, $j = 1, \ldots, n$. In such case the constraint matrix of Problem 5.3.1 has the form

$$
\begin{pmatrix}
\mathsf{A} & \mathsf{B} & \cdots & \mathsf{A} & \mathsf{B} & 0 & 0 & 0 & 0 \cdots 0 \\
-\mathsf{I} & \mathsf{I} & \cdots & -\mathsf{I} & \mathsf{I} & \mathsf{I} & 0 & 0 & 0 \cdots 0 \\
0 & \mathsf{I} & \cdots & 0 & \mathsf{I} & 0 & \mathsf{I} & 0 & 0 \cdots 0 \\
\mathsf{I} & 0 & \cdots & \mathsf{I} & 0 & 0 & 0 & \mathsf{I} & 0 \cdots 0 \\
0 & \mathsf{C}_1 & \cdots & 0 & \mathsf{C}_n & 0 & 0 & 0 & 0 \cdots 0 \\
\mathsf{I} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \mathsf{I} \cdots 0 \\
 & & \ddots & & & & & & \ddots \\
0 & 0 & & 0 & \mathsf{I} & 0 & 0 & 0 & 0 \cdots \mathsf{I}
\end{pmatrix}_{(2n+4)|S|+n-p \times (4n+3)|S|,}
\tag{5.7}
$$

where

$$
\mathsf{A} = \begin{pmatrix}
\mathsf{A}_1 & & 0 & \\
& \mathsf{A}_1 & & \\
& & \ddots & \\
0 & & & \mathsf{A}_1
\end{pmatrix}_{(|S|-p) \times |S|,}
$$

$$\mathsf{A}_1 = \begin{pmatrix} -1 & 1 & & & 0 \\ & & \ddots & & \\ 0 & & & -1 & 1 \end{pmatrix}_{(|R_l|-1)\times|R_l|},$$

$$\mathsf{B} = \begin{pmatrix} \mathsf{B}_1 & & 0 & \\ & \mathsf{B}_1 & & \\ & & \ddots & \\ 0 & & & \mathsf{B}_1 \end{pmatrix}_{(|S|-p)\times|S|},$$

$$\mathsf{B}_1 = \begin{pmatrix} 0 & -1 & & & 0 \\ & & -1 & & \\ & & & \ddots & \\ 0 & & & & -1 \end{pmatrix}_{(|R_l|-1)\times|R_l|},$$

and C_j, $j = 1, \ldots, n$, is the $n \times |S|$ zero matrix with all the elements in the ith row equal to one.

Observe that the following holds for matrix $\mathsf{A}_1^T = (a_{ij})$,

(i) The entries $a_{ij} \in \{0, 1, -1\}$ for all $i = 1, \ldots, |R_l|$ and $j = 1, \ldots, |R_l| - 1$.

(ii) In each column there are at most two nonzero coefficients, i.e.,

$$\sum_{i=1}^{|R_l|} |a_{ij}| \leqslant 2, \qquad j = 1, \ldots, |R_l| - 1.$$

(iii) Let $M_1 = \{1, 2, \ldots, |R_l|\}$ and $M_2 = \{\}$ then

$$\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0, \qquad j = 1, \ldots, |R_l| - 1.$$

Thus by Theorem 5.4.5 the matrix A_1^T is totally unimodular, and so it is A_1 by Theorem

5.4.2 part *(i)*. Note that the same properties mentioned above for A_1^T are true for the matrix $-\mathsf{A}^T$ and so $-\mathsf{A}$ is also totally unimodular. The matrix $-(\mathsf{A}\ \mathsf{B})$ is totally unimodular by Theorem 5.4.2 part *(iii)*. Thus, since $-|\mathsf{A}\ \mathsf{B}| = (-1)^n |\mathsf{A}\ \mathsf{B}|$ and the same applies for any square submatrix, then the determinant of $(\mathsf{A}\ \mathsf{B})$ and of any square submatrix is either $0, 1, -1$.

Also, if M_1 is the set containing all rows and M_2 is an empty set, then the matrix $(-\mathsf{I}\ \mathsf{I})^T$ is totally unimodular by Theorem 5.4.5 and its transpose by Theorem 5.4.2 part *(i)*.

Consider the matrix $\left(\begin{smallmatrix}\mathsf{A} & \mathsf{B}\\ -\mathsf{I} & \mathsf{I}\end{smallmatrix}\right)$. Observe that the only Eulerian matrices containing elements from both $(\mathsf{A}\ \mathsf{B})$ and $(-\mathsf{I}\ \mathsf{I})$ (since it has been shown that they are both totally unimodular individually) are any square zero matrices and those square matrices formed using the following combinations with rows and columns of zeros

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},$$
$$\begin{pmatrix} 1 & & & -1 & & 0 \\ & \ddots & & & \ddots & \\ & & 1 & & & -1 \\ -1 & & & 1 & & \\ & \ddots & & & \ddots & \\ 0 & & -1 & & & 1 \end{pmatrix}. \tag{5.8}$$

All the remaining entries of the latter matrix are equal to zero. Since the sum of the entries of all the matrices in (5.8) is a multiple of 4, then by Theorem 5.4.4 $\left(\begin{smallmatrix}\mathsf{A} & \mathsf{B}\\ -\mathsf{I} & \mathsf{I}\end{smallmatrix}\right)$ is totally unimodular.

The matrix

$$\begin{pmatrix} \mathsf{A} & \mathsf{B} \\ -\mathsf{I} & \mathsf{I} \\ \mathsf{0} & \mathsf{I} \\ \mathsf{I} & \mathsf{0} \end{pmatrix}, \tag{5.9}$$

is totally unimodular by Theorem 5.4.2 part *(iii)*. In the matrix

$$\begin{pmatrix} \mathsf{A} & \mathsf{B} & \cdots & \mathsf{A} & \mathsf{B} \\ -\mathsf{I} & \mathsf{I} & \cdots & -\mathsf{I} & \mathsf{I} \\ \mathsf{0} & \mathsf{I} & \cdots & \mathsf{0} & \mathsf{I} \\ \mathsf{I} & \mathsf{0} & \cdots & \mathsf{I} & \mathsf{0} \end{pmatrix}, \tag{5.10}$$

the same columns appear more than once. Thus, when selecting a square submatrix, two cases may occur. The square submatrix would either have identical columns, in which case the determinant is zero, or the square submatrix match one of the submatrices in (5.9) whose determinant is either $0, 1$ or -1. Now extend this matrix to

$$\begin{pmatrix} \mathsf{A} & \mathsf{B} & \cdots & \mathsf{A} & \mathsf{B} \\ -\mathsf{I} & \mathsf{I} & \cdots & -\mathsf{I} & \mathsf{I} \\ \mathsf{0} & \mathsf{I} & \cdots & \mathsf{0} & \mathsf{I} \\ \mathsf{I} & \mathsf{0} & \cdots & \mathsf{I} & \mathsf{0} \\ \mathsf{0} & \mathsf{C}_1 & \cdots & \mathsf{0} & \mathsf{C}_n \end{pmatrix}. \tag{5.11}$$

It is possible to form the following Eulerian matrices from both (5.10) and $(\mathsf{0}\ \mathsf{C}_1\ \ldots\ \mathsf{0}\ \mathsf{C}_n)$ (again since otherwise they are totally unimodular individually): square zero matrices and square matrices formed by using the following combinations with rows and columns of

zeros

$$\begin{pmatrix} \mathsf{I} & \cdots & \mathsf{I} \\ \mathsf{C}_i & \cdots & \mathsf{C}_k \end{pmatrix}, \qquad i, \ldots, k \in \{1, \ldots, n\}, \tag{5.12a}$$

$$\begin{pmatrix} -\mathsf{I} & \cdots & -\mathsf{I} \\ \mathsf{C}_i & \cdots & \mathsf{C}_k \end{pmatrix}, \qquad i, \ldots, k \in \{1, \ldots, n\}, \tag{5.12b}$$

$$\begin{pmatrix} -\mathsf{I} & \mathsf{I} & & -\mathsf{I} & \mathsf{I} \\ \mathsf{I} & 0 & \cdots & \mathsf{I} & 0 \\ 0 & \mathsf{C}_i & \cdots & 0 & \mathsf{C}_k \\ 0 & 0 & & 0 & 0 \end{pmatrix}, \qquad i, \ldots, k \in \{1, \ldots, n\}, \tag{5.12c}$$

$$\begin{pmatrix} -1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \tag{5.12d}$$

$$\begin{pmatrix} -1 & 0 & -1 & & -1 & 0 & -1 \\ -1 & 1 & 0 & \cdots & -1 & 1 & 0 \\ 0 & 1 & 1 & & & & \\ & & & \ddots & & & \\ & & & & 0 & 1 & 1 \\ 0 & & & \cdots & & & 0 \end{pmatrix}, \tag{5.12e}$$

$$\begin{pmatrix} -\mathsf{I} & & -\mathsf{I} \\ \mathsf{I} & \cdots & \mathsf{I} \\ \mathsf{I} & \cdots & \mathsf{I} \\ \mathsf{C}_i & & \mathsf{C}_k \end{pmatrix}, \qquad i, \ldots, k \in \{1, \ldots, n\}. \tag{5.12f}$$

The number of unit entries in the matrices C_j, $j = 1, \ldots, n$ in (5.12a) and (5.12b) must be the same even number, and there must be an even number of matrices C_j in either

case. In the matrices (5.12c) and in (5.12e) the rows of zeros are used to complete the square matrices, and the entries in each row must be selected so that their sum is an even number. Furthermore there must be an even number of matrices C_j. The sum of the entries in any matrix (5.12) is a multiple of 4, and by Theorem 5.4.4 the matrix in (5.11) is totally unimodular.

The matrix

$$\begin{pmatrix} \mathsf{A} & \mathsf{B} & \cdots & \mathsf{A} & \mathsf{B} \\ -\mathsf{I} & \mathsf{I} & \cdots & -\mathsf{I} & \mathsf{I} \\ 0 & \mathsf{I} & \cdots & 0 & \mathsf{I} \\ \mathsf{I} & 0 & \cdots & \mathsf{I} & 0 \\ 0 & \mathsf{C}_1 & \cdots & 0 & \mathsf{C}_n \\ \mathsf{I} & 0 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & & 0 & \mathsf{I} \end{pmatrix},$$

is totally unimodular by Theorem 5.4.2 part *(iii)*. Similarly, using the latter argument, it is concluded that the constraint matrix (5.7) of Problem 5.3.1 is totally unimodular by Theorem 5.4.2 part *(iii)*.

□

The characteristics of the constraint matrix of Problem 5.3.1 allow us to introduce the desired result.

Theorem 5.4.8. *The single machine scheduling problem* $1|p_j = p, r_j|\sum w_j T_j$ *is solvable in polynomial time to optimality.*

Proof. By Theorem 5.3.3

$$\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})} \leqslant \tau^{(\mathrm{IP}_{5.2.2})}.$$

Using Lemma 5.4.7 the constraint matrix of Problem 5.3.1 is totally unimodular and since

$$\boldsymbol{b} = \begin{pmatrix} -1 \\ \vdots \\ -1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

by Theorem 5.4.6 the feasible set of Problem 5.3.1 is integral. Provided that Problem 5.3.1 is feasible and bounded, $\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})}$ has an integer optimal value, and

$$\tau^{(\mathrm{LP}_{5.3.1\mathrm{r}})} = \tau^{(\mathrm{IP}_{5.2.2})}.$$

Problem 5.3.1 is a linear program and as such there exist efficient algorithms (see e.g. Wright (1997)), for solving all instances of the problem in polynomial time.

□

Remark If only one R_l set is nonempty then the problem can be simplified. This happens because the problem can be solved without using the variables y_{ij} which have been created to avoid overlapping jobs when more than one R_l exists. In this case the only constraints required are

$$\begin{aligned}
&\sum_{j=1}^{n} x_{ij} \leqslant 1, && i \in S,\\
&\sum_{i \in S_j} x_{ij} = 1, && j = 1, \ldots, n,\\
&x_{ij} \in \{0,1\}, && i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

The formulation for this particular case and its relaxation are given next.

Problem 5.4.9. *The formulation of the scheduling problem* $1|p_j = p, r_j|\sum w_j T_j$ *when there is only one set* R_l *which is nonempty, as an integer program* $(\mathrm{IP}_{5.4.9})$ *is*

$$\begin{aligned}
\tau^{(\mathrm{IP}_{5.4.9})} = \min_{\boldsymbol{x}} \quad & \sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S \\ i \leqslant k-p}} x_{ij} \right) \right)\\
s.t. \quad & \sum_{j=1}^{n} x_{ij} - 1 \leqslant 0, && i \in S,\\
& \sum_{i \in S} x_{ij} - 1 = 0, && j = 1, \ldots, n,\\
& x_{ij} \in \{0,1\}, && i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

Problem 5.4.10. *A linear programming relaxation, which is denoted by* $\mathrm{LP}_{5.4.10\mathrm{r}}$*, for* Problem 5.4.9 *is given by*

$$\begin{aligned}
\tau^{(\mathrm{LP}_{5.4.10\mathrm{r}})} = \min_{\boldsymbol{x}} \quad & \sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S \\ i \leqslant k-p}} x_{ij} \right) \right) \\
s.t. \quad & \sum_{j=1}^{n} x_{ij} - 1 \leqslant 0, \qquad i \in S, \\
& \sum_{i \in S} x_{ij} - 1 = 0, \qquad j = 1, \ldots, n, \\
& 0 \leqslant x_{ij} \leqslant 1, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

In the latter case, showing that the constraint matrix is totally unimodular is straightforward as indicated in the following.

Lemma 5.4.11. *The constraint matrix of* Problem 5.4.10 *is totally unimodular.*

Proof. The inequality constraints of Problem 5.4.10 can be written as equalities by adding

slack variables. The feasible set is then equivalent to

$$\begin{aligned}
\sum_{j=1}^{n} x_{ij} + u_i &= 1, && i \in S, \\
\sum_{i \in S} x_{ij} &= 1, && j = 1, \ldots, n, \\
x_{ij} + v_{ij} &= 1, && i \in S_j, && j = 1, \ldots, n, \\
x_{ij} &\geqslant 0, && i \in S_j, && j = 1, \ldots, n, \\
u_i &\geqslant 0, && i \in S, \\
v_{ij} &\geqslant 0, && i \in S_j, && j = 1, \ldots, n.
\end{aligned}$$

where u_i and v_{ij} are slack variables. Observe that if all jobs are release at the time $i = 1$, the coefficient matrix of the feasible set would be equivalent to

$$\begin{pmatrix}
 & \boldsymbol{e}_1 \ldots \boldsymbol{e}_n & \\
\mathsf{A} & & \\
 & 0 \ldots 0 & \\
\mathsf{I} & & \boldsymbol{e}_{2n+1} \ldots \boldsymbol{e}_{n^2+2n+1}
\end{pmatrix}, \tag{5.13}$$

where $\boldsymbol{e}_i$ denotes the zero vector with 1 in the ith component, I the identity matrix and

$$\mathsf{A} = \begin{pmatrix}
1 & & 0 & 1 & & 0 & 1 & & 0 \\
 & \ddots & & & \ddots & & & \ddots & \\
0 & & 1 & 0 & & 1 & 0 & & 1 \\
1 & \cdots & 1 & & & & & & 0 \\
 & & & & \ddots & & & & \\
0 & & & & & & 1 & \cdots & 1
\end{pmatrix}_{2n \times n^2}. \tag{5.14}$$

If a job has been released at a different time, then all the columns of A belonging to those previous times i such that $i < r_j$ for $j = 1, \ldots, n$, are not part of the matrix, and the number columns of A is less than or equal to n^2. Thus, no matter which columns of A do not appear in the constraint matrix of Problem 5.4.10, the conditions of Theorem 5.4.5 with $M_1 = \{1, 2, \ldots, |S|\}$ and $M_2 = \{|S|+1, \ldots, |S|+n\}$ are satisfied and A, in equation (5.14), is totally unimodular. By Theorem 5.4.2 part *(ii)* the matrix

$$\begin{pmatrix} \mathsf{A} \\ \mathsf{I} \end{pmatrix}$$

is totally unimodular. Finally (5.13) is also totally unimodular by Theorem 5.4.2 part *(iii)*.

□

A similar problem as the one defined in Problem 5.4.10 coincide with the case of minimizing makespan as well as weighted tardiness, which is addressed in the following section.

5.5 Minimizing makespan and weighted tardiness

In the general case of minimizing total weighted tardiness the schedule is optimal but the makespan is not necessary minimum as occurs in Example 5.2.1.

Two solutions schedules for Example 5.2.1 are given in Figure 5.1. The optimal solution (schedule (a)) has cost 5 but when minimizing makespan as well (schedule (b)) the cost of the schedule is 100.

In the case of minimizing makespan together with total weighted tardiness, as soon as

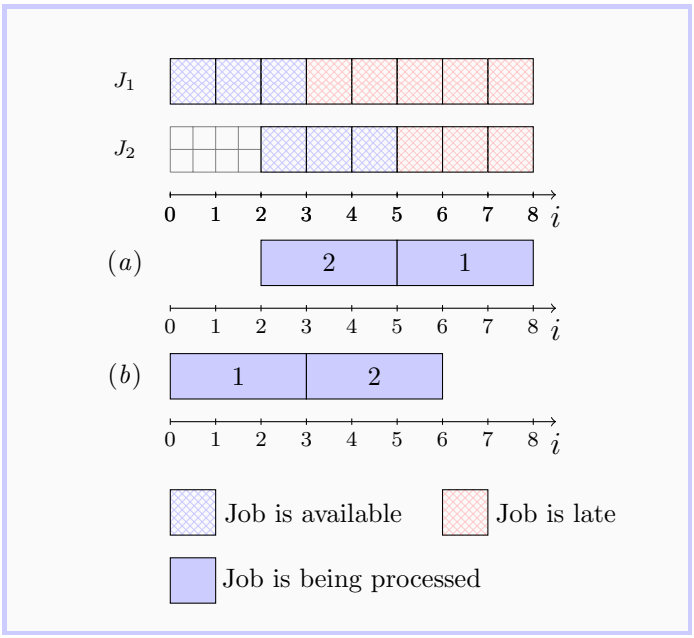


Figure 5.1: Schedules for Example 5.2.1 with values (a) 5 and (b) 100

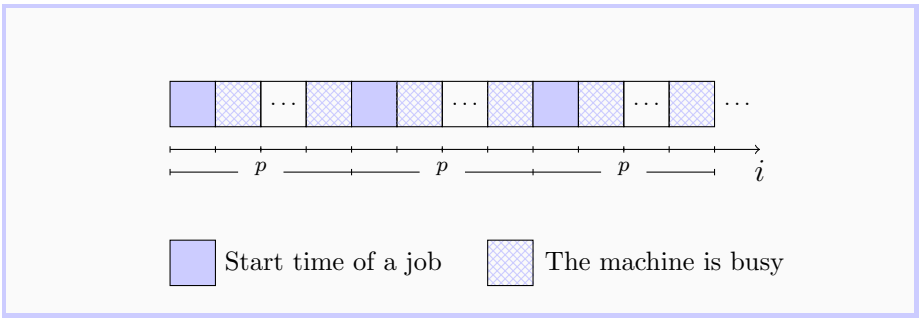
a job is available the machine will start processing one of the available jobs at the time i, and will stay unavailable during the interval $[i, i+1, \ldots, i+p-1]$. Note that there are two possible sub-cases

(a) If there are jobs available at all the times then the machine is never idle, as shown in Figure 5.2a and $m = np$.

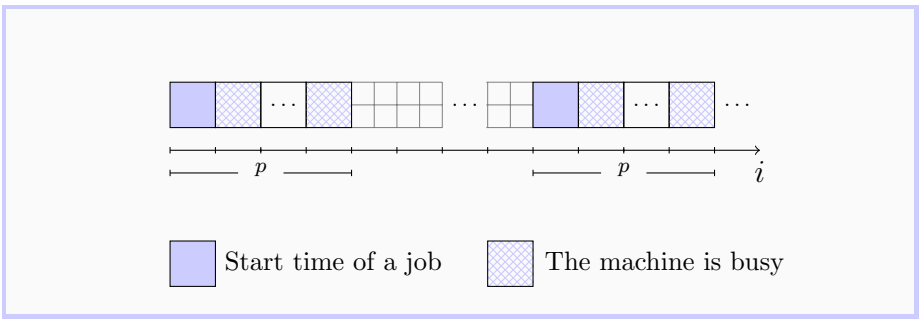
(b) If there are no jobs available when the machine has finished processing a job then it stays idle in accordance with such availability, as indicated in Figure 5.2b.

The set S in this case contains those times among m fulfilling two conditions: job availability and machine availability. Thus, without loss of generality let $\hat{r}_1 \leqslant \hat{r}_2 \leqslant \ldots \leqslant \hat{r}_n$ be the release dates in ascending order. Also, let

$$s_1 = \hat{r}_1 \tag{5.15}$$



(a) Jobs are available when the machine is free



(b) Jobs are not available at all times

Figure 5.2: Busy versus idle time of the machine according to availability of the jobs

and for $l = 2, \ldots, n$

$$s_l = \max\{s_{l-1} + p, \hat{r}_l\}. \tag{5.16}$$

Thus

$$m = s_n + p - 1.$$

Then S is

$$S = \{i = 1, \ldots, m \,|\, i = s_l, l = 1, \ldots, n\},$$

with s_l as in (5.15) and (5.16).

Variables: Let x_{ij} for $j = 1, \ldots, n$, with $i \in S$, for $i \geqslant r_j$ (otherwise the job is not available) be the binary variable indicating the start processing time for the jobs,

$$x_{ij} = \begin{cases} 1, & \text{if the processing of } j \text{ starts at } i; \\ 0, & \text{otherwise.} \end{cases}$$

Also, let $\boldsymbol{x}$ denote the row vector containing all the variables x_{ij},

$$\boldsymbol{x} = \left(x_{S_1 1}^T, \ldots, x_{S_j j}^T, \ldots, x_{S_n n}^T\right)^T,$$

where

$$S_j = S \cap \left\{i \geqslant r_j\right\}, \qquad j = 1, \ldots, n.$$

Objective function: The objective function

$$\sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S_j \\ i \leqslant k-p}} x_{ij} \right) \right),$$

remains the same.

Constraints: Once the machine is processing a job it cannot handle another. That is, once a time slot in S has been assigned for a job it cannot be associated with another for the next p units of time. In this case, for a minimal m the constraint is

$$\sum_{j=1}^{n} x_{ij} \leqslant 1, \qquad i \in S.$$

All jobs need to be processed, thus the constraint

$$\sum_{i \in S} x_{ij} = 1, \qquad j = 1, \ldots, n,$$

is added.

The integer programming formulation is given by the following.

Problem 5.5.1. *The formulation of the scheduling problem* $1|p_j = p, r_j|\sum w_j T_j$ *when the makespan is also minimum, as an integer program* $(\mathrm{IP}_{5.5.1})$ *is*

$$\begin{aligned}
\tau^{(\mathrm{IP}_{5.5.1})} = \min_{\boldsymbol{x}} \quad & \sum_{j=1}^{n} \sum_{k=d_j+1}^{m} \left(w_j \left(1 - \sum_{\substack{i \in S \\ i \leqslant k-p}} x_{ij} \right) \right) \\
s.t. \quad & \sum_{j=1}^{n} x_{ij} - 1 \leqslant 0, \qquad i \in S, \\
& \sum_{i \in S} x_{ij} - 1 = 0, \qquad j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\}, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

Note that the latter formulation coincide with the one given in Problem 5.4.9 whose relaxations have been given in Problem 5.4.10. Similar results as those introduced for the general case follow below.

Lemma 5.5.2. Problem 5.4.10 *is a relaxation for the scheduling problem* $1|p_j = p, r_j|T_{wt}$ *when the makespan is minimum, given in* Problem 5.5.1*, i.e.,*

(i) $\mathcal{F}^{(\mathrm{IP}_{5.5.1})} \subseteq \mathcal{F}^{(\mathrm{LP}_{5.4.10\mathrm{r}})}$.

(ii) $\tau^{(\mathrm{LP}_{5.4.10\mathrm{r}})} \leqslant \tau^{(\mathrm{IP}_{5.5.1})}$, *for all* $x_{ij} \in \mathcal{F}^{(\mathrm{IP}_{5.5.1})}$.

Proof. This proof can be done using the same ideas as Lemma 5.3.2.

□

Theorem 5.5.3. *The optimal value of* Problem 5.4.10 *is a lower bound for the optimal value in* Problem 5.5.1.

Proof. See Lemma 5.5.2 part *(ii)*.

□

It was shown in Lemma 5.4.11 that the constraint matrix of Problem 5.4.10 is totally unimodular, so the analysis is concluded with the introduction of the following.

Theorem 5.5.4. *The single machine scheduling problem* $1|p_j = p, r_j| \sum w_j T_j$ *when the makespan is minimum is solvable in polynomial time to optimality.*

Proof. Using Theorem 5.5.3

$$\tau^{(\mathrm{LP}_{5.4.10\mathrm{r}})} \leqslant \tau^{(\mathrm{IP}_{5.5.1})}.$$

However by Theorem 5.4.6, with $\boldsymbol{b} = \big(1, \cdots, 1\big)^T$, the set of basic feasible solutions of $\tau^{(\mathrm{LP}_{5.4.10\mathrm{r}})}$ is integral. Then provided that Problem 5.4.10 is feasible and bounded (a solution exists) such solution has an integer optimal value. Thus

$$\tau^{(\mathrm{LP}_{5.4.10\mathrm{r}})} = \tau^{(\mathrm{IP}_{5.5.1})}.$$

In other words, the linear programming relaxation in Problem 5.4.10 solves the integer program Problem 5.5.1 to optimality. Problem 5.4.10 can be solved in polynomial time (see Wright (1997)) for all instances of the problem.

□

5.6 Numerical solution of the problem

An example of the formulation of the problem is included here.

Example 5.6.1. *In a single machine environment $n = 5$ jobs need to be processed. The processing time is $p = 3$ units of time. Release date r_j, due date d_j and weight w_j for the jobs are given in the next table*

Job	Data		
	r_j	d_j	w_j
1	1	9	1
2	3	10	3
3	6	11	4
4	7	11	5
5	7	11	2

If the problem consist of minimizing makespan as well as tardiness (Problem 5.4.10) the set S is

$$S = \{1, 4, 7, 10, 13\},$$

and $m = 15$.

The problem is

$$
\begin{aligned}
\min_{x_{ij}\in\{0,1\}} \quad & 1\big((1 - x_{11} + x_{41}) + 3(1 - x_{11} + x_{41} + x_{71}) + 3(1 - x_{11} + x_{41} + x_{71} + x_{101})\big) + \\
& 3\big(3(1 - x_{42} + x_{72}) + 3(1 - x_{42} + x_{72} + x_{102}))\big) + \\
& 4\big(2(1 - x_{73}) + 3(1 - x_{73} + x_{103})\big) + \\
& 5\big(2(1 - x_{74}) + 3(1 - x_{74} + x_{104})\big) + \\
& 2\big(2(1 - x_{75}) + 3(1 - x_{75} + x_{105})\big) \\
s.t. \quad & x_{11} \leqslant 1, \\
& x_{41} + x_{42} \leqslant 1, \\
& x_{71} + x_{72} + x_{73}x_{74} + x_{75} \leqslant 1, \\
& x_{101} + x_{102} + x_{103}x_{104} + x_{105} \leqslant 1, \\
& x_{131} + x_{132} + x_{133}x_{134} + x_{135} \leqslant 1, \\
& x_{11} + x_{41} + x_{71} + x_{101} + x_{131} - 1 = 0, \\
& x_{42} + x_{72} + x_{102} + x_{132} - 1 = 0, \\
& x_{73} + x_{103} + x_{133} - 1 = 0, \\
& x_{74} + x_{104} + x_{134} - 1 = 0, \\
& x_{75} + x_{105} + x_{135} - 1 = 0, \\
& 0 \leqslant x_{ij} \leqslant 1,\ i \in S, j = 1, \ldots, n.
\end{aligned}
\tag{5.17}
$$

In the case of minimizing total weighted tardiness only (Problem 5.3.1), the formulation is similar to the one given in (5.17) but the sets R_l, $l = 0, \ldots, p-1$ are

$$R_0 = \{3, 6, 9, 12, 15, 18\},$$

$$R_1 = \{1, 4, 7, 10, 13, 16, 19\},$$

$$R_2 = \emptyset,$$

and so

$$S = \big\{1, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18, 19\big\}.$$

Graphically the solution schedule is given in Figure 5.3.

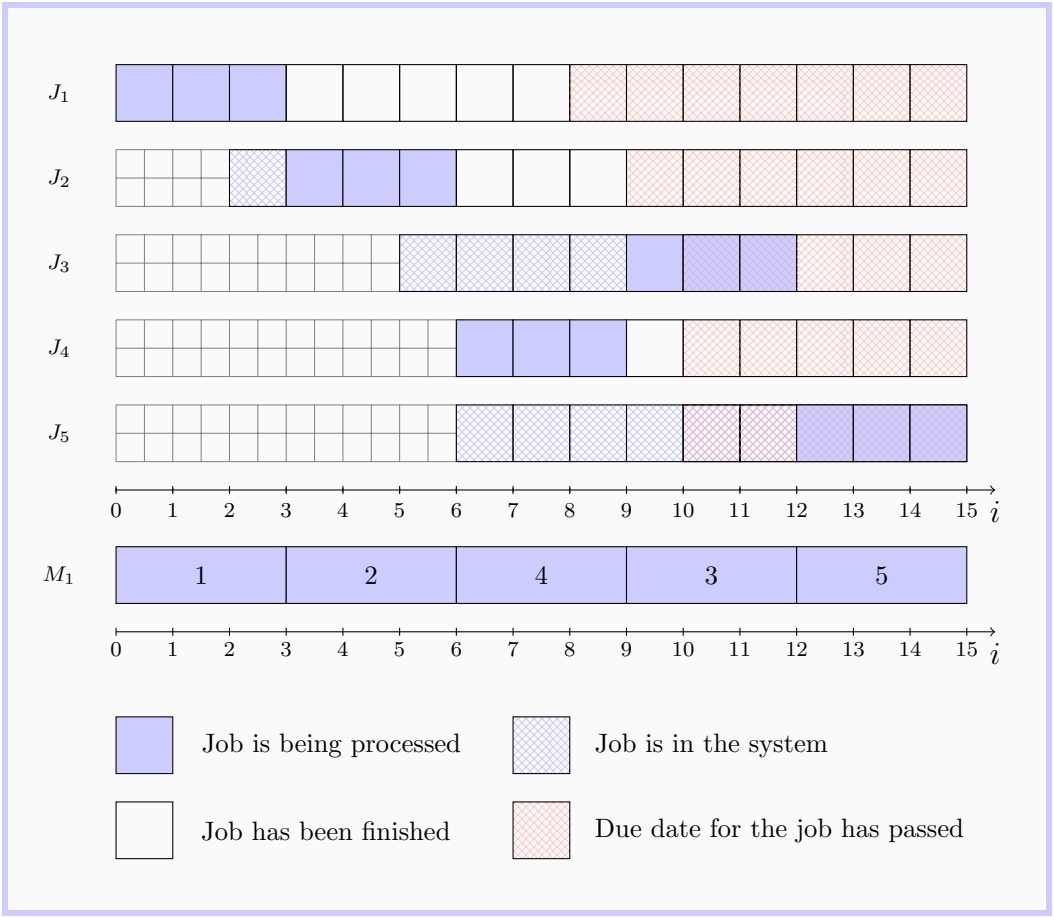


Figure 5.3: Solution schedule Example 5.6.1 with optimal value 18.

5.7 Conclusion

The single machine scheduling problem with equal processing times, release dates and the objective of minimizing total weighted tardiness was studied. The problem is modelled as an integer program and its linear programming relaxation was studied. The constraint matrix was shown to be totally unimodular and as a result the linear programming relaxation has an integer optimal solution, thus solving the integer program to optimality in polynomial time. The case when the schedule, under the same conditions as the general case, has a minimum completion time (or makespan) was also studied. Following a similar

methodology, the constraint matrix of the linear programming relaxation for this case was shown to be also totally unimodular and thus is also solved to optimality in polynomial time.

CHAPTER 6

$1 \mid r_j, pmtn \mid \sum w_j C_j$

In this chapter a single machine environment with n jobs to be processed is studied. The processing time $p_j \geqslant 0$, $p_j \in \mathbb{Z}_+$, required for each job j is given. Preemption is permitted, which means that the processing of a job can be interrupted at any time. The release date $r_j \geqslant 0$, $r_j \in \mathbb{Z}_+$, $j = 1, \ldots, n$, is also given. The objective is to minimize the weighted completion time defined as

$$\sum_{j=1}^{n} w_j C_j,$$

where C_j denotes the completion time of j and is available from a particular schedule. The weight $w_j \geqslant 0$, $j = 1, \ldots, n$, is known a priori. In Graham et al. (1979) notation this problem is denoted as $1 \mid r_j, pmtn \mid \sum w_j C_j$.

6.1 Complexity of the problem

Labetoulle et al. (1984) proved that the problem is $\mathcal{NP}$-hard in the strong sense, which means that it remains $\mathcal{NP}$-hard even when all the numerical data is bounded by a poly-

nomial function (Garey and Johnson (1978)). Sitters (2004) introduced a deterministic algorithm with a performance ratio of 1.56 which means that an approximate solution obtained with the algorithm will be less than 1.56 times the value of the optimal solution. Schulz and Skutella (2002) improved the latter result using a randomized linear programming based approximation algorithms with performance ratio of $\frac{4}{3}$ (1.33) and $\mathcal{O}(n \log n)$ time. Afrati et al. (1999) provided a polynomial-time approximation algorithm for the problem that finds a solution in $\mathcal{O}(2^{poly(1/\epsilon)}n + n \log n)$ time.

6.2 Formulation as an integer program

The problem is approached by dividing the interval of time that is considered in the schedule into m time slots.

The parameter m is established considering the release time and the processing time required for each job. Let $\hat{r}_1 \leqslant \hat{r}_2 \leqslant \ldots \leqslant \hat{r}_n$ be the release dates in ascending order and let $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n$ be the processing times corresponding to those release dates. In addition, let $s_1 = \hat{r}_1$ and for $l = 2, \ldots, n$,

$$s_l = \max\{s_{l-1} + \hat{p}_{l-1}, \hat{r}_l\}.$$

Thus $m = s_n + \hat{p}_n - 1$. Let S be the set of times at which there is at least one job available to be processed. For each s_l with $l = 1, \ldots, n$, calculate

$$R_l = \{s_l, s_l + 1, \ldots, s_l + \hat{p}_l - 1\}$$

and

$$S = R_1 \cup R_2 \cup \ldots \cup R_n.$$

Example 6.2.1. *In a single machine environment $n = 2$ jobs need to be processed. The release date r_j, processing time p_j and weight w_j for each job are given in the next table.*

Job	Data		
	p_j	r_j	w_j
1	2	1	1
2	3	2	5

With the given data, $s_1 = \hat{r}_1 = 1$ and $s_2 = \max\{s_1 + \hat{p}_1, \hat{r}_2\} = \max\{3, 2\} = 3$. The parameter $m = s_2 + \hat{p}_2 - 1 = 5$. Also, $R_1 = \{s_1, s_1 + 1\}$ and $R_2 = \{s_2, s_2 + 1, s_2 + 2\}$. Thus $S = \{1, 2, 3, 4, 5\}$.

Variables

Let x_j, $j = 1, \ldots, n$, be the variable indicating the units of time that a job spends in the system after it has been released. Also, let y_{ij} be the binary variable indicating whether a job is being processed at a particular time,

$$y_{ij} = \begin{cases} 1, & \text{if the job } j \text{ is processed at the time } i \in S,\ i \geqslant r_j; \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

The variable x_j, $j = 1, \ldots, n$, is an integer number, however for the existent relation between variables x_j and y_{ij}, which becomes clear in the next section, the constraint specifying x_j as an integer number is redundant and can be omitted.

Objective function

The objective is to minimize the total completion time,

$$\sum_{j=1}^{n} w_j C_j, \tag{6.2}$$

where C_j denotes the completion time of job j in a given schedule. Since a release date for each job is given then (6.2) becomes

$$\sum_{j=1}^{n} w_j (C_j - r_j).$$

In other words, a unit of time that a job spends in the system will increase the value of the objective function. Thus using the definition of x_j given above the objective function can be redefined as

$$\sum_{j=1}^{n} w_j x_j.$$

Constraints

On a single machine at most one job can be processed at the time, and so

$$\sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S.$$

The processing time for each job is p_j, thus

$$\sum_{i \in S_j} y_{ij} = p_j, \qquad j = 1, \ldots, n,$$

where

$$S_j = S \cap \{i \geqslant r_j\}, \qquad j = 1, \ldots, n.$$

A job is in the system once it arrives and until it has been completely processed, thus

$$y_{ij} \leqslant \frac{1}{i - r_j + 1}\, x_j, \qquad i \in S_j, \qquad j = 1, \ldots, n.$$

Letting $b_{ij} = \frac{1}{i-r_j+1}$,

$$y_{ij} \leqslant b_{ij}\, x_j, \qquad i \in S_j, \qquad j = 1, \ldots, n.$$

Integer programming formulation

Using the definitions given above the problem is formulated as an integer program.

Problem 6.2.2. *The formulation of the problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *as an integer program is given by*

$$\tau^{(\mathrm{IP})} = \min_{x_j,\, y_{ij}} \quad \sum_{j=1}^{n} w_j x_j \tag{6.3a}$$

$$s.t. \quad \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S, \tag{6.3b}$$

$$\sum_{i \in S_j} y_{ij} = p_j, \qquad j = 1, \ldots, n, \tag{6.3c}$$

$$y_{ij} - b_{ij} x_j \leqslant 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \tag{6.3d}$$

$$y_{ij}^2 - y_{ij} = 0, \qquad i \in S_j, \qquad j = 1, \ldots, n. \tag{6.3e}$$

6.3 Relaxation of the problem

6.3.1 Linear relaxation: By extension of the feasible set

Variables

Definition 6.3.1. *Let* $\boldsymbol{x}$ *denote the row vector containing all the variables* x_j, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$.

Definition 6.3.2. *Let* $\boldsymbol{y}_{(j)} = \left(y_{r_j j}, y_{r_j+1\, j}, \ldots, y_{mj}\right)$ *for* $j = 1, \ldots, n$ *with* y_{ij} *as defined by* (6.1).

Observe that the size of the vector $\boldsymbol{y}_{(j)}$ is at most $m - r_j + 1$, specifically is $|S_j|$, for all $j = 1, \ldots, n$. Let $u_j = |S_j|$. The definition of the positive semidefinite matrices related to the vector $\boldsymbol{y}_{(j)}$ is required.

Definition 6.3.3. *Let* $\mathsf{Y}_{(j)} = \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$ *be* $u_j \times u_j$ *matrices,*

$$\mathsf{Y}_{(j)} = \begin{pmatrix} y_{r_j j}^2 & y_{r_j j} y_{r_j+1\, j} & \cdots & y_{r_j j} y_{mj} \\ y_{r_j+1\, j} y_{r_j j} & y_{r_j+1\, j}^2 & \cdots & y_{r_j+1\, j} y_{mj} \\ \vdots & \vdots & \ddots & \vdots \\ y_{mj} y_{r_j j} & y_{mj} y_{r_j+1\, j} & \cdots & y_{mj}^2 \end{pmatrix}, \; j = 1, \ldots, n. \tag{6.4}$$

Remark The equalities $\mathsf{Y}_{(j)} = \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$ are nonlinear.

In the following lemmas some properties of the matrices $\mathsf{Y}_{(j)}$ are given.

Lemma 6.3.4. *Let* $\mathsf{Y}_{(j)}$ *as defined in* 6.3.3. *Then the following holds*

(i) The $\operatorname{rank}(\mathsf{Y}_{(j)}) = 1$, $j = 1, \ldots, n$.

(ii) The $\operatorname{nullity}(\mathsf{Y}_{(j)}) = |S_j| - 1$, $j = 1, \ldots, n$.

Proof. From the definition of the matrix $\mathsf{Y}_{(j)}$, it is possible to see that only one vector, $\boldsymbol{y}_{(j)} = \big(y_{r_j j}, y_{r_j+1\, j}, \ldots, y_{mj}\big)$, is required to generate the matrix and so $\operatorname{rank}(\mathsf{Y}_{(j)}) = 1$. With the well-known rank-nullity theorem[1] and since $\mathsf{Y}_{(j)}$ has $|S_j|$ columns, the dimension of the null space of $\mathsf{Y}_{(j)}$ is

$$\begin{aligned} \operatorname{nullity}(\mathsf{Y}_{(j)}) &= |S_j| - \operatorname{rank}(\mathsf{Y}_{(j)}), \qquad j = 1, \ldots, n, \\ \operatorname{nullity}(\mathsf{Y}_{(j)}) &= |S_j| - 1, \qquad j = 1, \ldots, n. \end{aligned}$$

□

Lemma 6.3.5. *Let* $\mathsf{Y}_{(j)}$ *as defined in* 6.3.3. *Then* $\operatorname{tr}(\mathsf{Y}_{(j)}) = \boldsymbol{y}_{(j)}^T\boldsymbol{y}_{(j)}$ *is the largest eigenvalue of* $\mathsf{Y}_{(j)}$, $j = 1, \ldots, n$, *and all the other eigenvalues are equal to zero.*

Proof. Let $\lambda_1, \ldots, \lambda_s$ be the eigenvalues of $\mathsf{Y}_{(j)}$. Without loss of generality denote the largest eigenvalue as λ_1. From Lemma 6.3.4 $\operatorname{rank}(\mathsf{Y}_{(j)}) = 1$, $j = 1, \ldots, n$. Then using the

[1] See Theorem A.2.3 in the Appendix A.

fact that the number of nonzero eigenvalues with multiplicity, denoted as $e(\mathsf{Y}_{(j)})$, is less than or equal to the rank of the matrix[2],

$$e(\mathsf{Y}_{(j)}) \leqslant 1, \qquad j = 1, \ldots, n$$

which indicates that there is at most one nonzero eigenvalue with multiplicity. Now since[3]

$$\sum_{i=1}^{s} \lambda_i = \operatorname{tr}(\mathsf{Y}_{(j)}), \qquad j = 1, \ldots, n$$

then $\lambda_1 = \operatorname{tr}(\mathsf{Y}_{(j)}) = \boldsymbol{y}_{(j)}^T \boldsymbol{y}_{(j)}$, $j = 1, \ldots, n$, and all the other eigenvalues λ_i for $i \neq 1$ are equal to zero.

□

Lemma 6.3.6. *The eigenvalues of* $\mathsf{Y}_{(j)}$, $j = 1, \ldots, n$, *as defined in* 6.3.3 *are nonnegative and so matrices* $\mathsf{Y}_{(j)}$ *are positive semidefinite for all* $j = 1, \ldots, n$.

Proof. This proof is omitted.

□

Objective function

Using vector notation $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^T$ and the objective function given in (6.3a) is rewritten as

$$\sum_{j=1}^{n} w_j x_j = \boldsymbol{w}^T \boldsymbol{x}.$$

[2]See Lemma A.2.5 in the Appendix A.
[3]See Corollary A.1.9 in the Appendix A.

Remark The objective function $\boldsymbol{w}^T\boldsymbol{x}$ is convex.

Constraints

Definition 6.3.7. *Let* $\hat{\mathsf{Y}}_{(j)}$ *be an* $|S| \times |S|$ *block diagonal matrix for* $j = 1, \ldots, n$*, such that the first block* B *corresponds to the* $(r_j - 1) \times (r_j - 1)$ *zero matrix and the second block corresponds to the matrix* $\mathsf{Y}_{(j)}$*,*

$$\hat{\mathsf{Y}}_{(j)} = \begin{pmatrix} \mathsf{B} & 0 \\ 0 & \mathsf{Y}_{(j)} \end{pmatrix}, \; j = 1, \ldots, n. \tag{6.5}$$

Note that B in Definition 6.3.7 can be a 0×0 matrix. In that case $\hat{\mathsf{Y}}_{(j)} = \mathsf{Y}_{(j)}$.

Lemma 6.3.8. *Let* $\hat{\mathsf{Y}}_{(j)}$ *as defined in* 6.3.7*. Then* $\hat{\mathsf{Y}}_{(j)}$ *is positive semidefinite for all* $j = 1, \ldots, n$*.*

Proof. A block diagonal matrix is positive semidefinite if and only if each diagonal block is positive semidefinite.

□

Denote the $|S| \times |S|$ matrix $\mathsf{A}_{(i)} = (a_{ij})$, $i \in S$, as the matrix with the element a_{ii} equal to 1 and all the other elements equal to zero. In the last notation the matrix $\mathsf{A}_{(1)}$

is for instance

$$\mathsf{A}_{(1)} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix}.$$

Remark The matrix $\mathsf{A}_{(i)}$ is positive semidefinite for all $i \in S$.

If $y_{ij}^2 - y_{ij} = 0$ (or $y_{ij} \in \{0, 1\}$), $i \in S_j$, $j = 1, \dots, n$, then constraints (6.3b) with Definition 6.3.7 can be rewritten as

$$\operatorname{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(1)}) + \operatorname{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(2)}) + \dots + \operatorname{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(n)}) \leqslant 1, \qquad i \in S,$$

and equations (6.3c) are equivalently represented by

$$\operatorname{tr}(\mathsf{Y}_{(j)}) = p_j, \qquad j = 1, \dots, n.$$

Let $\boldsymbol{b}_{(j)} = (b_{kj}, \dots, b_{lj})$, with $k, l \geqslant r_j$ and $k, l \in S$. Then constraints (6.3d) are equivalently represented by

$$\boldsymbol{y}_{(j)} - x_j\, \boldsymbol{b}_{(j)} \leqslant 0, \qquad j = 1, \dots, n.$$

Notation $(\mathsf{B})_{ij}$ is used to refer to the i, jth element of the matrix B, thus constraints (6.3e) are equivalent to

$$(\mathsf{Y}_{(j)})_{ii} = y_{ij}, \qquad i \in S_j, \qquad j = 1, \dots, n.$$

Using all the substitutions shown above the following problem definition is introduced.

Problem 6.3.9. *The scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *can be represented by the optimisation problem*

$$\begin{aligned}
\tau^{(\mathrm{IP}-2)} = \min_{x_j,\, \mathsf{Y}_{(j)},\, \boldsymbol{y}_{(j)}} \quad & \boldsymbol{w}^T\boldsymbol{x} \\
s.t. \quad & \mathrm{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(1)}) + \ldots + \mathrm{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(n)}) \leqslant 1, \qquad i \in S, \\
& \mathrm{tr}(\mathsf{Y}_{(j)}) = p_j, \qquad j = 1, \ldots, n, \\
& \boldsymbol{y}_{(j)} - x_j\,\boldsymbol{b}_{(j)} \leqslant 0, \qquad j = 1, \ldots, n, \\
& (\mathsf{Y}_{(j)})_{ii} - y_{ij} = 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& \mathsf{Y}_{(j)} = \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T, \qquad j = 1, \ldots, n, \\
& \hat{\mathsf{Y}}_{(j)} = \begin{pmatrix} \mathsf{B} & 0 \\ 0 & \mathsf{Y}_{(j)} \end{pmatrix}, \qquad j = 1, \ldots, n,
\end{aligned}$$

where $\mathsf{A}_{(i)} \in \mathbb{S}_+^{|S|}$ *for all* $i \in S$.

Note that Problem 6.3.9 is not a integer program however the notation $\tau^{(\mathrm{IP}-2)}$ is used for an equivalence given in Lemma 6.3.13. Also, the relaxation given in Problem 6.3.10 is denoted $\tau^{(\mathrm{LP}-2)}$ but is a semidefinite program.

By extending the feasible set of Problem 6.3.9 a semidefinite relaxation is provided.

Problem 6.3.10. *A relaxation for the scheduling problem* $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$ *stated in* Problem 6.3.9*, by extension of the feasible set, is given by*

$$
\begin{aligned}
\tau^{(\mathrm{LP}r-2)} = \min_{x_j,\, \mathsf{Y}_{(j)},\, \boldsymbol{y}_{(j)}} \quad & \boldsymbol{w}^T \boldsymbol{x} \\
s.t. \quad & \operatorname{tr}(\mathsf{A}_{(i)} \hat{\mathsf{Y}}_{(1)}) + \ldots + \operatorname{tr}(\mathsf{A}_{(i)} \hat{\mathsf{Y}}_{(n)}) \leqslant 1, \qquad i \in S, \\
& \operatorname{tr}(\mathsf{Y}_{(j)}) = p_j, \qquad j = 1, \ldots, n, \\
& \boldsymbol{y}_{(j)} - x_j \, \boldsymbol{b}_{(j)} \leqslant 0, \qquad j = 1, \ldots, n, \\
& (\mathsf{Y}_{(j)})_{ii} - y_{ij} = 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& \begin{pmatrix} 1 & \boldsymbol{y}_{(j)}^T \\ \boldsymbol{y}_{(j)} & \mathsf{Y}_{(j)} \end{pmatrix} \succcurlyeq 0, \qquad j = 1, \ldots, n, \\
& \hat{\mathsf{Y}}_{(j)} = \begin{pmatrix} \mathsf{B} & \mathsf{0} \\ \mathsf{0} & \mathsf{Y}_{(j)} \end{pmatrix}, \qquad j = 1, \ldots, n,
\end{aligned}
$$

where $\mathsf{A}_{(i)} \in \mathbb{S}_+^{|S|}$ *for all* $i \in S$.

In order to show that Problem 6.3.10 is in fact a relaxation of Problem 6.3.9 let $\mathcal{F}^{(\mathrm{IP}-2)}$ and $\mathcal{F}^{(\mathrm{LP}r-2)}$ denote the feasible sets of Problem 6.3.9 and Problem 6.3.10, respectively.

Lemma 6.3.11. Problem 6.3.10 *is a relaxation for the scheduling problem* $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$ *given in* Problem 6.3.9*, i.e.*

(i) $\mathcal{F}^{(\mathrm{IP}-2)} \subseteq \mathcal{F}^{(\mathrm{LP}r-2)}$.

(ii) $\tau^{(\mathrm{LP}r-2)} \leqslant \tau^{(\mathrm{IP}-2)}$*, for all* x_j, $\mathsf{Y}_{(j)}$, $\boldsymbol{y}_{(j)} \in \mathcal{F}^{(\mathrm{IP}-2)}$.

Proof. *(i)* The difference between the feasible sets of problems 6.3.9 and 6.3.10 relays only in one type of constraints. In $\mathcal{F}^{(\mathrm{IP}-2)}$ feasible $\mathsf{Y}_{(j)}$ matrices have the form $\mathsf{Y}_{(j)} = \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$, $j = 1, \ldots, n$, which is a non-convex constraint, and in $\mathcal{F}^{(\mathrm{LP}r-2)}$

$$\begin{pmatrix} 1 & \boldsymbol{y}_{(j)}^T \\ \boldsymbol{y}_{(j)} & \mathsf{Y}_{(j)} \end{pmatrix} \geqslant 0, \qquad j = 1, \ldots, n,$$

that is $\mathsf{Y}_{(j)} \geqslant \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$ for $j = 1, \ldots, n$, which is a convex constraint. This means that $\mathcal{F}^{(\mathrm{LP}r-2)}$ consider those feasible $\mathsf{Y}_{(j)}$ matrices required in $\mathcal{F}^{(\mathrm{IP}-2)}$ and some others. All the other constraints of both problems are exactly the same, thus $\mathcal{F}^{(\mathrm{IP}-2)} \subseteq \mathcal{F}^{(\mathrm{LP}r-2)}$.

(ii) Note that

$$\begin{aligned} \tau^{(\mathrm{LP}r-2)} &= \min_{x_j,\, \mathsf{Y}_{(j)},\, \boldsymbol{y}_{(j)} \in \mathcal{F}^{(\mathrm{LP}r-2)}} \boldsymbol{w}^T\boldsymbol{x} \\ &\leqslant \min_{x_j,\, \mathsf{Y}_{(j)}, \boldsymbol{y}_{(j)} \in \mathcal{F}^{(\mathrm{IP}-2)}} \boldsymbol{w}^T\boldsymbol{x} = \tau^{(\mathrm{IP}-2)}, \end{aligned}$$

where the first equality follows by definition and the second from the fact that $\mathcal{F}^{(\mathrm{IP}-2)} \subseteq \mathcal{F}^{(\mathrm{LP}r-2)}$.

Thus $\tau^{(\mathrm{LP}r-2)} \leqslant \tau^{(\mathrm{IP}-2)}$, for all x_j, $\mathsf{Y}_{(j)}$, $\boldsymbol{y}_{(j)} \in \mathcal{F}^{(\mathrm{IP}-2)}$ which completes the proof.

□

Remark Formulation given in Problem 6.3.10 is a convex quadratic relaxation.

Remark The difference between Problem 6.3.9 and Problem 6.3.10 is the replacement of the non-convex constraints and so the relaxation can be directly seen noting the extension of the feasible set of Problem 6.3.10.

Relation between Problem 6.3.10 and Problem 6.2.2

Problem 6.3.10 is a semidefinite programming relaxation. However it can be seen that such relaxation is equivalent to a linear programming relaxation of the integer formulation given in Problem 6.3.12.

Problem 6.3.12. *A relaxation for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *stated in* Problem 6.2.2 *is given by*

$$
\begin{aligned}
\tau^{(\mathrm{LP}r)} = \min_{x_j,\, y_{ij}} \quad & \sum_{j=1}^{n} w_j x_j \\
s.t. \quad & \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in S, \\
& \sum_{i \in S_j}^{m} y_{ij} = p_j, \qquad j = 1, \ldots, n, \\
& y_{ij} - b_{ij} x_j \leqslant 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& y_{ij} \in [0, 1], \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}
$$

In the following lemma the relation between Problems 6.3.12 and 6.3.10 is established.

Lemma 6.3.13. *The semidefinite programming relaxation in* Problem 6.3.10 *is equivalent to the relaxation of constraint* $y_{ij}^2 - y_{ij} = 0$ *for* $y_{ij} \in [0,1]$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, *in* Problem 6.2.2.

Proof. In the integer programming formulation, Problem 6.2.2, variables y_{ij} satisfy the constraint (6.3e), $y_{ij}^2 - y_{ij} = 0$, $i \in S_j$, $j = 1, \ldots, n$, which is $y_{ij} \in \{0, 1\}$. In Problem 6.3.9 such constraints have been modeled as $(\mathsf{Y}_{(j)})_{ii} = \boldsymbol{y}_{ij}$ and $\mathsf{Y}_{(j)} = \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$, for all $i \in S_j$, $j = 1, \ldots, n$. In the semidefinite programming relaxation in Problem 6.3.10 constraints (6.3e) have been in turn replaced with

$$\begin{aligned} (\mathsf{Y}_{(j)})_{ii} &= \boldsymbol{y}_{ij}, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\ \begin{pmatrix} 1 & \boldsymbol{y}_{(j)}^T \\ \boldsymbol{y}_{(j)} & \mathsf{Y}_{(j)} \end{pmatrix} &\succcurlyeq 0, \qquad j = 1, \ldots, n. \end{aligned}$$

However

$$\begin{pmatrix} 1 & \boldsymbol{y}_{(j)}^T \\ \boldsymbol{y}_{(j)} & \mathsf{Y}_{(j)} \end{pmatrix} \succcurlyeq 0 \iff \mathsf{Y}_{(j)} - \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T \succcurlyeq 0$$

which is $\mathsf{Y}_{(j)} \succcurlyeq \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T$. Now

$$\begin{aligned} \mathsf{Y}_{(j)} \succcurlyeq \boldsymbol{y}_{(j)}\boldsymbol{y}_{(j)}^T &\iff y_{ij} \geqslant y_{ij}^2 \\ &\iff y_{ij}^2 - y_{ij} \leqslant 0 \\ &\iff y_{ij} \in [0, 1] \end{aligned}$$

Hence the semidefinite programming relaxation in Problem 6.3.10 is equivalent to the relaxed Problem 6.2.2 with $y_{ij} \in [0, 1]$ instead of $y_{ij} \in \{0, 1\}$.

□

6.3.2 Semidefinite relaxation: Using Shor's technique

In this section a semidefinite programming relaxation using Shor's relaxation technique (Shor (1987)) is developed.

Variables

Let $\boldsymbol{x}$ be defined as in 6.3.1 and let $\mathsf{Y}_{(ij)}$ be defined as follows.

Definition 6.3.14. *Let* $\mathsf{Y}_{(ij)}$, $i \in S_j$, $j = 1, \ldots, n$, *be* 2×2 *matrices defined by*

$$\mathsf{Y}_{(ij)} = \begin{pmatrix} 1 & y_{ij} \\ y_{ij} & y_{ij}^2 \end{pmatrix}. \tag{6.8}$$

Lemma 6.3.15. *The matrices* $\mathsf{Y}_{(ij)}$ *as defined in* 6.3.14 *are positive semidefinite for all* $i \in S_j$, $j = 1, \ldots, n$.

Proof. Observe that

$$\mathsf{Y}_{(ij)} = \begin{pmatrix} 1 \\ y_{ij} \end{pmatrix} \begin{pmatrix} 1 \\ y_{ij} \end{pmatrix}^T$$

which means that only one vector, i.e. $(1, y_{ij})^T$, is required to generate the matrix $\mathsf{Y}_{(ij)}$. Thus $\operatorname{rank}(\mathsf{Y}_{(ij)}) = 1$, $i \in S_j$, $j = 1, \ldots, n$. Using Lemma 6.3.5 it can be seen that $\operatorname{tr}(\mathsf{Y}_{(ij)})$

is the largest eigenvalue of $\mathsf{Y}_{(ij)}$ and all the other eigenvalues are equal to zero. Since the trace is a nonnegative number then $\mathsf{Y}_{(ij)}$, $i \in S_j$, $j = 1, \ldots, n$, is positive semidefinite for any value y_{ij}.

□

Objective function

The objective function remains the same as in Section 6.3, i.e.

$$\sum_{j=1}^{n} w_j x_j = \boldsymbol{w}^T \boldsymbol{x}.$$

Constraints

Using Lemma B.4.13[4] and Definition 6.3.14 constraints (6.3b) can be rewritten as

$$\sum_{j=1}^{n} \operatorname{tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) \leqslant 1, \qquad i \in S.$$

In the same way, constraints (6.3c) are equivalent to

$$\sum_{i \in S_j} \operatorname{tr}\left(\begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = p_j, \qquad j = 1, \ldots, n.$$

[4] See Appendix B.

In the case of constraints (6.3d)

$$\left(\mathsf{Y}_{(ij)}\right)_{12} - b_{ij}\,x_j \leqslant 0, \qquad i \in S_j, \qquad j = 1, \ldots, n.$$

Finally constraint (6.3e) is

$$\operatorname{tr}\left(\begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = 0, \qquad i \in S_j, \qquad j = 1, \ldots, n.$$

Problem 6.2.2 is then rewritten in Problem 6.3.16.

Problem 6.3.16. *The scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *can be formulated as*

$$\begin{aligned}
\tau^{(\mathrm{SDP})} = \min_{x_j,\, \mathsf{Y}_{(ij)},\, y_{ij}} \quad & \boldsymbol{w}^T\boldsymbol{x} \\
s.t. \quad & \sum_{j=1}^{n} \operatorname{tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) \leqslant 1, \qquad i \in S, \\
& \sum_{i \in S_j}^{m} \operatorname{tr}\left(\begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = p_j, \qquad j = 1, \ldots, n, \\
& \left(\mathsf{Y}_{(ij)}\right)_{12} - b_{ij}\,x_j \leqslant 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& \operatorname{tr}\left(\begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = 0, \qquad i \in S_j, \qquad j = 1, \ldots, n, \\
& \mathsf{Y}_{(ij)} = \begin{pmatrix} 1 & y_{ij} \\ y_{ij} & y_{ij}^2 \end{pmatrix}, \qquad i \in S_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

A relaxation for Problem 6.3.16 is introduced next.

Problem 6.3.17. *A relaxation for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *stated in* Problem 6.3.16*, using Shor's relaxation technique, is given by*

$$
\begin{aligned}
\tau^{(\mathrm{SDP}r)} = \min_{x_j,\,\mathsf{Y}_{(ij)}} \quad & \boldsymbol{w}^T\boldsymbol{x} \\
s.t. \quad & \sum_{j=1}^{n} \operatorname{tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) \leqslant 1, \qquad i \in S, \\
& \sum_{i\in S_j}^{m} \operatorname{tr}\left(\begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = p_j, \qquad j = 1,\ldots,n, \\
& \left(\mathsf{Y}_{(ij)}\right)_{12} - b_{ij}\,x_j \leqslant 0, \qquad i \in S_j, \qquad j = 1,\ldots,n, \\
& \operatorname{tr}\left(\begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \mathsf{Y}_{(ij)}\right) = 0, \qquad i \in S_j, \qquad j = 1,\ldots,n, \\
& \left(\mathsf{Y}_{(ij)}\right)_{11} = 1, \qquad i \in S_j, \qquad j = 1,\ldots,n, \\
& \mathsf{Y}_{(ij)} \succcurlyeq 0, \qquad i \in S_j, \qquad j = 1,\ldots,n.
\end{aligned}
$$

Problem 6.3.17 is shown to be a relaxation of Problem 6.3.16 in Lemma 6.3.18. Let $\mathcal{F}^{(\mathrm{SDP})}$ and $\mathcal{F}^{(\mathrm{SDP}r)}$ denote the feasible set of Problem 6.3.9 and Problem 6.3.10, respectively.

Lemma 6.3.18. Problem 6.3.17 *is a relaxation for the scheduling problem* $1\,|\,p_j = p, r_j\,|\,T_{wt}$ *given in* Problem 6.3.16*, i.e.*

(i) $\mathcal{F}^{(\mathrm{SDP})} \subseteq \mathcal{F}^{(\mathrm{SDP}r)}$.

(ii) $\tau^{(\mathrm{SDP}r)} \leqslant \tau^{(\mathrm{SDP})}$, *for all* x_j, $\mathsf{Y}_{(ij)} \in \mathcal{F}^{(\mathrm{SDP})}$.

Proof. *(i)* Observe that Problem 6.3.16 has a linear objective and linear equality constraints with the symmetric matrix $\mathsf{Y}_{(ij)}$ as defined in 6.3.14. Matrices $\mathsf{Y}_{(ij)}$ are positive semidefinite as shown in Lemma 6.3.15 with $(\mathsf{Y}_{(ij)})_{11} = 1$. By restricting the feasible set to those matrices the relaxation is defined and $\mathcal{F}^{(\mathrm{SDP})} \subseteq \mathcal{F}^{(\mathrm{SDP}r)}$.

(ii) In this case it holds by definition that

$$\begin{aligned}\tau^{(\mathrm{SDP}r)} &= \min_{x_j,\,\mathsf{Y}_{(ij)} \in \mathcal{F}^{(\mathrm{SDP}r)}} \boldsymbol{w}^T\boldsymbol{x} \\ &\leqslant \min_{x_j,\,\mathsf{Y}_{(ij)} \in \mathcal{F}^{(\mathrm{SDP})}} \boldsymbol{w}^T\boldsymbol{x} = \tau^{(\mathrm{SDP})}\end{aligned}$$

where the second inequality holds since $\mathcal{F}^{(\mathrm{SDP})} \subseteq \mathcal{F}^{(\mathrm{SDP}r)}$.

Thus $\tau^{(\mathrm{SDP}r)} \leqslant \tau^{(\mathrm{SDP})}$, for all x_j, $\mathsf{Y}_{(ij)} \in \mathcal{F}^{(\mathrm{SDP})}$.

□

In the following section the relaxations, the linear programming based (Problem 6.3.12), denoted as LPr, the second linear programming based using matrix variables (Problem 6.3.10) denotes as LPr-2 and the semidefinite programming one (Problem 6.3.17), denoted as SDPr, developed for the scheduling problem $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ are tested. For this purpose a customised branch and bound algorithm has been developed.

In the subsequent pages Problems 6.2.2, 6.3.9 and 6.3.16 are referred as primal problems and Problems 6.3.12, 6.3.10 and 6.3.17 as its corresponding relaxations.

6.4 Branch and bound algorithm

By dividing the feasible set of the formulation given in the previous section for the scheduling problem $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$, a branch and bound algorithm is proposed. An enumeration tree is built by decomposing the feasible set into smaller sets. The enumeration tree grows only in those specific parts that are worth exploring. This is decided by lower and upper bounds that are found in the process.

Let $\tau^{(*)}$ and $\mathcal{F}^{(*)}$ denote the objective function and feasible set, respectively of the primal problem. Also let $\tau^{(\mathrm{R})}$ and $\mathcal{F}^{(\mathrm{R})}$ refer to the objective function and feasible set, respectively, of the corresponding relaxation for the above problem.

Definition 6.4.1. *Let $\underline{\tau}^{(*)}$ and $\overline{\tau}^{(*)}$ be lower and upper bounds respectively of the optimal value for the primal formulation of the scheduling problem $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$,*

$$\underline{\tau}^{(*)} \leqslant \tau^{(*)} \leqslant \overline{\tau}^{(*)}.$$

The relation between upper and lower bounds for each subproblem against the global problem is presented in the next Lemma.

Lemma 6.4.2. *Let* $\tau^{(*)}$ *be the optimal value of* Problem 6.3.9 (*or* Problem 6.3.16)*, i.e.*

$$\tau^{(*)} = \min_{x_j,\, \mathsf{Y}_{(j)}} \left\{ \boldsymbol{w}^T \boldsymbol{x} \,\middle|\, x_j,\, \mathsf{Y}_{(j)} \in \mathcal{F}^{(*)} \right\}. \tag{6.9}$$

Also, let

$$\mathcal{F}^{(*)} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_K, \tag{6.10}$$

be a decomposition of the feasible set. Let $\underline{\tau}^{(i)}$ *and* $\overline{\tau}^{(i)}$ *be lower and upper bounds, respectively, of*

$$\tau^{(i)} = \min_{x_j,\, \mathsf{Y}_{(j)}} \left\{ \boldsymbol{w}^T \boldsymbol{x} \,\middle|\, x_j,\, \mathsf{Y}_{(j)} \in \mathcal{F}_i \right\},$$

for $i = 1, \ldots, k$*. Then*

$$\underline{\tau}^{(*)} := \min_i \underline{\tau}^{(i)} \leqslant \tau^{(*)} \leqslant \min_i \overline{\tau}^{(i)} := \overline{\tau}^{(*)},$$

are upper and lower bounds of $\tau^{(*)}$*.*

Proof. By contradiction assume that $\underline{\tau}^{(i)} < \underline{\tau}^{(*)}$ for some i such that $\tilde{x}_j,\, \tilde{\mathsf{Y}}_{(j)} \in \mathcal{F}_i$. From equation (6.10) it holds that $\mathcal{F}_i \subseteq \mathcal{F}^{(*)}$. Thus the values of $\tilde{x}_j$ and $\tilde{\mathsf{Y}}_{(j)} \in \mathcal{F}^{(*)}$ are feasible in (6.9) and so $\underline{\tau}^{(*)} = \underline{\tau}^{(i)}$ which contradicts the initial assumption. The same analysis can be done with the upper bounds.

□

Three aspects need to be clarified to put the branch and bound method into context.

(i) Building a enumeration tree for the feasible set.

(*ii*) Calculating primal and dual bounds for each subproblem in the enumeration tree.

(*iii*) Tightening bounds and pruning branches.

6.4.1 Constructing a enumeration tree for the feasible set

Node selection

The enumeration tree is constructed by dividing the feasible set $\mathcal{F}^{(*)}$ on a specific node into subproblems, or child nodes, following a splitting rule. The selected node for branching can be any of the open nodes. In this case the most promising node, the one with the lowest lower bound is branched. The variables y_{ij} are defined as binary variables but the variables x_j need not to be forced to be integers because their values depend directly on the choices for the variables y_{ij}. Therefore the splitting rule considers only those variables y_{ij} that are not integers in a solution.

Branching strategy

In a traditional branch and bound algorithm a variable would be selected and two branches created by fixing the variable to both values, 0 and 1, such that

$$\mathcal{F}^{i+1}_{y_{ij}=0} = \mathcal{F}^i \cap \{y_{ij} \,|\, y_{ij} = 0\},$$

and

$$\mathcal{F}^{i+1}_{y_{ij}=1} = \mathcal{F}^i \cap \{y_{ij} \,|\, y_{ij} = 1\},$$

as in Figure 6.1. Observe that under this approach the enumeration tree has

$$\sum_{k=0}^{\tilde{n}} 2^k \tag{6.11}$$

nodes, where $\tilde{n}$ is the number of *variables*, not the number of *jobs*.

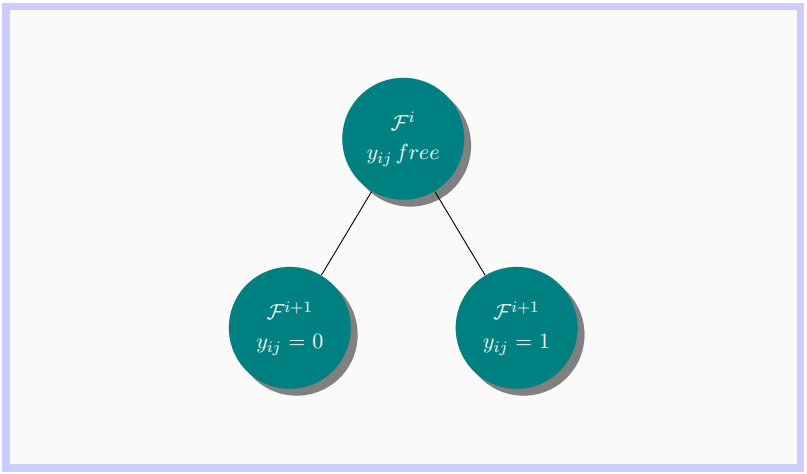


Figure 6.1: Standard branching rule for a traditional branch and bound algorithm

Now, recall that to allow only one job at the time in the machine, in the IP formulation in Problem 6.2.2, the constraint

$$\sum_{j=1}^{n} y_{ij} = 1, \qquad i \in S,$$

was added. Note that because of this constraint when one of the variables y_{ij} is equal to one all the other variables y_{ik}, with $k \neq j$, are equal to zero for the same i. The latter indicates that the variables y_{ij} constitute a special ordered set (Beale and Tomlin (1970)), in this case, a multiple choice set, at which at most one variable for the same i can take a nonzero value and all the others must be equal to zero.

Considering the characteristics of variables y_{ij} as an ordered set, not two but several branches, one per job available at the time, with $y_{ij} = 1$ would be created,

$$\mathcal{F}^{i,k}_{y_{ik}=1} = \mathcal{F}^i \cap \{y_{ik} \,|\, y_{ik} = 1, r_k \geqslant i\}\,.$$

To this point from each node there is one branch per job available and whose variable has been made equal to one. However a further characteristic has been added into the branching strategy. Theorem 6.4.3 gives the additional rules to be considered when branching.

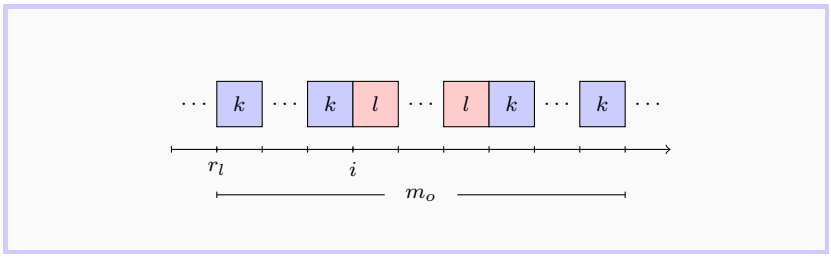
Theorem 6.4.3. *Let $j = 1, \ldots, n$ be given jobs with processing times p_j, release dates r_j and weights w_j. Also, let S be a schedule for the given data. Then S is optimal for $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$ if and only if*

(i) in S no job j_k is interrupted at the time i, to process another job j_l, unless j_l is arriving at the aforementioned time i, i.e. $r_l = i$.

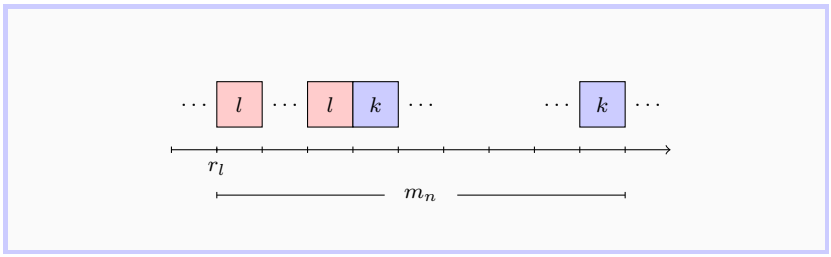
(ii) when the job j_k is interrupted to process the job j_l, the processing of j_k will be resumed immediately after j_l has been finished. If more than one job has been interrupted then a last-interrupted, first-resumed approach follows.

Proof. *(i)* By contradiction, assume that S_o is an optimal schedule for the given data, such that the processing of the job j_k in the machine has been interrupted at the time i to process the job j_l, and $r_l < i$. Let t be the difference between r_l and i, that is, $r_l + t = i$. See Figure 6.2a.

Without loss of generality, assume that no other job has arrived nor has been processed in the considered interval m_o. Recall the objective function, $\sum w_j x_j$ where x_j represents the units of time that a job spends in the system, thus the cost of S_o



(a) Schedule S_o



(b) Schedule S_n

Figure 6.2: Relation between preemption, release dates and optimality as established in Theorem 6.4.3

for the jobs k and l in the interval m_o is given by

$$w_k(p_k + p_l) + w_l(p_l + t).$$

Now, generate a new schedule from S_o, say S_n, starting the processing of j_l at the time r_l, see Figure 6.2b. Observe that the weighted processing time of S_n in the interval m_n is

$$w_l p_l + w_k(p_k + p_l).$$

Thus, since $w_l \geqslant 0$

$$w_k(p_k + p_l) + w_l(p_l + t) \geqslant w_l p_l + w_k(p_k + p_l)$$

$$w_l p_l + w_l t \geqslant w_l p_l$$

$$S_o \geqslant S_n$$

which contradicts the fact that S_o is an optimal schedule and the assertion that was claimed follows.

(ii) From *(i)* the processing of a job in an optimal schedule is interrupted if and only if another job has arrived, therefore in an optimal schedule unless another job is arriving the processing of the interrupted job will be resumed.

□

In this way considering the structure of the problem and to avoid the inefficient process of carrying out the traditional branching process, mentioned above, of selecting only one variable at the time (Tomlin (1988)), a more suitable enumeration tree for the formulation of the problem is required.

Using Theorem 6.4.3 and the characteristics of the ordered set of the variables, once a new job arrives a new branch is created with the alternative of processing a different job in the machine. This will affect the construction of the enumeration tree in the sense that a branch will only occur in three cases, that are explained below.

The job is released at the time: When a job is arriving to the system, which is indicated by the release date r_j, it can be assigned to the machine to be processed immediately.

The job was interrupted and should be resumed: Following the previous case, if a job was interrupted in the machine to give priority to one arriving job then the processing of the interrupted job will be resumed after the machine has finished with the arriving job. This process can continue and a job can be interrupted by more than one arriving job and in those cases the processing for each job is resumed in a last-interrupted, first-resumed basis.

The machine is free to process the job: When the machine has finished the processing of a job then a branch is created for each of the available jobs.

Following this new branching strategy at each node there will be the same number of branches as the number of jobs available at the time. The point at which the branch will split again is found by calculating the minimum between the remaining processing time and the release date among the jobs belonging to each branch, i.e.

$$l = \min \left\{ \hat{p}_j, \hat{r}_k \,\middle|\, k \in t, k \neq j \right\}, \tag{6.12}$$

where $\hat{r}_j$, $j = 1, \ldots, n$ are the release dates in ascending order and $\hat{p}_j$, $j = 1, \ldots, n$ are the remaining processing times corresponding to those release dates.

Relaxations revisited

The new branching strategy indicates that not all the variables y_{ij} in Problem 6.2.2 are active in the solution of the problem. Thus the definition of Problem 6.2.2 can be modified to reflect those changes. The first change consist in alter the set S so that it contains only those times i at which a different job can be processed in the machine.

Let $\hat{S}$ be the set containing the times defined in (6.12). Note that the set S_j, was defined as

$$S_j = S \cap \left\{ i \geqslant r_j \right\}, \qquad j = 1, \ldots, n,$$

and therefore will be updated as

$$\hat{S}_j = \hat{S} \cap \left\{ i \geqslant r_j \right\}, \qquad j = 1, \ldots, n,$$

to reflect the changes that have been made in the set S. Similarly, variables y_{ij}, that were

defined for all $i \in S$ are now defined only for those $i \geqslant r_j$ such that $i \in \hat{S}$.

Constraints (6.3b) required a change as well. In the older definition variables y_{ij} were defined for each member of the set S containing consecutive values of i. Now since some of variables y_{ij} are not active if $i \in \hat{S}$ the remaining y_{ij} need to compensate for those changes to give an appropriate solution to the problem. For example if $i = 1, 3$, $i \in \hat{S}$ but $i = 2$, $i \notin \hat{S}$, the variable y_{3j} in the processing time constraint has to be multiplied by two to indicate that such variable actually represents two slots of time. In such case if $y_{3j} = 1$ in a solution schedule, it will indicate that the job is being processed at the times $i = 2$ and $i = 3$.

This can be accomplished by letting the set $\hat{S}w$ contain the weights corresponding to each slot i, which is calculated by adding one for each consecutive missing i in $\hat{S}$. Note that $\hat{S}w$ depends on i.

Recall Example 6.2.1 where $S = \{1, 2, 3, 4, 5\}$. The updated set S becomes $\hat{S} = \{1, 2, 3, 5\}$. Since $i = 1, 2, 3$, are in S, then $\hat{S}w(1) = \hat{S}w(2) = \hat{S}w(3) = 1$. The time $i = 4$ is not in $\hat{S}$ and so $\hat{S}w(4) = 0$ and $\hat{S}w(5) = 2$. Thus $\hat{S}w = \{1, 1, 1, 0, 2\}$.

Equation (6.3d) remains exactly the same because b_{ij} correspond to each of the variables y_{ij}. In this case such constraints will only be created for each $i \in \hat{S}$.

The integer programming formulation given in Problem 6.2.2 is modified to reflect all these changes.

Problem 6.4.4. *The formulation of the problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *as an integer program is given by*

$$\begin{aligned}
\tau^{(\mathrm{IP})} = \min_{x_j,\, y_{ij}} \quad & \sum_{j=1}^{n} w_j x_j \\
s.t. \quad & \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in \hat{S}, \\
& \sum_{i \in \hat{S}_j}^{m} y_{ij} = p_j, \qquad j = 1, \ldots, n, \\
& y_{ij} - b_{ij} x_j \leqslant 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& y_{ij}^2 - y_{ij} = 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

Similarly the changes using Theorem 6.4.3 can be adjusted in Problems 6.3.12, 6.3.10 and 6.3.17 as follows.

Problem 6.4.5. *A relaxation for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *stated in Problem* 6.4.4 *is given by*

$$\begin{aligned}
\tau^{(\mathrm{LP}r)} = \min_{x_j,\, y_{ij}} \quad & \sum_{j=1}^{n} w_j x_j \\
s.t. \quad & \sum_{j=1}^{n} y_{ij} \leqslant 1, \qquad i \in \hat{S}, \\
& \sum_{i \in \hat{S}_j}^{m} y_{ij} = p_j, \qquad j = 1, \ldots, n, \\
& y_{ij} - b_{ij} x_j \leqslant 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& y_{ij} \in [0, 1], \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n.
\end{aligned}$$

Problem 6.4.6. *A relaxation for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ *stated in* Problem 6.3.9*, by extension of the feasible set, is given by*

$$
\begin{aligned}
\tau^{(\mathrm{LP}r-2)} = \min_{x_j,\,\mathsf{Y}_{(j)},\,\boldsymbol{y}_{(j)}} \quad & \boldsymbol{w}^T\boldsymbol{x} \\
s.t. \quad & \operatorname{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(1)}) + \ldots + \operatorname{tr}(\mathsf{A}_{(i)}\hat{\mathsf{Y}}_{(n)}) \leqslant 1, \qquad i \in \hat{S}, \\
& \operatorname{tr}(\mathsf{Y}_{(j)}) = p_j, \qquad j = 1, \ldots, n, \\
& \boldsymbol{y}_{(j)} - x_j\,\boldsymbol{b}_j \leqslant 0, \qquad j = 1, \ldots, n, \\
& (\mathsf{Y}_{(j)})_{ii} - y_{ij} = 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& \begin{pmatrix} 1 & \boldsymbol{y}_{(j)}^T \\ \boldsymbol{y}_{(j)} & \mathsf{Y}_{(j)} \end{pmatrix} \succcurlyeq 0, \qquad j = 1, \ldots, n, \\
& \hat{\mathsf{Y}}_{(j)} = \begin{pmatrix} 0 & 0 \\ 0 & \mathsf{Y}_{(j)} \end{pmatrix}, \qquad j = 1, \ldots, n,
\end{aligned}
$$

where $\mathsf{A}_{(i)} \in \mathbb{S}_+^m$ *for all* $i = 1, \ldots, m$.

Using similar technique to the one using in Lemma 6.3.11 it can be shown that Problem 6.4.6 is in fact a relaxation of Problem 6.3.9.

Problem 6.4.7. *A relaxation for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_j C_j$ *stated in* Problem 6.3.16*, using Shor's relaxation technique, is given by*

$$
\begin{aligned}
\tau^{(\mathrm{SDP}r)} = \min_{x_j,\, \mathsf{Y}_{(ij)}} \quad & \boldsymbol{w}^T \boldsymbol{x} \\
s.t. \quad & \sum_{j=1}^{n} \operatorname{tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \mathsf{Y}_{(ij)} \right) \leqslant 1, \qquad i \in \hat{S}, \\
& \sum_{i \in \hat{S}_j}^{m} \operatorname{tr}\left(\begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}, \mathsf{Y}_{(ij)} \right) = p_j, \qquad j = 1, \ldots, n, \\
& \left(\mathsf{Y}_{(ij)}\right)_{12} - b_{ij}\, x_j \leqslant 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& \operatorname{tr}\left(\begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \mathsf{Y}_{(ij)} \right) = 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& \left(\mathsf{Y}_{(ij)}\right)_{11} = 1, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n, \\
& \mathsf{Y}_{(ij)} \succcurlyeq 0, \qquad i \in \hat{S}_j, \qquad j = 1, \ldots, n.
\end{aligned}
$$

Problem 6.4.7 can be shown to be a relaxation of Problem 6.3.9 following a similar approach to the one used in Lemma 6.3.18.

Enumeration tree

Another characteristic that has been taking into account in the construction of the branch and bound algorithm is that if there is only one job available in the system then such job is assigned to the machine. A new enumeration tree will start when there are more than one job available. The enumeration tree is constructed following Algorithm 6.4.8.

Algorithm 6.4.8. *Branching for the scheduling problem* $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$.

Input: $\mathcal{F}^i$, y_{ij}, $i \in \hat{S}$, $j = 1, \ldots, n$.
Output: $\mathcal{F}^{l,j}_{y_{kj}=1} = \mathcal{F}^i \cap \left\{ y_{kj} = 1 \,|\, i \leqslant k \leqslant l, k \in \hat{S} \right\}$, for all $j = 1, \ldots, n$ with $r_j \geqslant i$.

(*Determine the number of branches*)
for $j = 1, \ldots, n$ **do**
  if $r_j \geqslant i$ (*job* j *is available*) **and** $\sum_{k=1}^{i-1} y_{kj} < p_j$ (*job* j *is unfinished*) **then**
    $t \leftarrow j$.
  end if
end for
(*Find the length of each branch*)
for $j \in t$ **do**
  $l = \min\left\{ \hat{p}_j, \hat{r}_k \,\middle|\, k \in t, k \neq j \right\}$.
end for
(*Create the branches*)
for $j \in t$ **do**
  $\mathcal{F}^{l,j} \leftarrow \mathcal{F}^{l,j} \cup \left(\mathcal{F}^i \cap y_{vj} = 1\right)$, $v \leqslant l$.
end for

The number of nodes in the enumeration tree using Algorithm 6.4.8 depends on the number of jobs and the release dates of those jobs. The maximum number of potential nodes in the enumeration tree using the above algorithm is addressed.

Let k_i denote the number of jobs available at the time i. Let h_i be the binary variable equal to one when a job has been release at the time i and there is more than one job available, or when the machine is free to process another job,

$$h_i = \begin{cases} 1, & i = r_j \text{ and } k_i > 1,\ j = 1, \ldots, n,\ i \in S, \text{ or the machine is free;} \\ 0, & \text{otherwise.} \end{cases}$$

Lemma 6.4.9. *Using* Algorithm 6.4.8 *and assuming that when there is only one job available it will be assigned to the machine, the maximum number of nodes in the enumeration tree for the branch and bound algorithm is bounded by*

$$\sum_{i\in S}\big(h_i k_i^{k_i-1} + (1-h_i)\big). \tag{6.15}$$

Proof. In a traditional branch and bound algorithm with binary variables at each level of the enumeration tree a decision with two choices, to fix a selected variable to one or to zero, is made. The process follows until no further variables, out of the $\tilde{n}$ variables remain. Thus $\sum_{k=0}^{\tilde{n}} 2^k$ represents the total number of nodes.

Under the new approach, the number of jobs available at a particular time i will affect the number of branches. At each level i the branching process will create k_i branches. The corresponding variable at each branch is fix to one. The process is repeated until $i = 1, \ldots, m$. Without considering anything else the number of nodes would be $\sum_{i=1}^{m} k_i^{k_i-1}$.

Now, by Theorem 6.4.3, not every $i = 1, \ldots, m$, is worth exploring but only those i's satisfying either one of the conditions,

(i) the machine has finished processing another job and is free at the time i to process a new one.

(ii) $i = r_j$ for some $j = 1, \ldots, n$, which are those times i at which a job has arrived and there was more than one job available.

Variable $h_i \in \{0, 1\}$ was defined to identify those i satisfying either of the cases mentioned above and the number of nodes can be counted at each level. Thus when $h_i = 1$ then at least one of the conditions is satisfied. The enumeration tree splits then

into k_i branches at any of the available nodes, k_i,

$$h_i k_i^{k_i - 1}$$

If there is only one job available then the enumeration tree has only one node, $1 - h_i = 1$, which corresponds to the assignment of the only available job to the machine. Adding the values given by the latter equation for $i \in S$ equation (6.15) is obtained.

□

Using the data of Example 6.2.1 the number of nodes of the enumeration tree is calculated.

The set $\hat{S} = \{1, 2, 3, 5\}$. However since there was only one job available in the machine at the time $i = 1$ then there are 2 enumeration trees. Therefore the set S is split into two one for each of the enumeration trees. In the first one $\hat{S} = 1$ and in the second one $\hat{S} = \{2, 3, 5\}$

For the first tree one job has been released at the time $i = 1$ and since there is only one job available then such job will be assigned to be processed at the machine during that time and $h_1 = 0$. The number of nodes for the first tree is equal to one.

For the second tree, at $i = 2$, $h_2 = 1$. Observe that in the node $i = 2$ there are only two jobs then independently of the value of equation (6.15) the main node splits into two branches each of them with one job and the corresponding $y_{ij} = 1$. Using equation (6.15) at the time $i = 2$, since there was only one node in the previous step, $k_2 - 1 = 1$, and so when $i = 3$, $(1)2^1 = 2$, and so two extra nodes can be placed in the enumeration tree by considering the processing times of both jobs. For the left branch of node 2, job $j = 1$ has been processed is being processed in the machine and since for such job there is

one unit of processing time remaining then in such case variable $y_{21} = 1$ and the branch is completed by setting the only possible choice left, by assigning job $j = 2$ after the processing of job $j = 1$ has been finished.

In the same way on the right branch of node 2, job $j = 2$ has been assigned to the machine, with three units of processing time remaining the variables $y_{22} = y_{32} = y_{42} = 1$ are fixed. Once the processing of the job one has been done then job $j = 1$ is assigned to the machine.

The number of nodes is $1 + 1 + 2 = 4$ as shown in Figure 6.3.

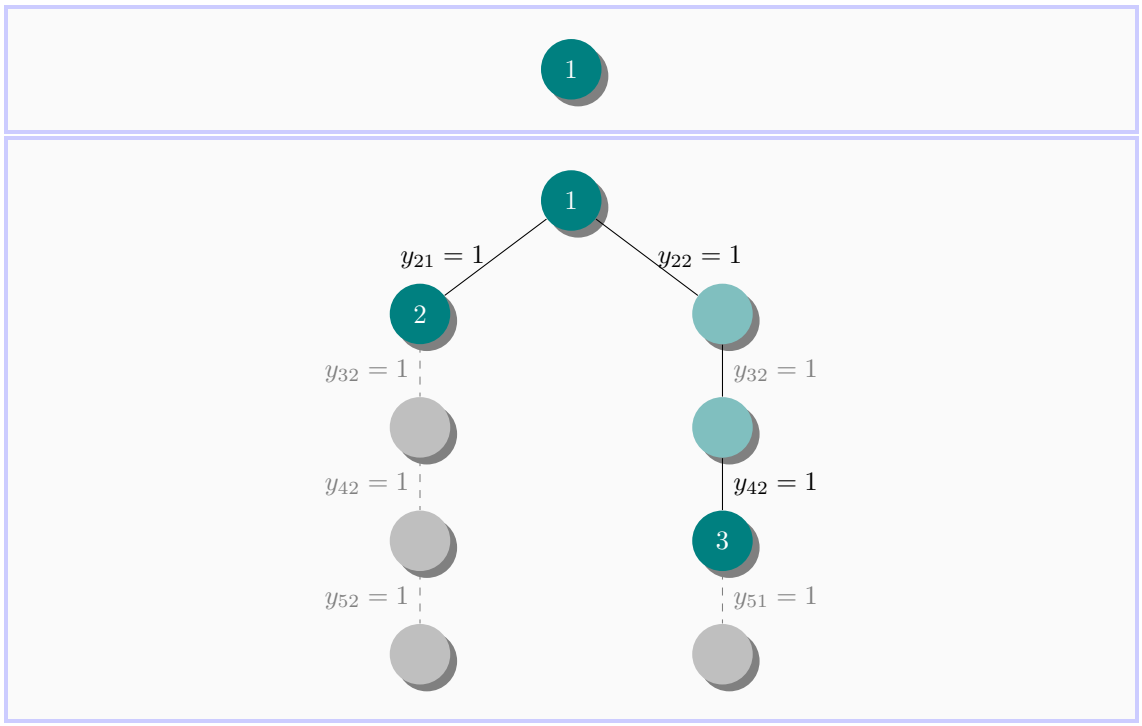


Figure 6.3: Enumeration tree for Example 6.2.1 using Algorithm 6.4.8

6.4.2 Calculating primal and dual bounds

Dual bounds

A dual bound for a minimization problem is any optimal solution obtained for a relaxation. Using Lemma 6.3.11 (also Lemma 6.3.18)),

$$\tau^{(R)} \leqslant \tau^{(*)}.$$

Thus the relaxations in Problems 6.4.5, 6.4.6 and 6.4.7 are used to determine dual bounds.

Primal bounds

A primal bound for a minimization problem is any feasible solution. If the solution to the relaxation problems is binary, then it is used as an upper bound for the problem.

6.4.3 Tightening bounds and pruning branches

Pruning mechanisms

The major advantage of using a branch and bound algorithm is the ability to prune those branches of the enumeration tree which are not worth exploring further. This decision is made by considering the lower and upper bounds which have been obtained so far. The pruning rules are explained below.

Pruning by bound: If the lower bound for a subproblem $\underline{\tau}_k^s$, is greater than the upper bound of the global problem $\overline{\tau}^{(*)}$, $\underline{\tau}_k^s \geqslant \overline{\tau}^{(*)}$, then the branch s can be pruned.

Pruning by infeasibility: In the enumeration tree if the feasible set for one of the branches turns to be an empty set, $\mathcal{F}^i = \emptyset$, then that means that there is no feasible solution for that subproblem and in that case the corresponding branch can be pruned.

Pruning by optimality: This can be done when $\underline{\tau}_k^s \leqslant \overline{\tau}_k^s$ for some s, k. This branch can be pruned because an optimal solution has been achieved and the certificate that such solution is optimal is offered by the equality of the lower and the upper bound. In this case the solution is kept among possible solutions for the global problem.

The formal Algorithm for pruning and branching follows next.

Algorithm 6.4.10. *Pruning and branching in the enumeration tree for the scheduling problem* $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$.

Input: L, $\underline{\tau}^{(*)}$, $\overline{\tau}^{(*)}$, $\underline{\tau}_k^s$, $\overline{\tau}_k^s$.
Output: L, $\underline{\tau}^{(*)}$, $\overline{\tau}^{(*)}$.

if $\overline{\tau}^{(*)} \leqslant \underline{\tau}_k^s$ **then**
    $L \leftarrow L \setminus \{k\}$ (*Prune by bound*)
else if $\mathcal{F}_k^s = \emptyset$ **then**
    $L \leftarrow L \setminus \{k\}$ (*Prune by infeasibility*)
else if $y_{ij} \in \mathcal{F}^{(*)}$ **then**
    $\tau^{(*)} \leftarrow \tau_k^s$
    $y_{ij}^* \leftarrow y_{ij}$
    $L \leftarrow L \setminus \{k\}$ (*Prune by optimality*)
else
    Algorithm 6.4.8 (*Branching*)
end if

6.4.4 Branch and bound algorithm

The complete branch and bound algorithm is formulated next.

Algorithm 6.4.11. *Branch and bound algorithm for the scheduling problem* $1 \mid r_j, pmtn \mid \sum w_j C_j$.

Input: n, p_j, r_j, w_j.
Output: x_j^*, y_{ij}^*, $\tau^{(*)}$.

1: $\underline{\tau}^{(*)} \leftarrow \infty$
2: Solve the relaxation problem
3: **case** τ
4: Binary solution: $\overline{\tau}^{(*)} \leftarrow \tau$ (*Optimal solution*)
5: **end case**
6: $L := \{1\}$.
7: Algorithm 6.4.8 (*Branching*)

8: **while** $L \neq \emptyset$ **do**
9: **for** $k \in L$ **do**
10: Solve the relaxation problem (*Dual bound*)
11: **case** τ
12: A solution exists: $\underline{\tau}_k^s \leftarrow \tau$
13: The problem is unbounded: $\underline{\tau}_k^s \leftarrow \infty$.
14: Infeasible: $\mathcal{F}_k^s \leftarrow \emptyset$.
15: Binary solution: $\overline{\tau}_k^s \leftarrow \tau$ (*Primal bound*)
16: **end case**
17: **end for**
18: Algorithm 6.4.10 (*Pruning and branching*)
19: **end while**

20: **return** x_j^*, y_{ij}^*, $\tau^{(*)}$

6.4.5 Example 1

Example 6.2.1 is solved next using Algorithm 6.4.11 and the formulations given in Problems 6.4.6 and 6.4.7.

Solution 1: Using Problem 6.4.6. Initially the algorithm splits the problem into two subproblems. The first subproblem corresponds to $i = 1$ because there is only one job

Table 6.1: Results for Example 6.2.1 using Problem 6.4.6

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
1	1.0000	1.0000	1.0000	$y_{11} = 1$	1.0000	0		
Elapsed time is 0.005284 seconds								
1	∞	–	–	–	12.7270	1	16	32
2	12.7270	–	–	$y_{21} = 1$	21.0000	2	14	43
3	12.7270	21.0000	–	$y_{22} = 1$ $y_{32} = 1$ $y_{42} = 1$	19.0000	1	14	23
	19.0000	19.0000	19.0000	–	–	0	Optimal achieved	
Elapsed time is 0.377875 seconds								

available at that time. Thus job 1 is assigned to the machine, $y_{11}^* = 1$. The new value of $\hat{p}_1$ is 1.

The second subproblem is then solved for the times $i = 2, \ldots, 5$. The algorithm finds a lower bound, 12.7270. Two branches are created, in the first one y_{21} is fix to one, which leaves y_{22} equal to zero, with an objective value of 21.0000. In the second branch y_{22} and y_{42} are fix to one, which in turn indicates that y_{32} is one and leaving y_{21}, y_{31} and y_{41} equal to zero with a value of 19.0000. In total the algorithm with relaxation of Problem 6.4.6 required 98 iterations, or nodes explored, and it takes $0.005284 + 0.377875 = 0.383159$ seconds.

The results after running the algorithm for Example 6.2.1 using the relaxation Problem 6.4.7, including the data for solving each of the nodes, are shown Table 6.1.

Enumeration tree The tree using the branch and bound algorithm is graphically shown in Figure 6.4.

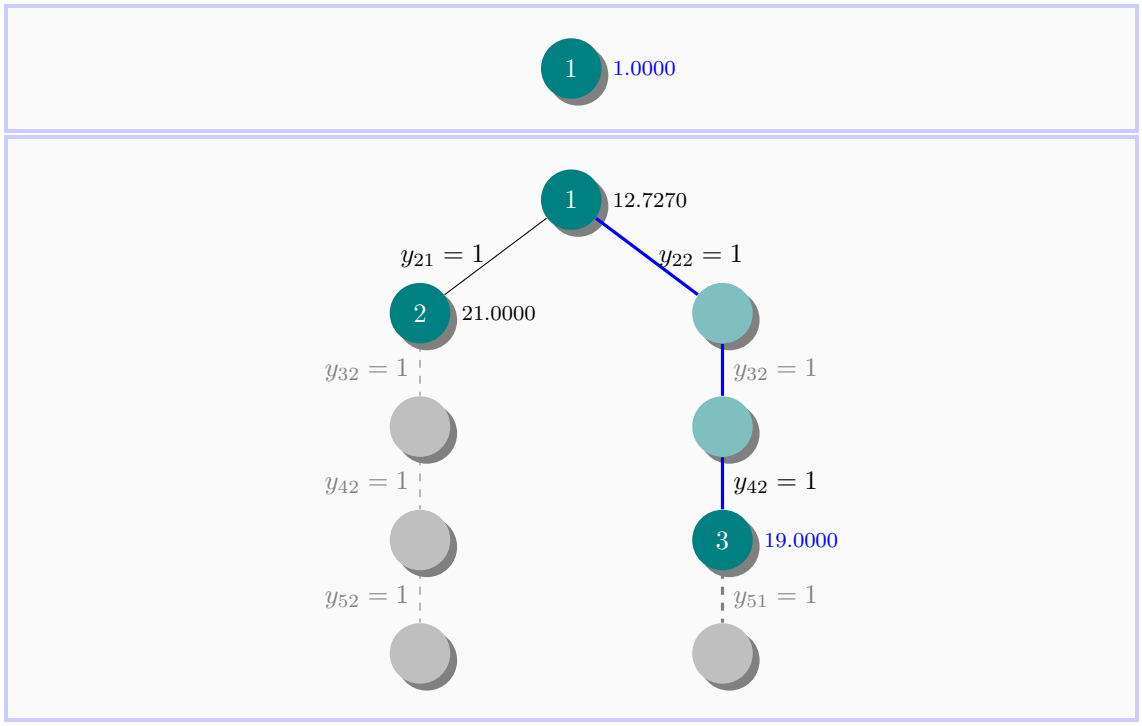


Figure 6.4: Branch and bound algorithm for Example 6.2.1 using either relaxation in Problem 6.4.6 or 6.4.7

Optimal solution The optimal solution is $1 + 19 = 20$ and is obtained with

$$y_{11}^* = y_{51}^* = 1,$$

$$y_{22}^* = y_{32}^* = y_{42}^* = 1.$$

and for the remaining variables $y_{ij}^* = 0$. Graphically the solution schedule looks like Figure 6.5.

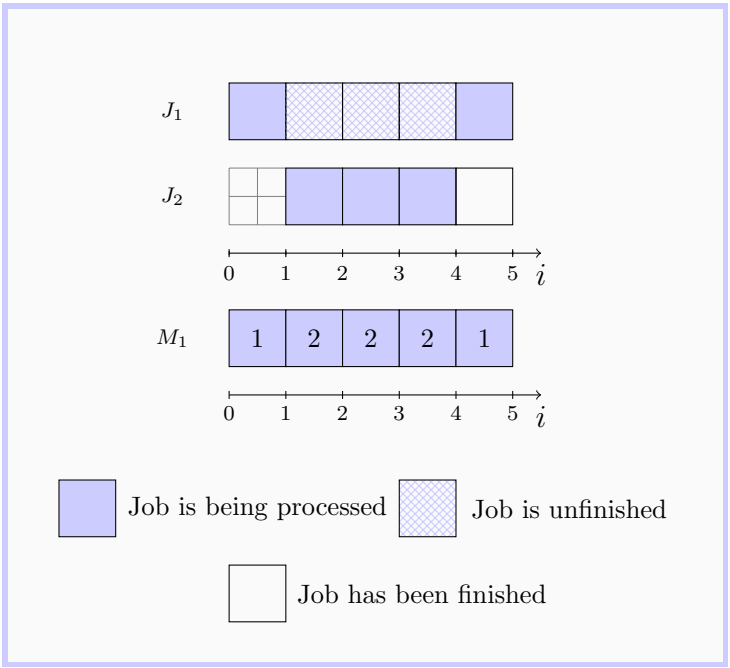


Figure 6.5: Solution schedule for Example 6.2.1 with optimal value 20

Table 6.2: Results for Example 6.2.1 using Problem 6.4.7

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
1	1.0000	1.0000	1.0000	$y_{21} = 1$	1.0000	0		
Elapsed time is 0.005351 seconds								
1	∞	–	–	–	12.7270	1	15	31
2	12.7270	–	–	$y_{21} = 1$	21.0000	2	14	42
3	12.7270	21.0000	–	$y_{22} = 1$ $y_{32} = 1$ $y_{42} = 1$	19.0000	1	14	23
	19.0000	19.0000	19.0000	–	–	0	Optimal achieved	
Elapsed time is 0.442445 seconds								

Solution 2: Using Problem 6.4.7. As in the previous case, the algorithm splits the problem into two subproblems. The approach for solving the problem is fairly similar to the one shown using Problem 6.4.6. The difference however relays on the solving time used by the algorithm when working with this relaxation. In total the algorithm with relaxation of Problem 6.4.7 required 96 iterations, or nodes explored, and it takes $0.005351 + 0.442445 = 0.447796$ seconds.

The results after running the algorithm for Example 6.2.1 using the relaxation Problem 6.4.7 are shown Table 6.2. Observe that the difference between the results in Table 6.1 and 6.2 is in the time required for the algorithm to solve the problem and the number of iterations.

Enumeration tree The tree using the branch and bound for this case is the same as shown in Figure 6.4.

Optimal solution The optimal solution is $1 + 19 = 20$ and is obtained with

$$y_{11}^* = y_{51}^* = 1,$$
$$y_{22}^* = y_{32}^* = y_{42}^* = 1.$$

and for the remaining variables $y_{ij}^* = 0$. The solution schedule was shown in Figure 6.5.

6.4.6 Example 2

Another example using Algorithm 6.4.11 with $n = 5$ jobs follows.

Example 6.4.12. *In a single machine environment $n = 5$ jobs need to be processed. The release date r_j, processing time p_j and weight w_j for each job are given in the next table.*

Job	Data		
	p_j	r_j	w_j
1	2	1	1
2	3	2	3
3	4	6	4
4	4	6	5
5	3	7	2

Solution 1: Using Problem 6.4.6. Initially the algorithm splits the problem into 3 subproblems according to the number of jobs available. There is only one job available at $i = 1$. Thus it assigns job 1 to the machine at the that time, i.e. $y^*_{11} = 1$.

After that it runs the algorithm two more times for the remaining problems. For the first problem if finds two optimal solutions with either $y^*_{21} = y^*_{32} = y^*_{42} = y^*_{52} = 1$ or $y^*_{22} = y^*_{32} = y^*_{42} = y^*_{51} = 1$ both with a value of 13. In the second problem the algorithms finds the optimal solution with $y^*_{64} = y^*_{74} = y^*_{84} = y^*_{94} = 1$, $y^*_{10\,3} = y^*_{11\,3} = y^*_{12\,3} = y^*_{13\,3} = 1$ and $y^*_{14\,5} = y^*_{15\,5} = y^*_{16\,5} = 1$. This solution has a value of 72.

In total the algorithm with relaxation of Problem 6.4.6 required 668 iterations and it takes $0.006479 + 0.352276 + 2.074337 = 2.426613$ seconds.

The results using Algorithm 6.4.11 with Problem 6.4.6 are shown in Table 6.3.

Table 6.3: Results for Example 6.2.1 using Problem 6.4.6

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
1	1.0000	1.0000	1.0000	$y_{11} = 1$	1.0000	0		
Elapsed time is 0.006479 seconds								
1	∞	–	–	–	8.3640	1	16	36
2	8.3640	–	–	$y_{21} = 1$	13.0000	1	15	42
3	8.3640	13.0000	–	$y_{22} = 1$ $y_{32} = 1$ $y_{42} = 1$	13.0000	1	15	26
	13.0000	13.0000	13.0000	–	–	0	Optimal achieved	
Elapsed time is 0.352276 seconds								
1	∞	–	–	–	32.4650	1	15	48
2	32.4650	–	–	$y_{63} = 1$	33.0990	1	14	35
3	32.4650	–	–	$y_{64} = 1$	32.4650	2	14	35
4	32.4650	–	–	$y_{64} = 1$ $y_{74} = 1$ $y_{84} = 1$ $y_{94} = 1$	50.5620	2	14	43
5	32.4650	–	–	$y_{64} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$	53.6310	3	14	55
6	33.0990	–	–	$y_{63} = 1$ $y_{73} = 1$ $y_{83} = 1$ $y_{93} = 1$	51.4030	3	14	40
7	33.0990	–	–	$y_{63} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$	54.8410	4	14	54
8	50.5620	–	–	$y_{64} = 1$ $y_{74} = 1$ $y_{84} = 1$ $y_{94} = 1$ $y_{10\,3} = 1$ $y_{11\,3} = 1$ $y_{12\,3} = 1$ $y_{13\,3} = 1$	72.0000	4	14	39
9	50.5620	72.0000	–	$y_{64} = 1$ $y_{74} = 1$ $y_{84} = 1$ $y_{94} = 1$ $y_{10\,5} = 1$ $y_{11\,5} = 1$ $y_{12\,5} = 1$	76.0000	4	14	35
10	51.4030	72.0000	–	$y_{63} = 1$ $y_{73} = 1$ $y_{83} = 1$ $y_{93} = 1$ $y_{10\,4} = 1$	76.0000	3	14	56

Continued on next page

Table 6.3: (Continued)

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
				$y_{114} = 1$ $y_{124} = 1$ $y_{134} = 1$				
11	51.4030	72.0000	–	$y_{63} = 1$ $y_{73} = 1$ $y_{83} = 1$ $y_{93} = 1$ $y_{105} = 1$ $y_{115} = 1$ $y_{125} = 1$	83.0000	3	13	36
12	53.6310	72.0000	–	$y_{64} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$ $y_{104} = 1$ $y_{114} = 1$ $y_{124} = 1$	85.0000	2	13	45
13	54.8410	72.0000	–	$y_{63} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$ $y_{103} = 1$ $y_{113} = 1$ $y_{123} = 1$	89.0000	1	13	43
	72.0000	72.0000	72.0000	–	–	0	Optimal achieved	

Elapsed time is 2.074337 seconds

Enumeration tree The tree using the branch and bound algorithm is graphically shown in Figures 6.6.

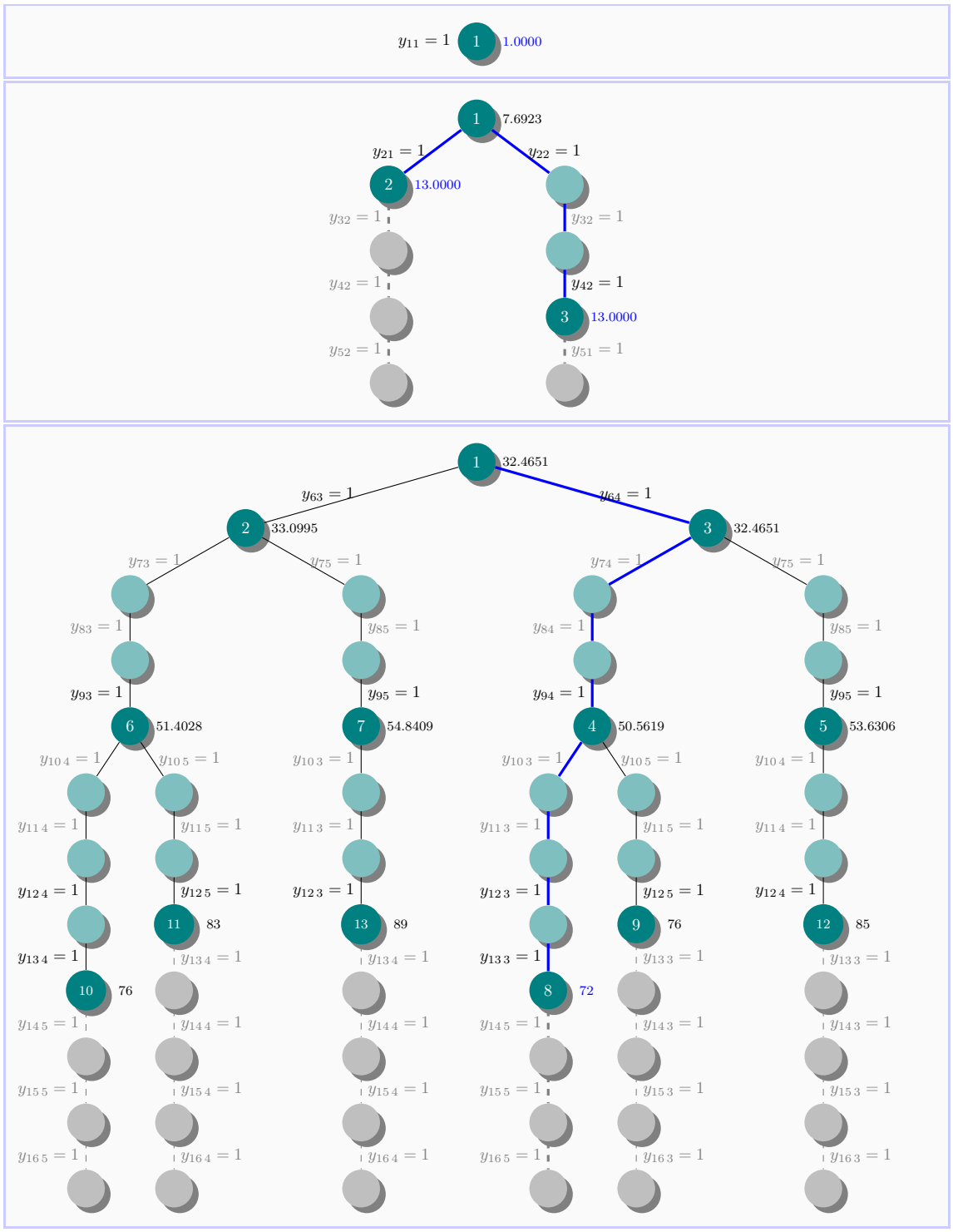


Figure 6.6: Complete tree for the branch and bound algorithm for Example 6.4.12 using either relaxation in Problem 6.4.6 or 6.4.7

Optimal solution The optimal solution is $1 + 13 + 72 = 86$ and is obtained with

$$\begin{aligned} y^*_{11} = y^*_{21} &= 1, \\ y^*_{22} = y^*_{32} = y^*_{42} &= 1, \\ y^*_{10\,3} = y^*_{11\,3} = y^*_{12\,3} = y^*_{13\,3} &= 1, \\ y^*_{64} = y^*_{74} = y^*_{84} = y^*_{94} &= 1, \\ y^*_{14\,5} = y^*_{15\,5} = y^*_{16\,5} &= 1. \end{aligned}$$

and for the remaining variables $y^*_{ij} = 0$.

Graphically the solution schedule looks like Figure 6.7.

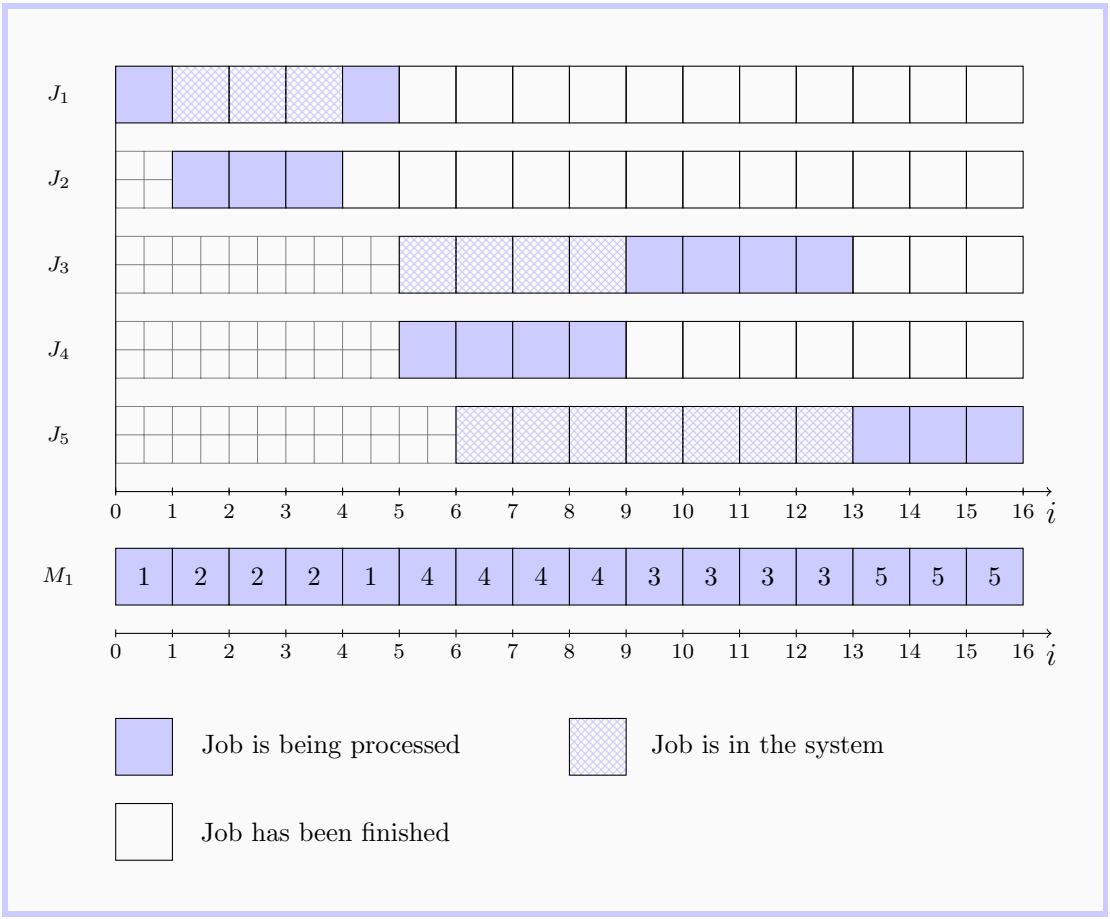


Figure 6.7: Solution schedule for Example 6.2.1 with optimal value 86

Solution 2: Using Problem 6.4.7. The results after running the algorithm for Example 6.4.12 using the relaxation Problem 6.4.7 are shown in Table 6.4. The tree using this algorithm is explored in a similar way as in the previous case. In total the algorithm with relaxation of Problem 6.4.7 required 648 iterations and it takes $0.005450+0.432528+2.553199 = 2.991177$ seconds.

Table 6.4: Results for Example 6.2.1 using Problem 6.4.7

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
1	1.0000	1.0000	1.0000	$y_{11}=1$	1.0000	0		
Elapsed time is 0.005450 seconds								
1	∞	–	–	–	8.3640	1	15	29
2	8.3640	–	–	$y_{21}=1$	13.0000	1	15	45
3	8.3640	13.0000	–	$y_{22}=1$ $y_{32}=1$ $y_{42}=1$	13.0000	1	15	24
	13.0000	13.0000	13.0000	–	–	0	Optimal achieved	
Elapsed time is 0.432528 seconds								
1	∞	–	–	–	32.4650	1	15	38
2	32.4650	–	–	$y_{63}=1$	33.0990	1	14	37
3	32.4650	–	–	$y_{64}=1$	32.4650	2	14	37
4	32.4650	–	–	$y_{64}=1$ $y_{74}=1$ $y_{84}=1$ $y_{94}=1$	50.5620	2	14	48
5	32.4650	–	–	$y_{64}=1$ $y_{75}=1$ $y_{85}=1$ $y_{95}=1$	53.6310	3	14	52
6	33.0990	–	–	$y_{63}=1$ $y_{73}=1$ $y_{83}=1$ $y_{93}=1$	51.4030	3	14	45
7	33.0990	–	–	$y_{63}=1$ $y_{75}=1$ $y_{85}=1$ $y_{95}=1$	54.8410	4	14	55
8	50.5620	–	–	$y_{64}=1$ $y_{74}=1$ $y_{84}=1$ $y_{94}=1$ $y_{103}=1$ $y_{113}=1$	72.0000	4	14	41

Continued on next page

Table 6.4: (Continued)

Node	Bounds		$\tau^{(*)}$	Subproblem		Open	Iterations	
	$\underline{\tau}^{(*)}$	$\overline{\tau}^{(*)}$		$\mathcal{F}_i$	$\underline{\tau}_{y_{ij}}$		Outer	Inner
				$y_{123} = 1$ $y_{133} = 1$				
9	50.5620	72.0000	–	$y_{64} = 1$ $y_{74} = 1$ $y_{84} = 1$ $y_{94} = 1$ $y_{105} = 1$ $y_{115} = 1$ $y_{125} = 1$	76.0000	4	14	34
10	51.4030	72.0000	–	$y_{63} = 1$ $y_{73} = 1$ $y_{83} = 1$ $y_{93} = 1$ $y_{104} = 1$ $y_{114} = 1$ $y_{124} = 1$ $y_{134} = 1$	76.0000	3	14	43
11	51.4030	72.0000	–	$y_{63} = 1$ $y_{73} = 1$ $y_{83} = 1$ $y_{93} = 1$ $y_{105} = 1$ $y_{115} = 1$ $y_{125} = 1$	83.0000	3	13	31
12	53.6310	72.0000	–	$y_{64} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$ $y_{104} = 1$ $y_{114} = 1$ $y_{124} = 1$	85.0000	2	13	44
13	54.8410	72.0000	–	$y_{63} = 1$ $y_{75} = 1$ $y_{85} = 1$ $y_{95} = 1$ $y_{103} = 1$ $y_{113} = 1$ $y_{123} = 1$	89.0000	1	13	45
	72.0000	72.0000	72.0000	–	–	0	Optimal achieved	

Elapsed time is 2.553199 seconds

Enumeration tree The tree using the branch and bound algorithm is the same graphically shown in Figure 6.6.

Optimal solution The optimal solution is $1 + 13 + 72 = 86$ and is obtained with

$$\begin{aligned} y^*_{11} = y^*_{21} &= 1, \\ y^*_{22} = y^*_{32} = y^*_{42} &= 1, \\ y^*_{10\,3} = y^*_{11\,3} = y^*_{12\,3} = y^*_{13\,3} &= 1, \\ y^*_{64} = y^*_{74} = y^*_{84} = y^*_{94} &= 1, \\ y^*_{14\,5} = y^*_{15\,5} = y^*_{16\,5} &= 1. \end{aligned}$$

and for the remaining variables $y^*_{ij} = 0$.

Graphically the solution schedule is exactly the same as the one in Figure 6.7.

6.5 Quality of the relaxations and the branch and bound algorithm

6.5.1 Generation of test problems

In order to test the problem formulations and due to the lack of standard benchmark problems, a similar technique to randomly generate instances of the problems used by van den Akker et al. (2010) with an adaptation for this particular scheduling problem will be used.[5] In this case each problem generated have a number of jobs n from the

[5] In their approach they generate instances for the scheduling problem $1 \,|\, r_j, p_j = p \,|\, \sum w_j T_j$. The number of jobs is selected from the set $\{70, 80, 90, 100\}$ and p_j from $\{5, 10, 15, 20, 25, 30\}$. Then for each combination of the parameters they create 50 instances

- r_j is randomly selected from the interval $[0, (n-6)p)$.
- w_j is randomly selected from the interval $[0, 120)$.

Table 6.5: Sets for the randomly generated data for p_j, r_j and w_j

Group	n	$\max k_i$	data			Number of problems
			p_j	r_j	w_j	
1	10	2	{5,10,15}	[0,(n-3)20)	[0,60)	40
2		3				40
3		4				40
4		5				40
5		6				40
6		7				40
7		8				32
8		9-10				10
9	70	2-7	{5,10,15}	[0,(n-6)27)	[0,60)	40

set $J = \{10, 70\}$. The randomised problems are classified into groups according to the maximum number of jobs at the time per problem, denoted with $\max k_i$, ranging from 2 to 10 jobs at the time, as shown in Table 6.5. The values of p_j, r_j and w_j for each case are randomly generated in accordance with the sets given in the same table.

Thus following this generation model a total of 322 problems were developed using MATLAB. The results obtained after solving the problems are explained in the next section.

6.5.2 Numerical experiments

All the experiments were run in an AMD Athlon(tm) 64 processor, 3500+, 2.21 GHz and 1.00 GB of RAM. The solution for Problem 6.4.5 (LPr) is obtained with SeDuMi (Sturm (1999)). Problems 6.4.6 (LPr-2) and 6.4.7 (SDPr-2) are solved using PENBMI (Kočvara and Stingl (2003)). In all cases YALMIP (Löfberg (2004)) for MATLAB is also used.

- d_j is randomly selected from the interval $[r_j + p, (n-5)p)$.

.

After solving each instance created, the information that was gathered is explained here. Observe that for each particular group of problems different parameters are calculated from the ones given below.

Largest subproblem: It is the subproblem after the partition with the maximum number of jobs at any time.

(No) Indicates the problem run.

($\max k_i$**)** Indicates the maximum number of jobs that are available in the system at any time.

(m**)** Length of the schedule for the largest subproblem.

($|\hat{S}|$**)** Size of the set $\hat{S}$ for the largest subproblem.

(Time (s)) Time in seconds spent solving the largest subproblem.

(LPr**)** Time using the linear programming relaxation given in Problem 6.4.5.

(LPr**-2)** Time using the linear programming relaxation given in Problem 6.4.6.

(SDPr**)** Time using the semidefinite programming relaxation given in Problem 6.4.7.

Main problem Includes the complete information for all the subproblems.

(ET) Recall that a new enumeration tree is created after those times at which the machine can finished all the available jobs in the system. Thus, ET gives the total number of enumeration trees or subproblems actually solved.

(Nodes or N) It is calculated by adding the nodes for each tree when solving the problem.

(LPr**)** Information for the main problem using the linear programming relaxation given in Problem 6.4.5.

(s) Solving time in seconds for the whole problem.

(LPr-2) Information for the main problem using the linear programming relaxation given in Problem 6.4.6.

(var) Total number of variables.

(cons) Total number of constraints.

(iter) Total number of iterations required to solve all the nodes.

(s) Solving time in seconds for the whole problem.

(SDPr) Information for the main problem using the semidefinite programming relaxation given in Problem 6.4.7.

(var) Total number of variables.

(cons) Total number of constraints.

(iter) Total number of iterations required to solve all the nodes.

(s) Solving time in seconds for the whole problem.

(Gap) Gap between the solution and the actual optimal solution. In fact all the problem were solved to optimality. This information is included only to emphasise this aspect.

Group 1: $n = 10$**,** $\max k_i = 2$

For the case when $n = 10$ and the maximum number of available jobs at any time, for the length of the schedule, is equal to 2, relaxations LPr-2 and SDPr are used only. For both cases the number of variables, constraints and iterations for the algorithm are calculated. The results for this case are presented in Table 6.6. The solving time is graphically compared in Figure 6.8. The number of variables for both cases is compared in Figure 6.9. The number of constraints is illustrated in Figure 6.10 and the number iterations in Figure 6.11. From the results the following is concluded.

(i) When the maximum number of jobs available at any time is 2, LPr-2 performs better than SDPr in 87.5% (35 out of 40) of the cases.

(ii) The number of variables is slightly bigger for SDPr than for LPr-2 and increases with the problem size. The difference between both relaxations varies from 2 to 14 variables bigger for SDPr.

(iii) The number of constraints is bigger when using SDPr. However the maximum size of the matrix constraints for SDPr is 2×2. The maximum size of the matrix constraints for LPr-2 is 4×4. Additionally, the number of constraints increases, in both cases, with the problem size.

(iv) The number of iterations is almost identical for both relaxations and increases with the problem size.

(v) The length of the schedule m is not directly related with the solving time. For example problem 13 with $m = 20$ was solved faster than problem 35 with $m = 7$.

Table 6.6: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 2$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem								
	m	$\lvert\hat{S}\rvert$	Time (s)		ET	Nodes	LPr-2				SDPr			
			LPr-2	SDPr			var	cons	iter	(s)	var	cons	iter	(s)
1	8	3	0.40	0.45	10	12	49	78	48	0.43	51	117	48	0.48
2	7	3	0.38	0.46	10	12	49	78	53	0.41	51	117	53	0.48
3	12	3	0.49	0.53	10	12	49	78	47	0.52	51	117	47	0.56
4	15	3	0.46	0.52	10	12	49	78	52	0.48	51	117	53	0.56
5	13	3	0.42	0.56	10	12	49	78	50	0.48	51	117	50	0.60
6	15	3	0.41	0.43	9	13	98	156	112	0.85	101	234	111	0.87
7	19	3	0.42	0.45	10	14	98	156	88	0.84	102	234	83	0.88
8	13	3	0.40	0.44	10	14	98	156	116	0.82	102	234	115	0.90
9	8	3	0.39	0.44	10	14	98	156	97	0.83	102	234	96	0.90
10	15	3	0.48	0.44	9	13	98	156	121	0.92	102	234	128	0.90
11	14	3	0.46	0.46	10	14	98	155	113	0.86	102	234	119	0.94
12	13	3	0.40	0.48	10	14	98	156	95	0.80	102	234	94	0.95
13	20	3	0.41	0.49	9	13	98	156	120	1.11	102	234	120	0.95
14	20	3	0.39	0.50	10	14	98	156	107	0.80	102	234	112	0.95
15	12	3	0.47	0.52	10	14	98	156	95	0.89	102	234	81	0.99
16	7	3	0.45	0.58	10	14	98	156	108	0.86	102	234	110	1.08
17	17	3	0.39	0.41	9	15	109	194	135	1.10	123	285	132	1.13
18	12	3	0.40	0.42	10	16	128	213	150	1.15	138	318	150	1.19
19	20	2	0.39	0.34	8	14	128	214	173	1.25	138	318	170	1.21
20	20	3	0.38	0.42	9	15	128	214	125	1.14	138	318	122	1.21
21	20	3	0.41	0.44	9	15	128	214	125	1.22	138	318	122	1.24
22	9	3	0.39	0.43	10	16	147	234	163	1.20	153	351	165	1.28
23	11	3	0.39	0.43	10	16	147	234	140	1.25	153	351	137	1.28
24	6	3	0.38	0.43	10	16	147	234	141	1.15	153	351	132	1.29
25	16	3	0.40	0.42	10	16	147	234	163	1.24	153	351	168	1.31
26	11	3	0.42	0.42	10	16	147	234	186	1.23	153	351	185	1.31
27	13	3	0.42	0.44	10	16	147	234	160	1.19	153	351	160	1.32
28	25	3	0.41	0.45	9	15	147	234	142	1.23	153	351	142	1.34
29	12	3	0.40	0.48	9	15	128	214	155	1.24	138	318	156	1.34
30	24	3	0.60	0.52	10	16	147	234	169	1.47	153	351	174	1.39
31	13	3	0.40	0.52	10	16	147	234	138	1.20	153	351	127	1.40
32	28	3	0.50	0.49	10	16	147	234	175	1.54	153	351	176	1.40
33	20	3	0.41	0.48	10	15	147	234	159	1.18	153	351	158	1.42
34	23	3	0.41	0.49	10	16	147	234	174	1.21	153	351	172	1.45
35	7	3	0.49	0.57	10	16	147	234	176	1.34	153	351	176	1.52
36	6	3	0.37	0.58	9	16	146	234	170	1.18	153	351	170	1.64
37	16	3	0.39	0.78	9	15	147	234	182	1.19	153	351	187	1.82
38	25	3	0.49	0.64	9	17	196	312	217	1.74	204	468	220	2.04
39	15	3	0.49	0.65	8	16	196	311	213	1.68	204	468	211	2.06
40	22	3	0.54	0.59	9	19	226	370	283	2.32	240	552	284	2.31
Av	**15**	**3**	**0.43**	**0.49**	**10**	**15**	**122**	**197**	**136**	**1.09**	**129**	**296**	**136**	**1.20**

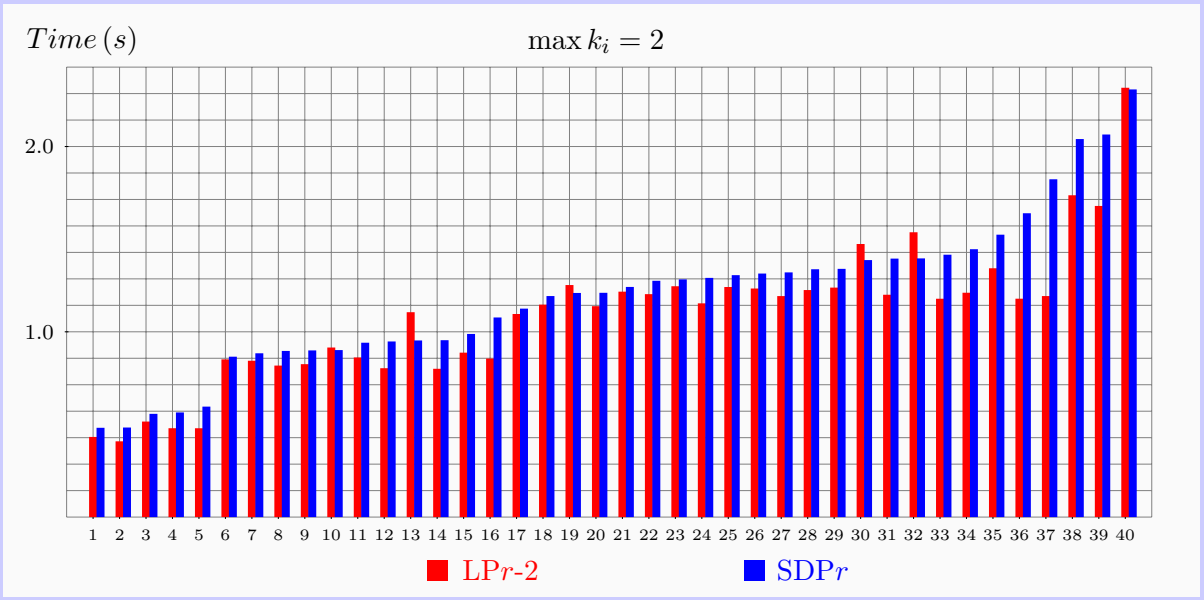


Figure 6.8: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 2$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

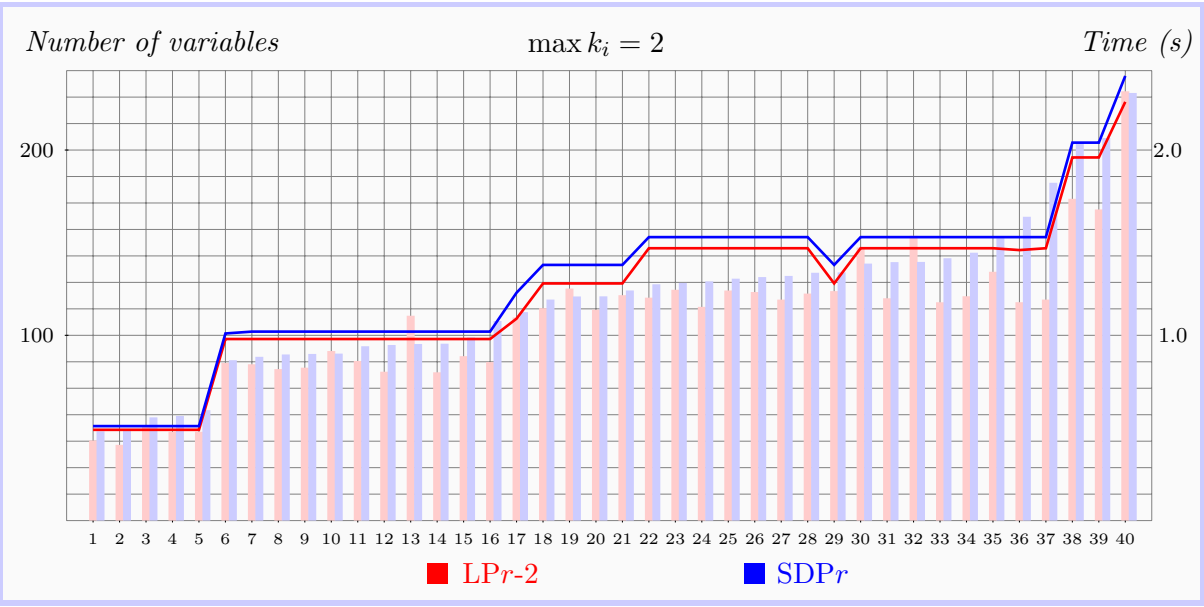


Figure 6.9: Comparison of the number of variables between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 2$. Note that the scale of solving time is different from the one indicating the number of variables

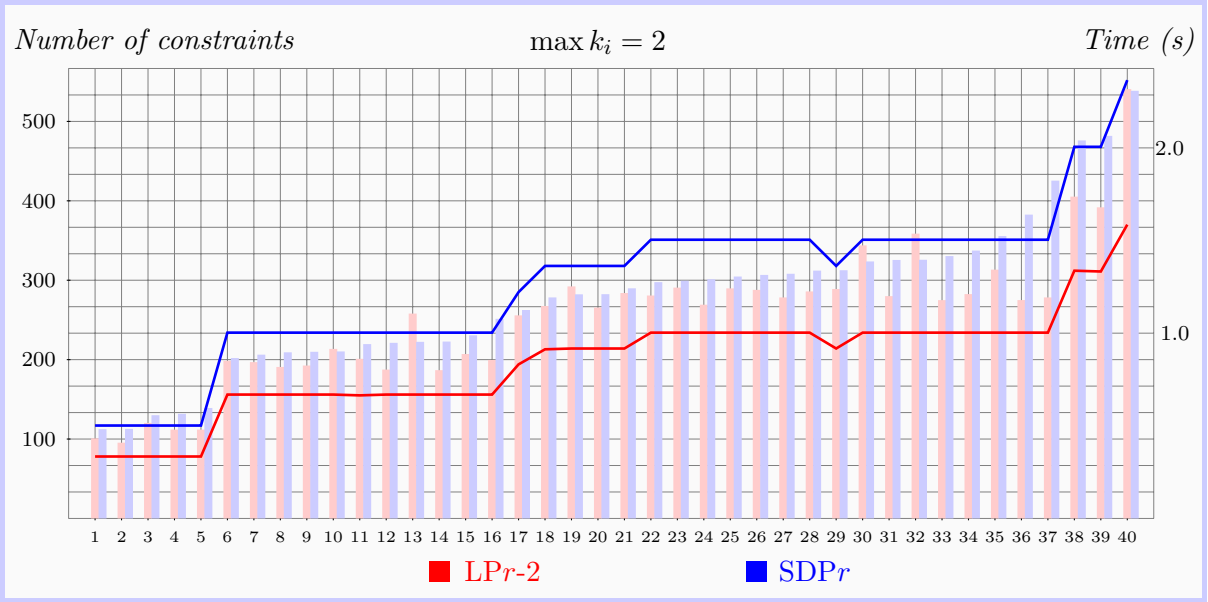


Figure 6.10: Comparison of the number of constraints between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 2$. Note that the scale of solving time is different from the one indicating the number of constraints

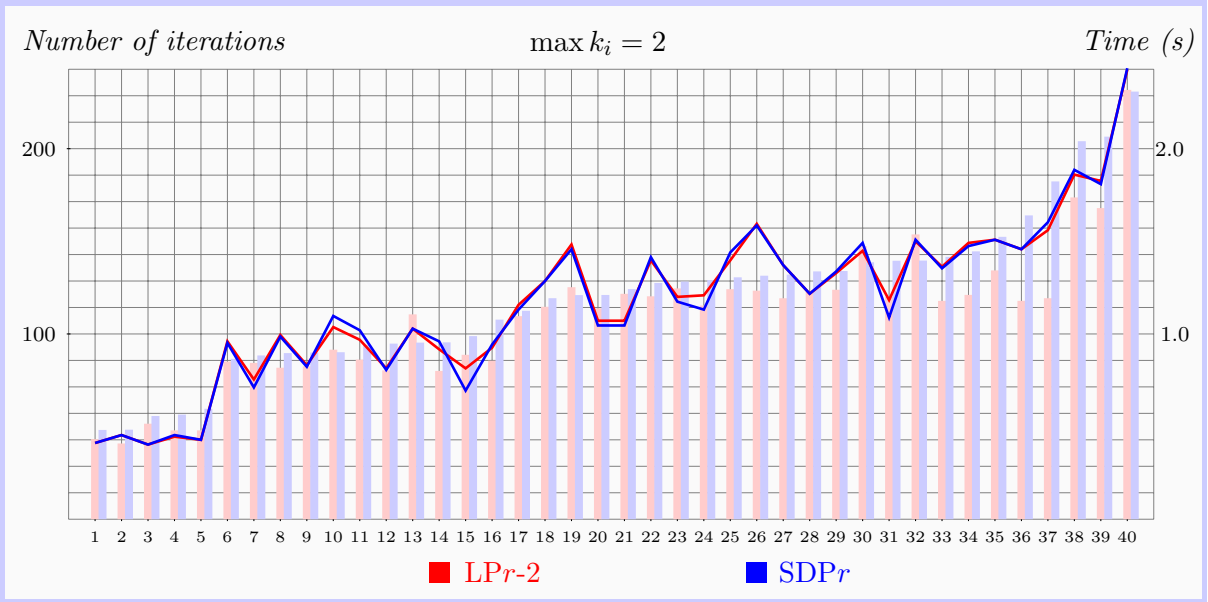


Figure 6.11: Comparison of the number of iterations between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 2$. Note that the scale of solving time is different from the one indicating the number of iterations

Group 2: $n = 10$**,** $\max k_i = 3$

The problems in this group were solved using LPr-2 and SDPr only. The results for $n = 10$ when the maximum number of available jobs is 3 are presented in Table 6.7. The solving time is graphically compared in Figure 6.12. The number of variables is illustrated in Figure 6.13. The number of constraints is included in Figure 6.14 and the number iterations in Figure 6.15. From the results when the maximum number of jobs at the time is equal to 3, the following is concluded.

(i) LPr-2 performs better than SDPr in 90% (36 out of 40) of the problems.

(ii) The number of variables is higher when using LPr-2 and increases for both cases with the problem size. The difference varies between 20 (problem 3) to 458 (problem 40).

(iii) The number of constraints is higher in SDPr. However the maximum size of the matrix constraints for SDPr is 2×2. The maximum size of the matrix constraints for LPr-2 is 9×9. The number of constraints increases with the problem size.

(iv) The number of iterations is relatively the same independently of the relaxation used, although slightly bigger for SDPr. The number of iterations increases with the problem size.

(v) The length of the schedule m is not directly related with the solving time. For example problem 2 with $m = 28$ was solved faster than problem 28 with $m = 17$.

(vi) The size of the set $\hat{S}$ seems to be smaller for problems with less solving time although for this group of problems this characteristic is not conclusive.

Table 6.7: Results using Algorithm 6.4.10 for 40 problems with $n = 10$ and $\max k_i = 3$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem								
	m	$\lvert\hat{S}\rvert$	Time (s)		ET	Nodes	LPr-2				SDPr			
			LPr-2	SDPr			var	cons	iter	(s)	var	cons	iter	(s)
1	21	4	1.00	1.12	8	19	256	389	272	1.95	264	577	267	2.34
2	28	6	1.32	1.68	9	16	308	359	233	1.89	258	547	246	2.49
3	11	5	1.27	1.53	9	18	284	373	211	1.87	264	561	218	2.35
4	20	6	1.51	1.79	9	19	373	444	247	2.49	321	672	265	2.60
5	18	7	2.06	2.62	9	18	530	514	204	2.36	390	790	206	2.76
6	19	7	2.36	2.84	9	19	562	545	242	2.50	414	837	252	2.87
7	34	6	1.38	1.87	8	17	357	437	272	2.57	309	664	283	3.06
8	23	5	1.17	1.57	9	19	354	456	254	2.16	324	688	262	3.13
9	24	6	1.62	2.06	9	21	422	522	311	2.60	372	790	323	3.19
10	25	6	1.12	1.48	9	17	357	437	272	2.11	309	664	278	3.15
11	16	7	2.28	2.69	9	20	579	592	345	2.90	441	907	346	3.24
12	12	6	1.65	2.24	9	21	422	522	324	2.69	372	789	328	3.32
13	23	7	2.04	2.48	9	20	583	612	274	2.50	450	938	276	3.61
14	18	7	2.00	2.66	9	20	594	618	284	2.96	456	947	298	3.65
15	18	7	2.77	3.18	9	23	729	728	302	3.17	546	1117	314	3.66
16	29	7	2.70	2.71	8	21	628	670	345	3.49	492	1024	356	3.69
17	25	5	2.30	2.80	9	25	506	646	406	3.15	468	967	411	3.69
18	20	6	2.45	2.83	9	24	607	709	387	3.47	516	1074	391	3.77
19	22	7	2.45	2.93	9	23	750	762	300	3.22	567	1169	306	3.89
20	18	7	2.53	2.95	9	23	647	694	364	3.29	510	1059	373	3.90
21	27	6	2.25	2.62	9	21	422	522	350	3.04	372	789	363	4.08
22	28	5	1.69	1.97	8	24	612	737	437	4.33	534	1115	447	4.18
23	27	8	4.21	3.79	9	23	959	843	371	4.65	645	1302	382	4.29
24	34	7	2.81	3.00	8	23	882	874	343	4.12	660	1342	351	4.35
25	25	7	2.96	2.93	9	26	797	856	426	4.27	627	1309	422	4.45
26	16	6	1.70	2.00	7	25	697	810	450	3.59	591	1227	467	4.57
27	22	8	3.24	3.44	9	27	1004	967	463	4.38	720	1485	475	4.75
28	17	6	2.17	2.50	8	26	769	893	497	4.08	648	1357	514	5.06
29	24	7	2.27	2.42	8	27	903	958	509	4.27	711	1462	522	5.34
30	24	8	3.42	3.97	9	27	1023	987	480	4.61	735	1518	493	5.46
31	24	8	3.49	3.65	7	26	1184	1112	497	5.04	840	1714	517	5.48
32	22	7	2.51	2.78	8	29	1128	1130	666	6.23	849	1732	677	5.81
33	28	8	2.96	3.17	8	28	1242	1154	525	5.18	876	1774	539	5.88
34	21	7	2.64	2.80	8	31	1088	1145	624	5.19	855	1747	626	6.00
35	23	8	3.27	3.62	8	29	1410	1287	562	5.48	981	1988	580	6.18
36	20	6	2.23	2.85	7	33	1169	1254	667	5.47	930	1911	667	6.51
37	25	7	3.11	2.71	7	33	1295	1343	643	7.35	1008	2053	659	6.51
38	29	8	3.40	3.60	8	31	1510	1386	593	5.94	1001	2140	607	6.82
39	30	8	2.91	3.39	8	29	1631	1409	571	6.42	1083	2184	598	7.20
40	25	7	2.67	2.88	6	37	1751	1702	799	7.51	1293	2614	821	8.04
Av	**23**	**7**	**2.35**	**2.65**	**8**	**24**	**783**	**810**	**408**	**3.86**	**600**	**1239**	**418**	**4.38**

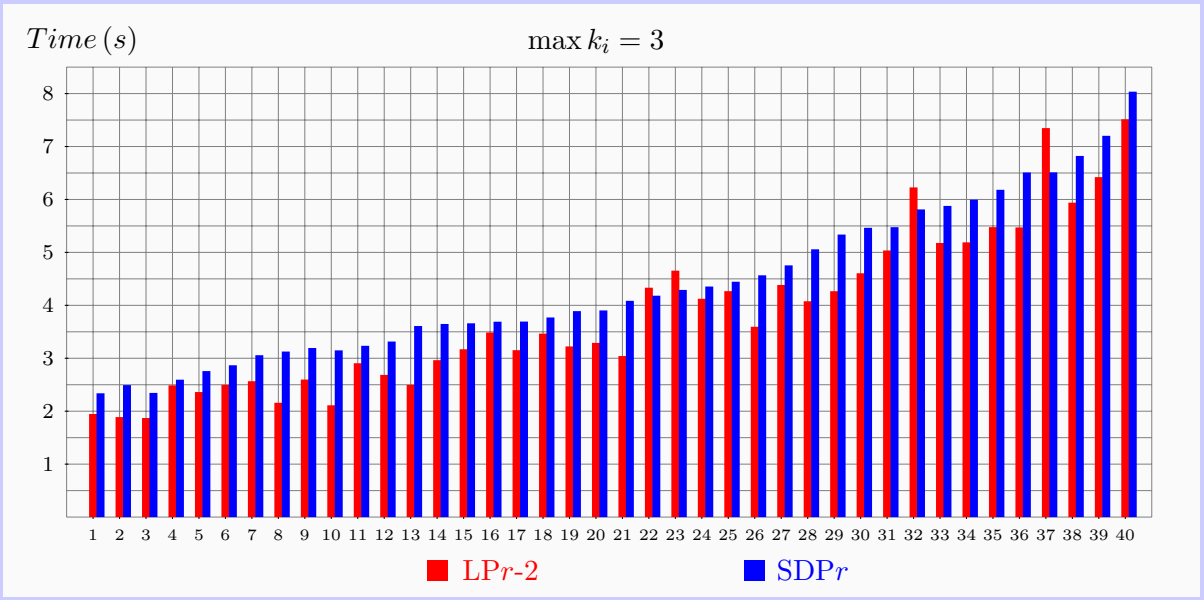


Figure 6.12: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 3$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

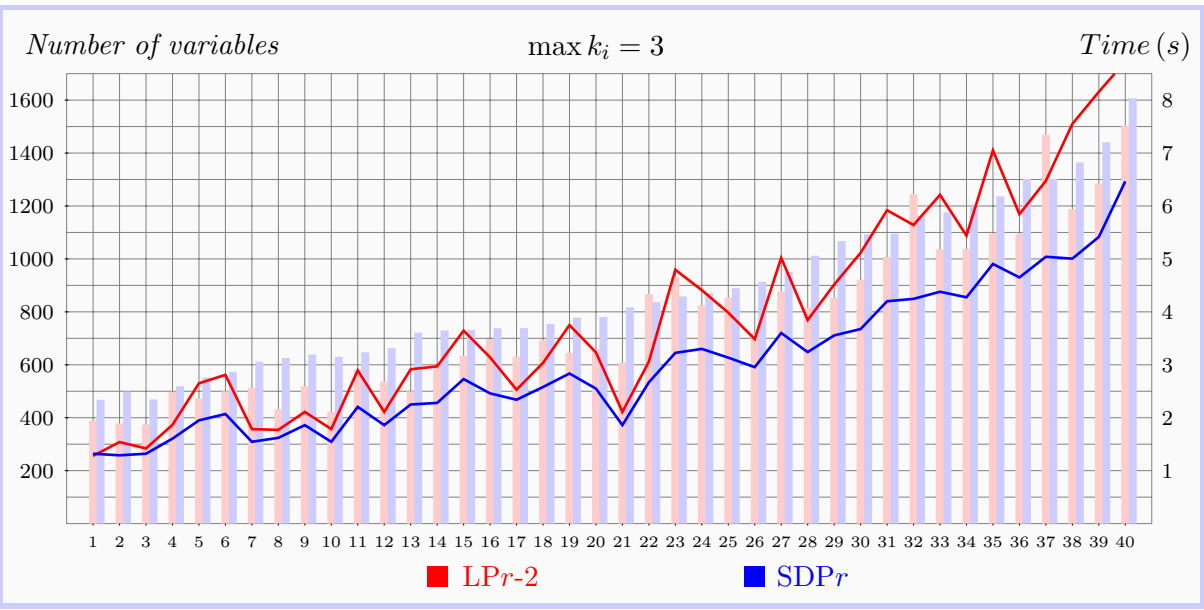


Figure 6.13: Comparison of the number of variables between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 3$. Note that the scale of solving time is different from the one indicating the number of variables

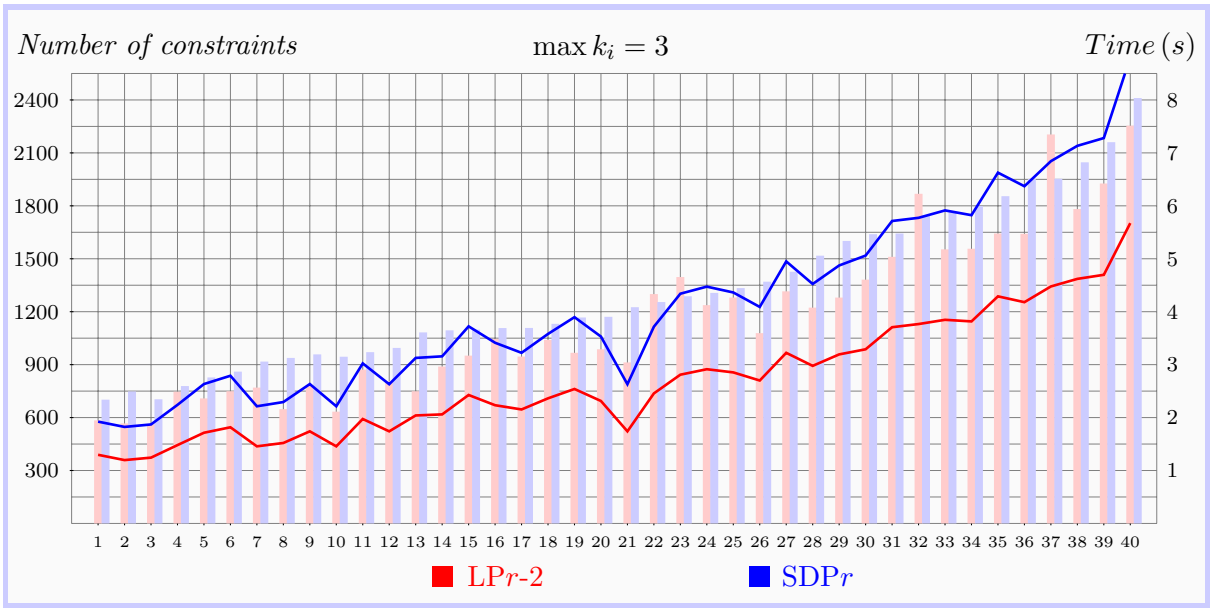


Figure 6.14: Comparison of the number of constraints between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 3$. Note that the scale of solving time is different from the one indicating the number of constraints

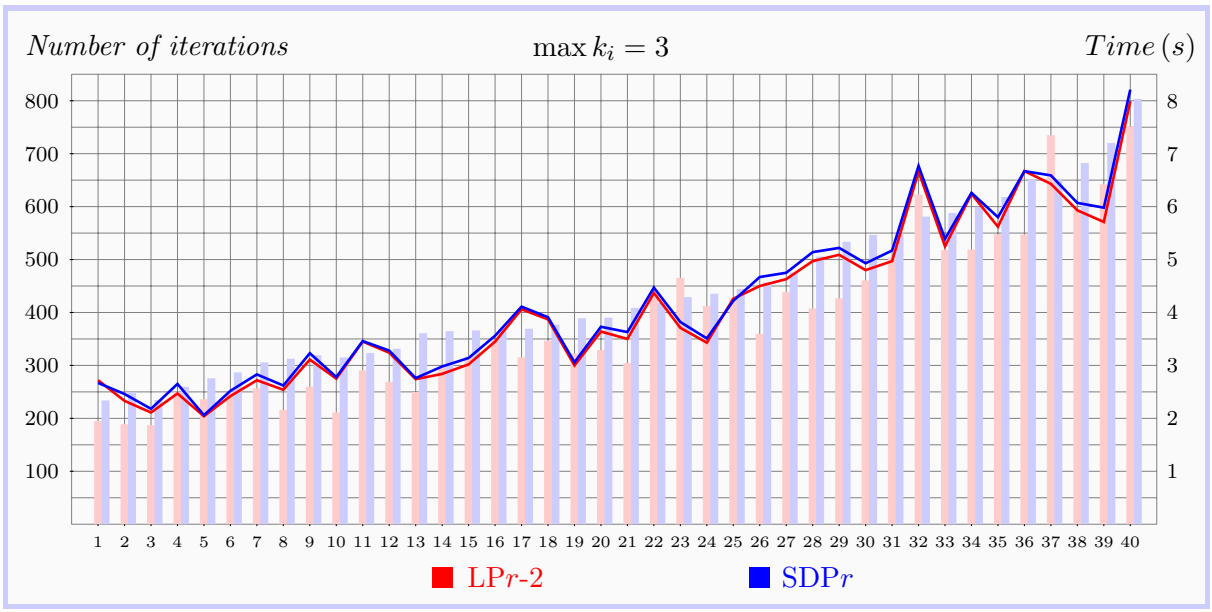


Figure 6.15: Comparison of the number of iterations between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 3$. Note that the scale of solving time is different from the one indicating the number of iterations

Group 3: $n = 10$**,** $\max k_i = 4$

The results for $n = 10$ when the maximum number of available jobs at any time is equal to 4 are presented in Table 6.8. This group was solved using relaxations LPr-2 and SDPr. The solving time is graphically compared in Figure 6.16. The number of variables is represented in Figure 6.17. The number of constraints is compared in Figure 6.18 and the number iterations in Figure 6.19.

From the results when the maximum number of jobs at the time is equal to 4 the following is concluded.

(i) For this group of problems SDPr performs better than LPr-2 in 95% (38 out of 40) of the cases. This result may obey to the sizes of the matrix variables for LPr-2.

(ii) The difference in the number of variables between both relaxations increases dramatically from the previous case. It changes between 246 (problem 2) to 3682 (problem 38).

(iii) The number of constraints is higher for SDPr. The worst difference is up to 2413 constraints more for SDPr (problem 40). However the size of matrix constraints for SDPr is 2×2 whereas for LPr-2 is in the worst case 15×15.

(iv) The number iterations is bigger for SDPr than for LPr-2, but relatively the same.

(v) The length of the schedule m is not directly related with solving time. In problem 39, for example, with $m = 34$ the solving time with both relaxations was higher than problem 1 with $m = 48$.

(vi) The solving time increases with the value of $|\hat{S}|$. It changes between 9 for problems with shorter solving times and 14 for those with larger solving time.

Table 6.8: Solving time comparison for 40 problems with $n = 10$ and a $\max k_i = 4$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem								
	m	$\lvert\hat{S}\rvert$	Time (s)		ET	N	LPr-2				SDPr			
			LPr-2	SDPr			var	cons	iter	(s)	var	cons	iter	(s)
1	48	9	6.40	5.09	8	27	1173	1082	571	7.04	834	1648	598	5.79
2	46	8	8.59	5.47	8	28	1088	1107	613	9.37	842	1687	616	6.48
3	47	10	18.25	6.51	8	27	1595	1281	538	18.28	1013	1969	573	6.54
4	34	10	8.93	7.68	8	28	2051	1565	526	8.95	1263	2417	552	7.71
5	35	9	12.53	6.93	7	33	1747	1563	778	13.32	1220	2389	803	7.90
6	42	10	7.58	6.03	7	31	1962	1644	617	9.17	1275	2529	646	8.10
7	33	10	8.76	7.34	8	32	2087	1698	693	9.57	1344	2612	716	8.29
8	37	9	10.18	7.89	8	37	1934	1732	733	10.97	1352	2650	765	8.88
9	32	10	5.94	6.08	6	36	2235	1923	776	8.39	1496	2954	807	9.22
10	34	10	25.05	8.78	8	32	2334	1834	659	25.47	1475	2764	690	9.28
11	25	9	7.38	8.47	7	43	2094	1971	853	8.31	1530	2997	869	9.51
12	43	12	15.76	9.51	8	37	3170	2234	711	16.19	1795	3460	760	10.01
13	33	11	11.38	9.73	8	36	2720	2037	756	11.81	1635	3149	782	10.21
14	32	11	24.29	8.00	7	41	2920	2324	950	26.87	1850	3584	997	10.80
15	25	8	7.48	6.58	4	52	2591	2501	1161	12.19	1974	3890	1203	11.20
16	32	11	12.22	8.20	7	41	3419	2568	844	15.14	2048	3980	880	11.53
17	34	12	23.95	10.07	7	37	3229	2291	804	25.42	1831	3549	838	11.80
18	29	11	19.76	11.06	8	36	3641	2481	786	20.71	2014	3862	811	12.00
19	31	11	15.18	7.54	6	40	3358	2562	1051	21.92	2049	3951	1109	12.07
20	39	12	18.19	11.23	8	40	3260	2346	897	19.00	1874	3628	943	12.35
21	31	11	12.93	10.06	7	49	3436	2714	1177	15.90	2151	4186	1224	13.04
22	32	13	25.59	12.22	8	39	4242	2716	860	26.39	2203	4231	909	13.18
23	31	13	20.52	13.32	8	42	5217	3259	718	20.91	2676	5096	743	13.81
24	45	11	22.24	13.75	7	55	4425	3343	1280	23.03	2691	5158	1324	14.70
25	29	13	28.86	14.36	8	45	5308	3274	903	29.26	2661	5118	934	14.85
26	45	10	11.57	8.53	6	54	3780	3150	1274	18.71	2504	4843	1322	14.98
27	39	13	19.68	12.14	7	46	5130	3348	942	22.16	2725	5243	990	15.08
28	32	12	24.97	14.91	8	56	4820	3354	1160	25.42	2702	5194	1209	15.39
29	36	12	39.44	12.59	7	41	3563	2555	1105	42.03	2032	3967	1147	15.42
30	51	13	22.39	12.92	7	46	5432	3471	911	24.57	2816	5432	944	15.48
31	37	13	34.07	13.30	7	47	4698	3128	1139	36.08	2508	4860	1198	16.03
32	43	11	38.81	8.42	6	51	4675	3492	1252	49.79	2816	5409	1314	16.58
33	35	12	27.12	15.99	8	57	5232	3631	1257	27.85	2933	5626	1314	16.89
34	31	14	54.70	14.88	6	42	5753	3429	1052	57.15	2800	5367	1094	17.89
35	29	14	34.86	17.41	8	48	6766	3924	1021	35.25	3240	6143	1070	17.91
36	38	12	27.82	14.81	7	54	5381	3763	1354	31.28	3047	5850	1427	18.23
37	43	14	108.83	15.74	7	48	6506	3882	1154	111.52	3181	6081	1222	18.64
38	49	14	44.45	18.85	8	46	6950	3945	998	44.87	3268	6199	1041	19.41
39	34	13	41.22	16.09	6	60	6187	4141	1433	44.03	3336	6430	1506	20.00
40	39	12	22.06	12.42	6	62	6067	4373	1532	37.51	3538	6786	1584	20.85
Av	**37**	**11**	**23.25**	**10.77**	**7**	**43**	**3804**	**2691**	**946**	**25.54**	**2164**	**4172**	**987**	**12.95**

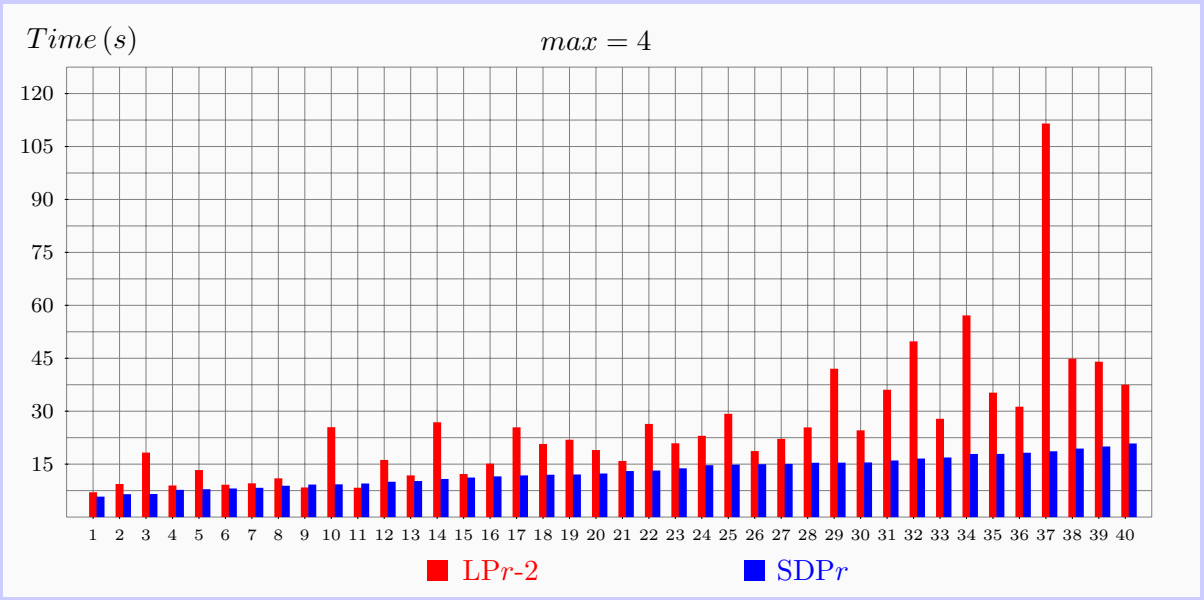


Figure 6.16: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 4$ jobs at the time using Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr)

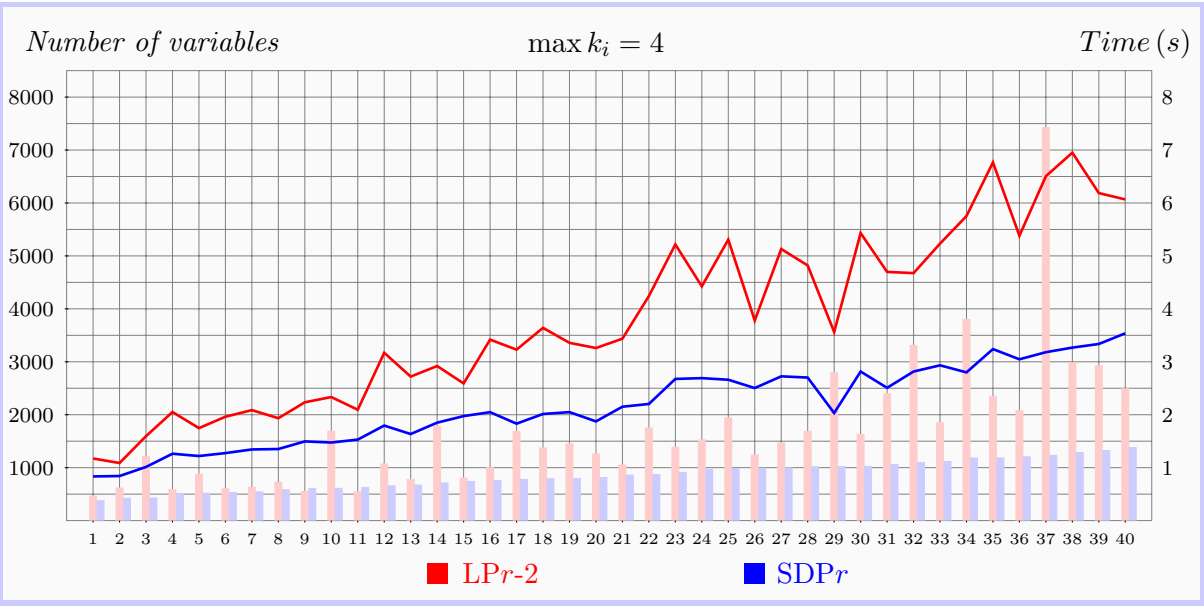


Figure 6.17: Comparison of the number of variables between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 4$. Note that the scale of solving time is different from the one indicating the number of variables

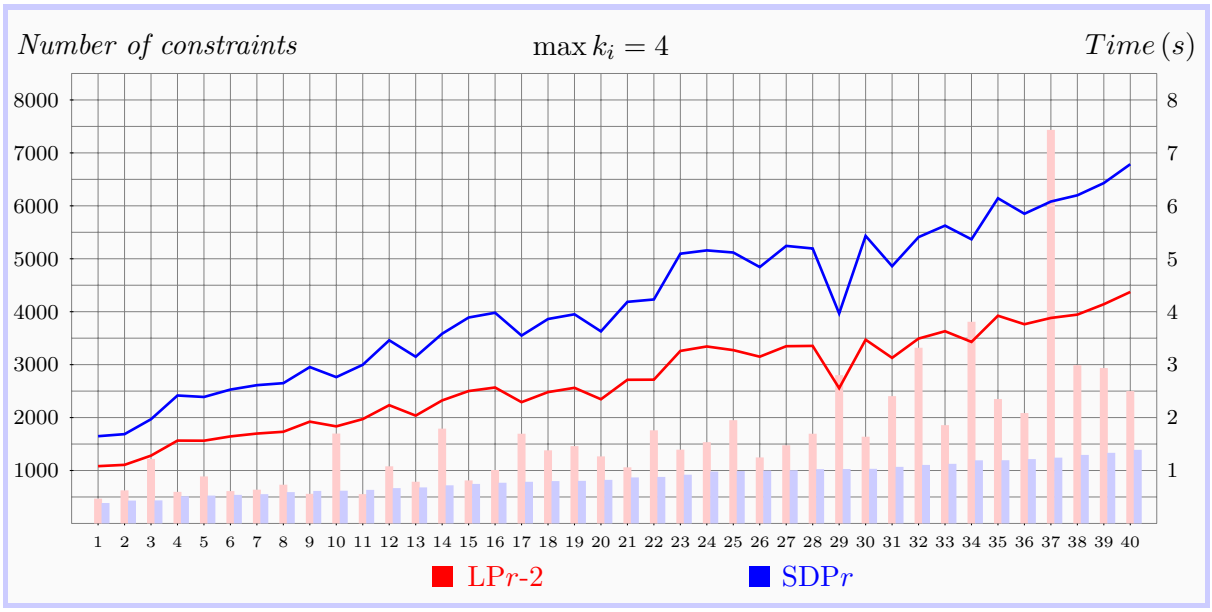


Figure 6.18: Comparison of the number of constraints between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 4$. Note that the scale of solving time is different from the one indicating the number of constraints

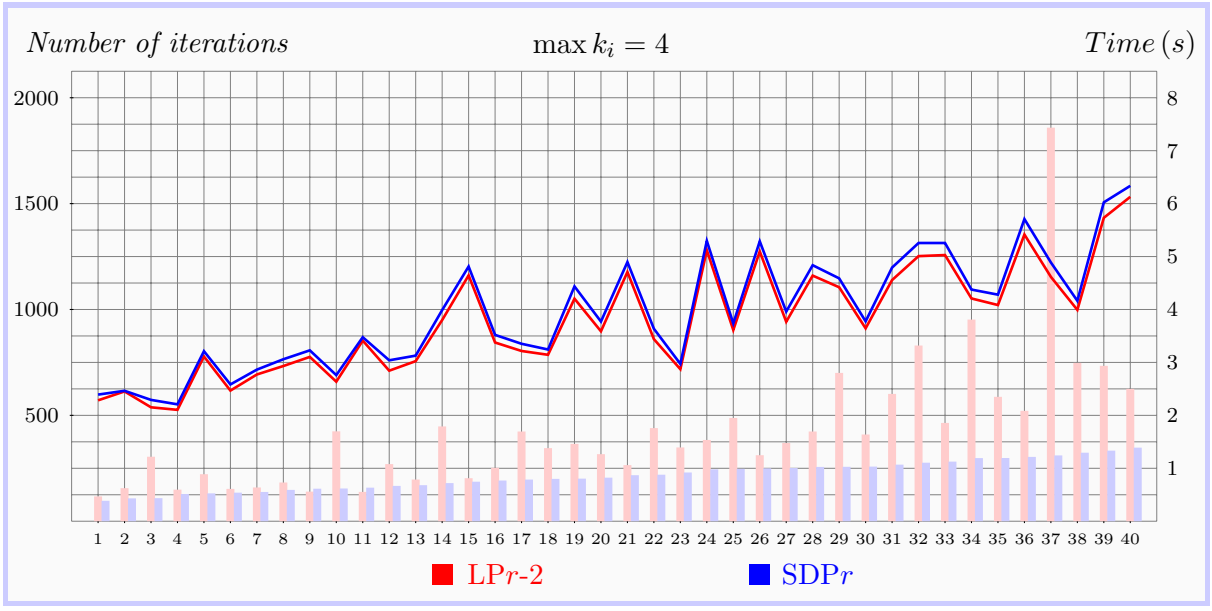


Figure 6.19: Comparison of the number of iterations between Relaxation 6.4.6 (LPr-2) and Relaxation 6.4.7 (SDPr) with $\max k_i = 4$. Note that the scale of solving time is different from the one indicating the number of iterations

Group 4: $n = 10$**,** $\max k_i = 5$

In order to solve the problems when the maximum number of available jobs at any time is 5 and $n = 10$ the three relaxations, namely LPr, LPr-2 and SDPr are used. The results are shown in Table 6.9. The solving time is graphically compared in Figure 6.20. In Figure 6.21 another comparison different from the ones introduced before has been made. The size of the set $\hat{S}$ and the length of the schedule m have been represented there graphically.

From the results, with a maximum of 5 jobs at any time, the following is concluded.

(i) SDPr and LPr have consistently outperformed LPr-2 in 100% (40 out of 40) of the problems. Despite of the fact that LPr performs better than SDPr in 90% (36 out of 40) of the cases, the results obtained with both relaxations are similar.

(ii) The length of the schedule m is not directly related with solving time. As an example for this group consider problem 2 where $m = 59$ which is solved faster than problem 36 where $m = 34$.

(iii) The solving time increases as the value of $|\hat{S}|$ increases. The value for $|\hat{S}|$ varies from 12 to 20 for this particular set of problems.

Table 6.9: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 5$ jobs at the time using Relaxations 6.4.5(LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr)

No	Largest subproblem						Main problem					Gap
	$\max k_i$	m	$\lvert\hat{S}\rvert$	Time (s)			ET	Nodes	Time (s)			
				LPr	LPr-2	SDPr			LPr	LPr-2	SDPr	
1	5	48	12	13.12	35.32	12.69	7	33	13.13	35.35	12.71	-
2	5	59	13	10.52	27.01	13.82	7	32	11.00	27.36	14.22	-
3	5	49	13	8.51	47.61	11.75	7	41	12.38	51.11	15.98	-
4	5	57	12	20.19	35.86	16.66	7	63	20.70	36.25	17.16	-
5	5	40	12	22.07	31.49	20.71	7	62	22.10	31.53	20.75	-
6	5	54	13	18.85	89.91	22.20	7	53	19.48	90.47	22.70	-
7	5	49	14	22.14	71.17	24.14	7	61	22.73	71.64	24.74	-
8	5	39	14	21.23	36.72	23.54	6	76	24.33	38.88	26.27	-
9	5	47	13	18.08	68.94	18.26	5	71	27.37	104.82	29.66	-
10	5	46	14	26.45	57.86	29.28	7	80	27.05	58.33	29.82	-
11	5	28	15	19.83	67.93	25.40	4	80	24.88	74.23	30.64	-
12	5	39	15	18.59	60.64	23.76	4	66	25.33	71.15	31.22	-
13	5	53	16	24.49	105.17	32.33	7	60	24.52	105.28	32.38	-
14	5	43	16	28.65	126.11	33.91	7	90	29.27	126.86	34.40	-
15	5	48	15	25.96	184.08	35.31	7	73	26.52	184.49	35.84	-
16	5	43	15	23.94	386.79	30.18	5	89	32.23	392.55	36.76	-
17	5	42	14	25.38	205.43	34.94	6	74	27.45	206.90	36.80	-
18	5	33	17	22.53	91.32	33.61	6	76	26.46	94.41	37.32	-
19	5	30	12	37.90	59.23	37.64	7	143	38.47	59.81	38.12	-
20	5	40	15	31.04	419.09	38.32	5	97	33.90	421.26	40.88	-
21	5	43	18	31.90	432.12	39.68	5	96	36.19	435.85	43.81	-
22	5	45	16	39.19	115.30	46.83	6	110	39.89	115.99	47.64	-
23	5	43	17	34.62	568.19	47.28	7	82	35.27	568.61	47.84	-
24	5	45	15	35.99	318.09	42.40	4	118	40.92	324.86	48.20	-
25	5	49	18	33.12	319.59	48.12	7	85	33.80	320.03	48.63	-
26	5	62	18	31.85	359.25	49.24	6	84	34.43	361.00	51.27	-
27	5	74	14	45.61	391.89	53.32	7	138	46.23	392.30	53.83	-
28	5	50	16	43.12	257.30	54.81	5	125	47.30	260.72	58.46	-
29	5	31	17	37.02	301.44	55.89	5	125	40.29	304.31	58.86	-
30	5	48	20	31.57	469.75	58.08	6	77	33.72	471.09	59.69	-
31	5	49	16	49.61	442.80	60.87	7	142	49.65	442.85	60.92	-
32	5	47	19	31.95	1120.46	54.21	5	93	38.94	1129.32	60.96	-
33	5	41	19	41.45	1145.71	67.01	6	96	42.72	1146.59	67.95	-
34	5	57	19	42.68	903.67	74.37	7	102	43.29	904.11	74.90	-
35	5	58	19	35.15	498.08	65.15	5	109	44.07	540.47	76.17	-
36	5	34	18	48.41	463.07	74.99	6	146	51.19	465.55	78.05	-
37	5	44	18	54.71	711.89	75.61	6	155	57.70	714.38	78.58	-
38	5	50	15	49.97	710.95	54.79	3	181	72.91	967.50	87.46	-
39	5	46	19	40.77	1528.39	90.83	7	85	41.39	1528.78	91.33	-
40	5	42	18	43.99	486.25	64.44	4	151	68.06	666.01	98.41	-
Av	**5**	**46**	**16**	**31.05**	**343.80**	**42.41**	**6**	**93**	**34.68**	**358.58**	**46.53**	-

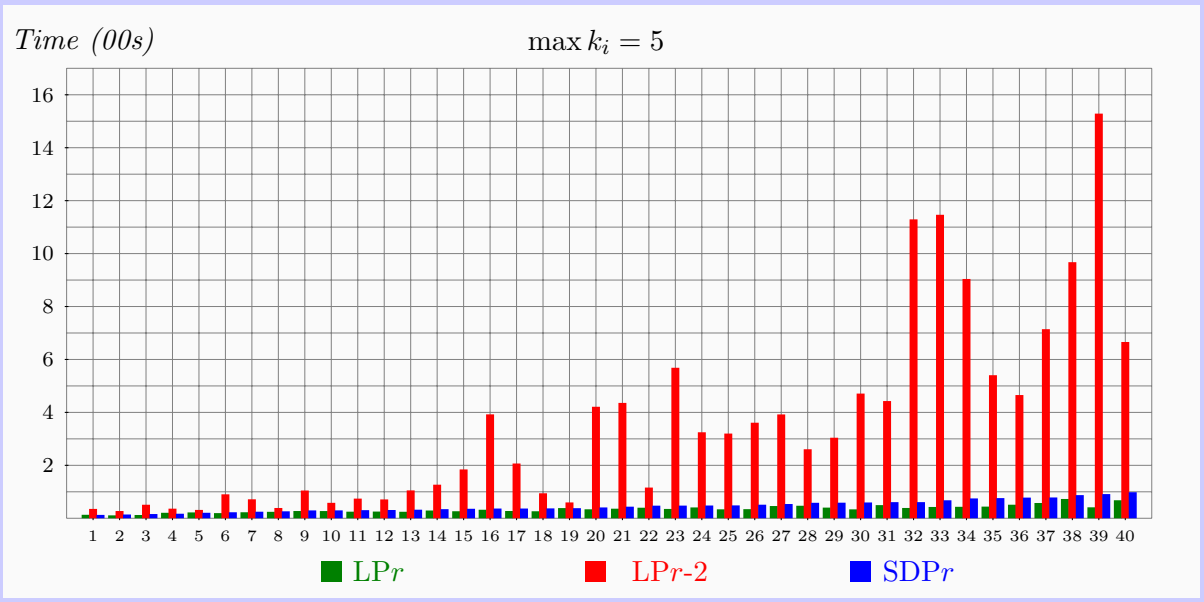


Figure 6.20: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 5$ jobs at the time using Relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr)

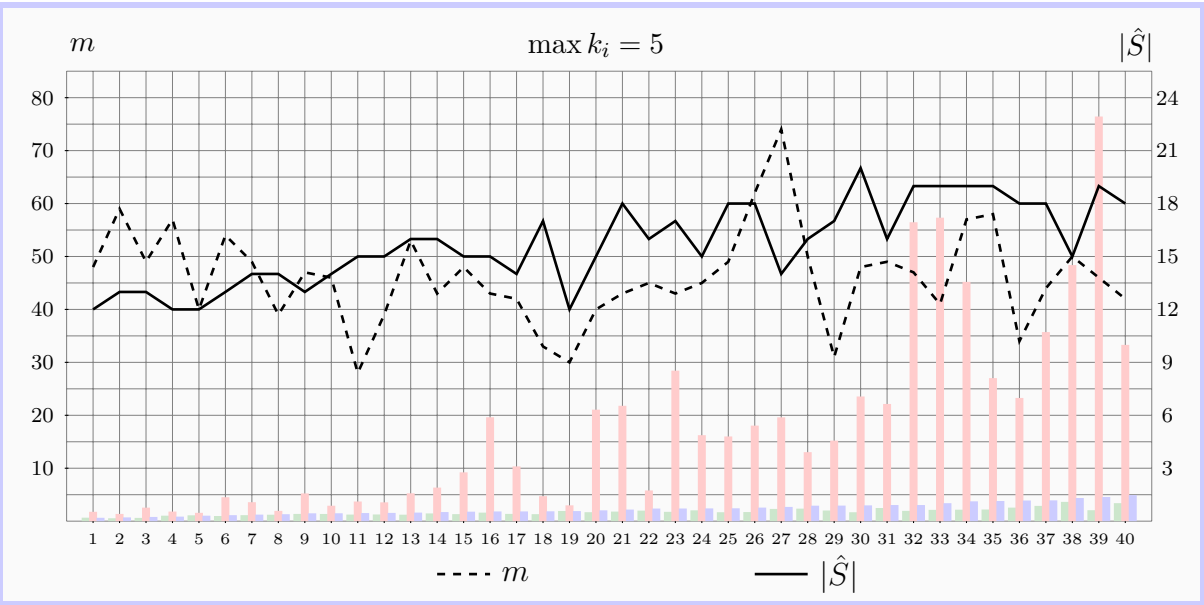


Figure 6.21: Comparison between m, $|\hat{S}|$ and solving time using Relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr) with $\max k_i = 5$. Note that the scale used for m, $|\hat{S}|$ and solving time are all different

Group 5: $n = 10$**,** $\max k_i = 6$

As in the previous case this group of problems is solved with the three relaxations generated. The results for $n = 10$ when the maximum number of available jobs is equal to 6 are presented in Table 6.10. The solving time is graphically compared in Figure 6.22. The length of the schedule m and the size of the set $\hat{S}$ are graphically shown in Figure 6.23.

The following follows from the results.

(i) LPr outperforms in time LPr-2 and SDPr in 100% (40 out of 40) of the problems. Therefore it is not practical to use LPr-2 any further. LPr and SDPr have, however, comparable alike results.

(ii) The length of the schedule m is not directly related with solving time. See for example problem 39 with $m = 48$ and problem 5 with $m = 69$. The former is solved faster with any of the relaxations than the latter.

(iii) The solving time increases as the value of $|\hat{S}|$ increases. This tendency can be observed in Figure 6.23. The values for $|\hat{S}|$ are between 15 and 28.

Table 6.10: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 6$ jobs at the time using Relaxations 6.4.5(LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr)

No	Largest subproblem						Main problem					Gap
	$\max k_i$	m	$\lvert\hat{S}\rvert$	Time (s)			ET	Nodes	Time (s)			
				LPr	LPr-2	SDPr			LPr	LPr-2	SDPr	
1	6	48	16	17.94	194.98	21.86	6	43	18.44	195.46	22.31	-
2	6	35	15	26.09	97.93	35.51	5	67	26.12	97.96	35.54	-
3	6	52	17	34.91	867.90	38.53	6	87	35.63	868.36	39.01	-
4	6	51	18	29.55	1032.03	42.60	6	68	29.58	1032.19	42.63	-
5	6	69	17	28.36	410.44	42.24	6	62	28.98	410.97	42.74	-
6	6	65	16	27.08	388.40	40.89	5	72	30.57	392.76	44.29	-
7	6	48	16	36.49	825.80	46.02	6	94	37.10	826.72	46.56	-
8	6	57	19	39.08	143.48	49.73	6	104	39.77	143.88	50.25	-
9	6	64	17	42.93	301.06	52.97	6	113	43.50	301.45	53.44	-
10	6	55	16	51.61	182.60	54.73	6	138	52.21	182.98	55.23	-
11	6	67	17	44.88	261.78	57.67	6	106	45.53	262.26	58.17	-
12	6	59	20	42.67	811.00	70.53	6	98	43.23	811.45	71.10	-
13	6	55	15	63.63	212.11	78.85	5	187	65.43	213.31	80.38	-
14	6	47	18	65.11	519.60	78.08	5	180	68.45	522.42	81.36	-
15	6	68	19	36.74	622.32	74.17	4	110	49.23	641.23	86.06	-
16	6	71	18	60.09	619.22	89.69	6	122	60.11	619.25	89.71	-
17	6	61	19	70.07	1965.99	104.87	6	164	70.64	1966.39	105.37	-
18	6	64	20	88.07	633.50	120.73	6	204	88.54	633.89	121.06	-
19	6	64	22	68.32	2598.33	124.34	5	150	71.00	2600.29	126.73	-
20	6	57	22	71.70	1703.67	129.42	6	145	71.72	1703.75	129.44	-
21	6	58	22	81.13	2568.66	136.97	6	185	81.71	2569.03	137.55	-
22	6	47	22	87.64	1718.65	148.05	5	229	90.50	1720.70	150.69	-
23	6	42	22	91.35	1243.35	141.62	4	237	102.55	1260.24	151.61	-
24	6	55	19	111.24	1435.67	156.72	4	270	113.65	1439.30	158.53	-
25	6	62	20	105.43	1634.71	164.95	6	233	105.97	1635.13	165.46	-
26	6	56	26	56.86	4900.30	166.61	5	104	57.66	4900.64	167.02	-
27	6	58	22	103.62	2649.65	179.46	6	241	104.26	2650.09	180.05	-
28	6	73	24	86.06	5974.88	180.03	6	164	86.69	5975.40	180.58	-
29	6	60	21	119.41	2988.16	182.73	6	280	119.43	2988.18	182.75	-
30	6	64	24	75.62	7443.68	202.97	6	141	75.68	7443.82	203.05	-
31	6	71	24	92.50	3041.39	198.43	4	182	97.58	3048.70	204.41	-
32	6	68	24	110.59	4525.69	215.22	6	224	111.24	4526.18	215.90	-
33	6	49	21	167.10	2794.23	216.03	6	413	167.79	2797.82	216.64	-
34	6	58	25	135.38	4757.38	255.38	5	302	138.08	4759.11	257.87	-
35	6	60	22	184.47	5339.82	294.67	6	441	185.44	5340.44	295.35	-
36	6	56	27	121.00	6198.33	304.39	4	257	133.99	6242.45	322.70	-
37	6	70	23	191.23	3800.04	335.41	6	443	191.37	3800.25	335.67	-
38	6	56	25	197.94	9786.92	323.73	6	453	197.98	9786.93	323.88	-
39	6	48	23	187.97	18783.18	353.02	6	417	190.31	18784.15	355.23	-
40	6	62	28	207.07	30660.67	553.27	5	383	214.59	30670.31	555.60	-
Av	**6**	**58**	**21**	**86.47**	**3415.94**	**151.65**	**5**	**198**	**88.56**	**3419.15**	**153.62**	-

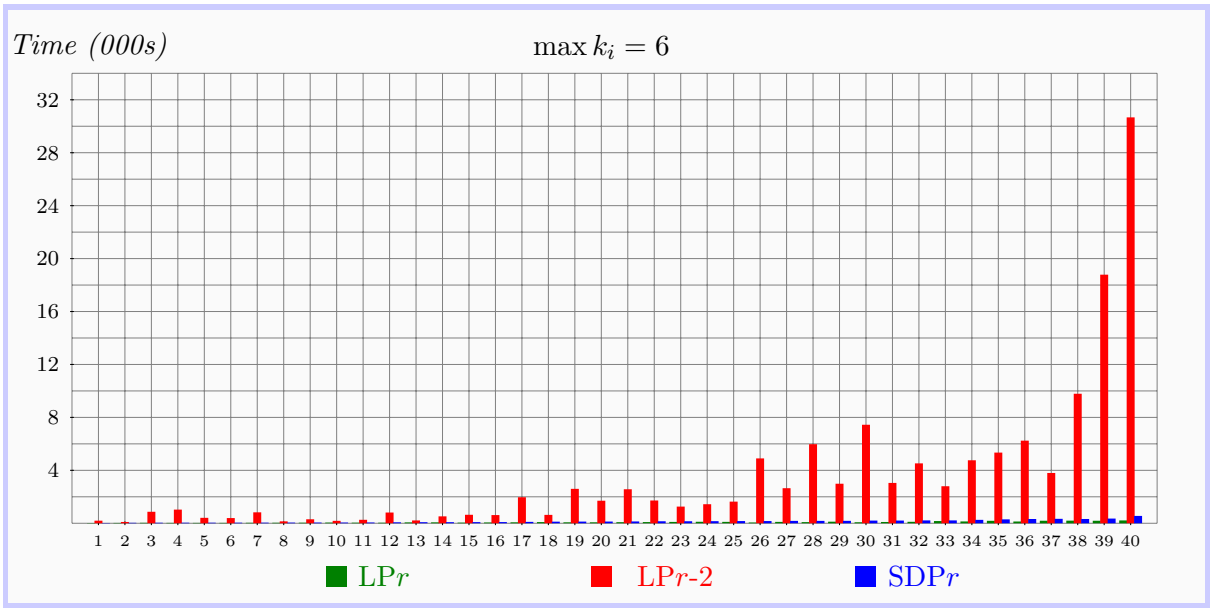


Figure 6.22: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 6$ jobs at the time using Relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr)

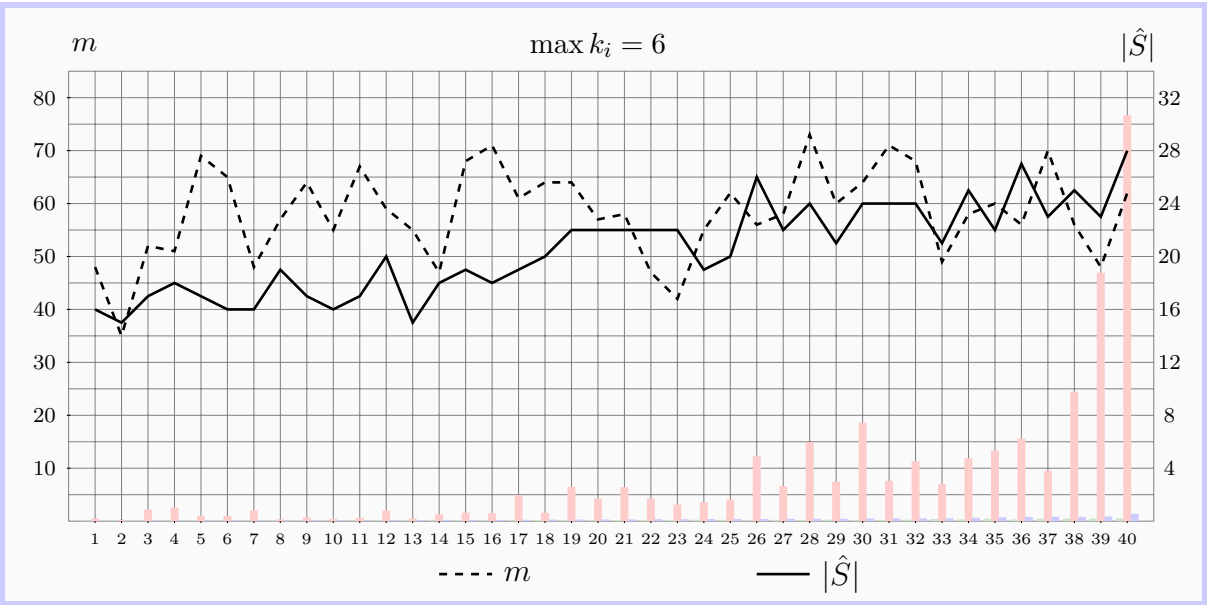


Figure 6.23: Comparison between m, $|\hat{S}|$ and solving time using Relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr) with $\max k_i = 6$. Note that the scale used for m, $|\hat{S}|$ and solving time are all different

Group 6: $n = 10$**,** $\max k_i = 7$

This group of problems with $n = 10$ when the maximum number of available jobs is equal to 7 is solved using relaxations LPr and SDPr only. The results are presented in Table 6.11. The solving time is graphically represented in Figure 6.24. The values of m and $|\hat{S}|$ are shown in Figure 6.25.

From the results the following is highlighted.

(i) Using relaxation LPr the problems with $\max k_i = 7$ are solved in average under 300 seconds. On the other hand using SDPr the average is under 700 seconds.

(ii) If $|\hat{S}| \leqslant 20$ a solution is obtained under 110 seconds using LPr and under 141 seconds using SDPr. When $|\hat{S}| \leqslant 30$ the problems are solved in under 400 seconds using LPr. Whereas using SDPr the same type of problems are solved under 1300 seconds. When $|\hat{S}| > 30$ then the solving time can vary depending on the characteristics of the problem. Observe for example that problems 34 and 39 have both the same $|\hat{S}|$ but their solving time is almost more than double using LPr and less than that amount using SDPr.

(iii) Similarly to previous cases, the length of the schedule m is not directly related with the solving time reported by the algorithm.

Table 6.11: Solving time for 40 problems with $n = 10$ and $\max k_i = 7$ jobs at the time using Relaxation 6.4.5 (LPr) and 6.4.7 (SDPr)

No	Largest subproblem					Main problem				
	$\max k_i$	m	$\lvert\hat{S}\rvert$	Time (s)		ET	Nodes	Time(s)		Gap
				LPr	SDPr			LPr	SDPr	
1	7	61	18	43.29	53.47	3	109	46.14	55.57	-
2	7	92	19	70.03	94.94	5	137	70.62	95.46	-
3	7	45	20	54.02	95.60	4	307	54.03	95.62	-
4	7	52	20	115.33	137.11	5	281	116.70	137.61	-
5	7	69	20	111.79	140.64	5	253	111.88	140.74	-
6	7	76	22	62.69	158.30	5	119	63.37	158.81	-
7	7	74	24	76.77	163.65	4	134	80.08	166.56	-
8	7	61	21	181.76	192.72	4	431	185.99	196.29	-
9	7	84	22	148.58	202.71	4	295	152.69	205.83	-
10	7	79	23	57.30	207.53	4	87	59.01	208.78	-
11	7	83	23	128.90	217.39	5	234	129.49	217.88	-
12	7	71	21	135.84	231.05	5	262	135.87	231.07	-
13	7	72	27	85.96	239.17	5	142	86.74	239.71	-
14	7	64	24	127.97	241.33	5	252	128.06	241.40	-
15	7	45	21	178.30	300.07	5	436	179.14	300.75	-
16	7	84	27	134.00	301.80	5	249	134.75	302.45	-
17	7	73	24	169.27	302.70	5	341	169.92	303.22	-
18	7	84	25	113.89	322.55	5	209	113.92	322.67	-
19	7	69	25	157.72	338.25	5	308	157.81	338.34	-
20	7	46	26	153.07	343.86	5	289	153.16	343.92	-
21	7	65	23	232.06	346.54	5	460	232.83	347.36	-
22	7	65	26	168.85	371.32	4	300	169.01	371.47	-
23	7	85	22	158.19	372.70	4	245	158.21	372.72	-
24	7	77	26	151.18	380.39	5	272	151.92	381.14	-
25	7	88	26	178.45	385.40	4	307	179.07	386.10	-
26	7	64	24	271.20	505.49	5	550	271.63	505.86	-
27	7	86	28	237.23	569.47	4	416	240.08	570.84	-
28	7	71	35	146.25	631.92	5	201	146.96	632.56	-
29	7	98	25	309.39	730.80	5	536	309.97	731.40	-
30	7	65	20	550.96	737.73	5	1309	551.01	737.83	-
31	7	61	29	390.91	852.59	5	699	390.94	852.62	-
32	7	66	31	333.05	1175.10	5	593	334.35	1176.46	-
33	7	70	30	400.85	1259.24	4	721	402.38	1260.56	-
34	7	74	34	376.93	1500.25	5	556	376.96	1500.27	-
35	7	61	32	801.40	1613.76	5	1567	805.52	1617.58	-
36	7	79	33	477.79	1643.96	5	718	477.81	1643.99	-
37	7	62	31	689.04	1978.49	5	1123	692.48	1982.07	-
38	7	63	37	760.19	2392.31	5	1324	760.21	2392.42	-
39	7	89	34	771.57	2840.75	5	1179	772.19	2841.35	-
40	7	69	35	1114.99	3305.57	5	2025	1115.01	3305.60	-
Av	**7**	**71**	**26**	**270.67**	**696.97**	**5**	**499**	**271.70**	**697.82**	-

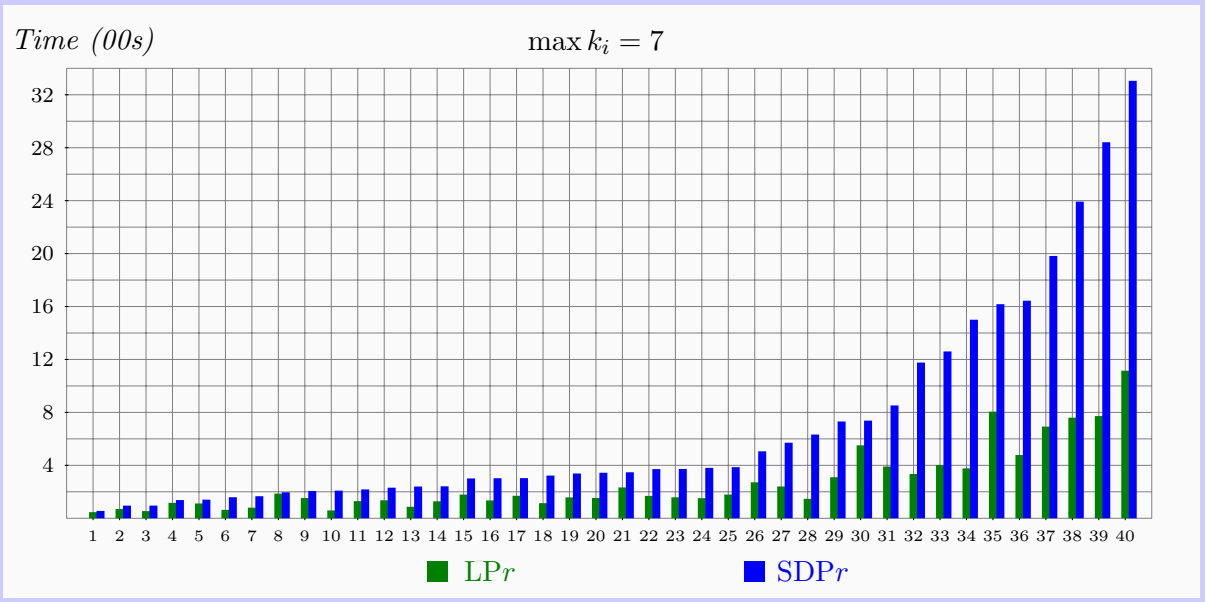


Figure 6.24: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 7$ jobs at the time using Relaxation 6.4.5 (LPr) and Relaxation 6.4.7 (SDPr)

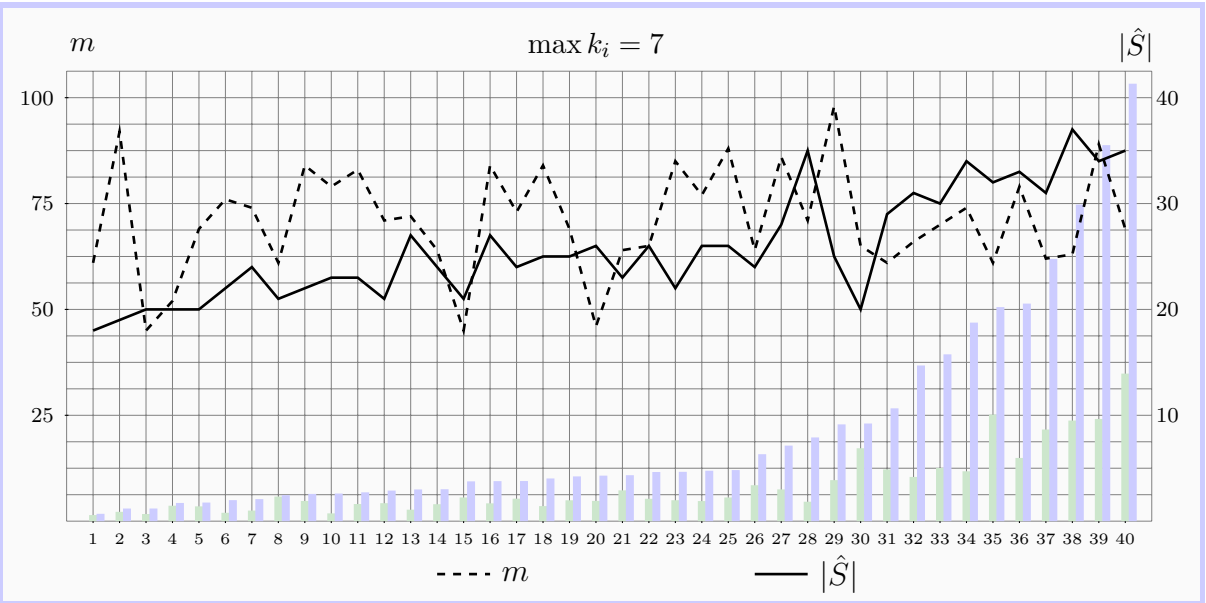


Figure 6.25: Comparison between m, $|\hat{S}|$ and solving time using Relaxation 6.4.5 (LPr) and Relaxation 6.4.7 (SDPr) with $\max k_i = 7$. Note that the scale used for m, $|\hat{S}|$ and solving time are all different

Group 7: $n = 10$**,** $\max k_i = 8$

The results for $n = 10$ when the maximum number of available jobs is equal to 8 are presented in Table 6.12. For this case relaxation SDPr was used. After solving the problems with $\max k_i = 8$ the following is concluded.

(i) When $\max k_i = 8$ and $\hat{S} \leqslant 30$ a solution for the problem is likely to be found under 1000 seconds using SDPr and this may be improved if LPr is used instead.

(ii) There is no guarantee of always finding a solution in an efficient time if $\max k_i = 8$.

(iii) The solving time increases with the size of the set $\hat{S}$. Also, it may be dramatically affected when $|\hat{S}| > 30$. See for example problem 18 with $|\hat{S}| = 31$ which is solved in 1147 seconds and in contrast problem 30 with $|\hat{S}| = 33$ takes 5024 seconds.

(iv) The average time for solving problems with $n = 10$ and $\max k_i = 8$ is 1776 seconds.

(v) Numerical experiments can be carried out to determine solving times using LPr.

Table 6.12: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 8$ jobs at the time using Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem			
	$\max k_i$	m	$\lvert\hat{S}\rvert$	SDPr (s)	ET	Nodes	SDPr (s)	Gap
1	8	84	21	204.00	4	279	204.13	-
2	8	85	24	286.87	4	285	287.08	-
3	8	69	22	364.55	6	689	372.23	-
4	8	74	23	389.90	4	531	390.64	-
5	8	55	22	401.94	3	432	402.19	-
6	8	72	27	454.79	3	333	454.81	-
7	8	54	25	487.04	4	910	487.67	-
8	8	89	27	606.65	3	714	607.44	-
9	8	73	30	627.25	4	323	627.52	-
10	8	95	26	633.25	3	376	633.46	-
11	8	67	27	670.18	4	664	671.12	-
12	8	72	26	718.04	3	426	718.84	-
13	8	99	31	733.86	4	214	734.01	-
14	8	61	26	956.01	3	604	956.03	-
15	8	81	30	1001.32	4	660	1001.34	-
16	8	80	29	1028.77	3	669	1028.80	-
17	8	102	29	1061.39	3	639	1061.42	-
18	8	98	33	1146.83	3	544	1151.64	-
19	8	71	28	1264.43	4	3965	1264.53	-
20	8	68	31	1333.14	3	897	1333.62	-
21	8	60	29	1571.15	4	1001	1573.61	-
22	8	62	35	1666.87	4	696	1668.69	-
23	8	79	31	1905.96	4	2053	1914.38	-
24	8	74	28	1733.01	4	1722	1733.04	-
25	8	89	33	2246.92	4	893	2246.94	-
26	8	95	39	2675.16	4	793	2675.19	-
27	8	79	37	2781.97	4	825	2785.53	-
28	8	76	37	4221.98	3	939	4225.34	-
29	8	66	35	4817.17	4	2413	4851.10	-
30	8	89	33	5023.78	4	2522	5024.25	-
31	8	87	43	6724.72	4	1666	6755.86	-
32	8	79	38	7099.21	3	3186	7099.24	-
Av	**8**	**78**	**30**	**1776.19**	**4**	**1027**	**1779.43**	-

Group 8: $n = 10$**,** $\max k_i = 9, 10$

The results for $n = 10$ when the maximum number of available jobs is 9 are presented in Table 6.13 and when the maximum number is 10 are in Table 6.14. This cases were both solved using relaxation SDPr only.

From the results it is easy to see the following.

(i) When $\max k_i = 9$ and $\hat{S} \leqslant 35$ the problems are likely to be solved in under an hour. However this is not guaranteed since a slight change in the size of the set $\hat{S}$ may increase the solving time drastically. The latter can be seen in the results of problem 7 in Table 6.13 which with $|\hat{S}| = 37$ was solved in 13967 seconds which is almost 4 times bigger than the one reported for problem 6 with $\hat{S} = 36$ and 3683 seconds.

(ii) When $\max k_i = 10$ if the conditions of the problem are favourable, i.e $|\hat{S}| \leqslant 32$ one could expect to solve the problem under one hour. In general problems with 10 available jobs at any time for the length of the schedule cannot be solved using the algorithm in an efficient time.

Table 6.13: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 9$ jobs at the time using Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem			
	$\max k_i$	m	$\lvert\hat{S}\rvert$	SDPr (s)	ET	Nodes	SDPr (s)	Gap
1	9	89	29	1226.67	2	1155	1226.68	-
2	9	92	34	1795.03	2	1284	1795.06	-
3	9	80	29	2301.61	2	1640	2301.62	-
4	9	84	34	3425.16	2	2072	3425.18	-
5	9	95	35	3440.17	2	1722	3440.19	-
6	9	102	36	3682.91	2	1424	3682.93	-
7	9	107	37	13966.94	2	3850	13966.96	-
Av	**9**	**93**	**33**	**4262.64**	**2**	**1878**	**4262.66**	-

Table 6.14: Solving time comparison for 40 problems with $n = 10$ and $\max k_i = 10$ jobs at the time using Relaxation 6.4.7 (SDPr)

No	Largest subproblem				Main problem			
	$\max k_i$	m	$\lvert\hat{S}\rvert$	SDPr (s)	ET	Nodes	SDPr (s)	Gap
1	10	99	32	2763.28	1	1838	2763.28	-
2	10	124	39	7208.63	2	2678	7208.67	-
3	10	124	51	59473.31	2	5411	59473.36	-
Av	**10**	**116**	**41**	**23148.41**	2	3309	**23148.44**	-

Group 9: $n = 70$**,** $\max k_i \in \{2, 3, 4, 5, 6, 7\}$

This group was created in order to determine the effectiveness of the algorithm in problems with any number of jobs n. Using the relaxations that have shown better performance in the numerical experiments with $n = 10$, namely LPr and SDPr, the algorithm was used to solve 40 instances, randomly generated, with $n = 70$ and up to 8 jobs at the time. The results are shown in Table 6.15.

From the results the following information is gathered.

(i) In average the problems with $p_j \in \{5, 10, 15\}$ $r_j \in [0, (\text{n}-6)27)$ and $w_j \in [0, 60)$ can be solved in 170 seconds using LPr and 298 using SDPr.

(ii) The branch and bound algorithm presented here can solve efficiently instances of the scheduling problem $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$ when a maximum of 7 jobs is available at a particular time and $|\hat{S}|$ is less than or equal to 35.

(iii) Numerical experiments suggest that if the conditions for the problem are favourable instances of this scheduling problem are solvable in and efficient time.

Table 6.15: Solving time comparison for 40 problems with $n = 70$ using Algorithm 6.4.11 and Relaxations 6.4.5 (LPr) and 6.4.7 (SDPr)

No	Largest subproblem					Main problem				
	$\max k_i$	m	$\lvert\hat{S}\rvert$	Time (s)		ET	Nodes	Time(s)		Gap
				LPr	SDPr			LPr	SDPr	
1	4	33	10	6.75	7.52	67	116	14.76	14.29	-
2	3	21	7	2.78	2.99	63	133	21.25	20.67	-
3	4	50	10	7.17	6.98	62	149	26.14	22.78	-
4	4	28	10	9.95	10.19	61	158	30.55	25.78	-
5	4	25	9	9.96	8.94	56	212	47.05	39.02	-
6	5	41	16	15.34	22.54	62	169	38.39	42.57	-
7	5	57	15	28.74	30.72	56	258	71.46	75.27	-
8	6	66	19	40.10	61.23	62	207	58.30	78.16	-
9	5	50	12	24.20	32.06	58	253	64.93	78.72	-
10	4	43	13	13.86	14.65	49	302	83.44	81.63	-
11	5	40	14	34.22	35.85	56	299	82.35	89.40	-
12	5	49	20	43.01	71.41	62	230	63.43	90.33	-
13	6	55	15	25.00	35.44	56	287	77.44	93.60	-
14	6	60	15	55.35	72.68	59	306	88.11	104.15	-
15	5	48	20	44.00	70.77	60	261	76.12	105.48	-
16	5	38	16	31.18	40.07	54	316	107.81	124.30	-
17	6	71	22	52.70	112.43	59	237	79.67	137.48	-
18	6	64	17	60.17	61.64	54	388	131.41	144.73	-
19	6	54	22	37.07	75.67	55	287	98.39	153.14	-
20	5	41	18	56.72	79.89	55	388	130.88	168.62	-
21	5	61	18	58.31	90.83	54	433	137.62	175.93	-
22	6	55	20	115.15	195.14	56	458	150.96	230.11	-
23	6	63	19	67.06	99.02	49	488	174.13	238.93	-
24	6	47	21	78.17	172.77	56	347	135.27	240.58	-
25	7	54	19	125.34	168.04	52	485	181.58	242.08	-
26	7	70	22	114.86	206.95	53	424	161.33	252.17	-
27	6	48	28	74.72	218.40	56	327	118.23	265.77	-
28	6	62	23	90.87	160.87	55	464	173.81	269.44	-
29	7	78	24	97.27	224.92	59	334	149.82	295.23	-
30	6	81	24	180.68	288.69	58	545	220.96	329.77	-
31	6	60	22	178.21	344.80	60	491	198.03	364.10	-
32	7	89	24	150.46	216.96	50	464	328.35	419.39	-
33	7	64	20	223.70	317.62	58	739	293.48	437.87	-
34	7	71	25	116.01	346.79	52	491	210.65	473.50	-
35	6	63	26	167.75	481.85	56	481	204.33	532.07	-
36	6	61	30	149.13	461.07	51	530	252.42	588.89	-
37	7	69	28	104.43	289.33	51	796	361.81	694.48	-
38	8	76	32	424.37	914.49	56	935	485.36	989.73	-
39	7	65	27	649.61	1035.40	49	1796	824.81	1410.20	-
40	7	60	32	556.04	1668.97	55	1429	647.77	1803.16	-
Av	**6**	**56**	**20**	**108.01**	**218.91**	**56**	**435**	**170.07**	**298.45**	-

Table 6.16: Average results for 282 instances of the scheduling problem with $n = 10$, $p_j \in \{5, 10, 15\}$, $r_j \in [0, (n-6)20)$, $w_j \in [0, 60)$ using Algorithm 6.4.8 and Relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr)

T	Largest subproblem						Main problem				Gap
	$\max_{k_i}$	m	$\lvert\hat{S}\rvert$	Time (s)			ET	Nodes	Time (s)		
				LPr	LPr-2	SDPr			LPr	LPr	SDPr	
40	2	15	3	-	0.43	0.49	10	15	-	1.09	1.20	-
40	3	23	7	-	2.35	2.65	8	24	-	3.86	4.38	-
40	4	37	11	-	23.25	10.77	7	43	-	25.54	12.95	-
40	5	46	16	31.05	343.80	42.41	6	93	34.68	358.58	46.53	-
40	6	58	21	86.47	3415.94	151.65	5	198	88.56	3419.15	153.62	-
40	7	52	26	270.67	-	696.97	5	499	271.70	-	697.82	-
32	8	78	30	-	-	1776.19	4	1027	-	-	1779.43	-
7	9	93	33	-	-	4262.64	2	1878	-	-	4262.66	-
3	10	116	41	-	-	23148.41	2	3309	-	-	23148.44	-

Summary of results

In Table 6.16 the average results when using the branch and bound algorithm presented in 6.4.8 with relaxations 6.4.5 (LPr), 6.4.6 (LPr-2) and 6.4.7 (SDPr), and the different groups of problems solved when $n = 10$, are summarised.

The relation between the maximum number of available jobs at any time, $\max k_i$, the size of the set $\hat{S}$ and the solving time using Algorithm 6.4.8 is categorised in Table 6.17. The results presented in the latter table consider the ones reported for the largest subproblem for each of the cases and no the total for the main problem. As determined by the numerical experiments these results may be improved using different relaxations and appropriate solvers for the type of relaxation.

Table 6.17: Relation between $\max k_i$, $|\hat{S}|$ and solving time using Algorithm 6.4.8

$\max k_i$	$\lvert\hat{S}\rvert$	Solving time (seconds)
2	up to 3^*	less than 1
3	up to 8^*	less than 2
4	up to 10	less than 10
	up to 14	less than 20
5	up to 13	less than 20
	up to 15	less than 50
	up to 20	less than 100
6	up to 15	less than 80
	up to 20	less than 165
	up to 28	less than 555
7	up to 19	less than 100
	up to 30	less than 1259
	up to 35	less than 3600
8	up to 25	less than 500
	up to 30	less than 1100
	up to 38	less than 7100
9,10	up to 30	less than 3600
	more than 30	depends on the problem

*Maximum possible size

In summary, the main characteristics of the branch and bound algorithm presented in 6.4.8 for the scheduling problem $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ are:

(i) Overall performance: The algorithm solves efficiently problems with any number of jobs n provided the maximum number of available jobs at any time is 7 and $|\hat{S}| \leqslant 30$.

(ii) Customisation: The branch and bound algorithm provided here analyses the main characteristics of the problem and splits the problem into appropriate subproblems. For each subproblem a new enumeration tree is developed and the algorithm identifies those nodes and branches that are worth exploring. This is determined by

selecting those times in the considered interval at which a new job has been release into the system or when the machine is free to process another job.

(iii) Calculating lower bounds: The main advantage of the branch and bound algorithm is its customised designed. Therefore different relaxations can be used to calculate the lower bounds. In this study three relaxations where provided. Relaxation 6.4.5 (LPr) using SeDuMi (Sturm (1999)) works better than Relaxation 6.4.7 (SDPr) using PENBMI (Kočvara and Stingl (2003)), however both solve any instance of the problem in an efficient time with a maximum of 7 jobs at any time and the size of the set $\hat{S}$ is less than or equal to 30. Relaxation 6.4.6 (LPr-2) with PENBMI (Kočvara and Stingl (2003)) shows good performance with a maximum of 3 jobs at any time, but for any instance with $\max k_i > 3$ other relaxations should be used.

(iv) Number of nodes: The maximum number of explored nodes per tree is given by (see Lemma 6.4.9)

$$\sum_{i \in |\hat{S}|} \left(h_i k_i^{k_i - 1} + (1 - h_i) \right)$$

and it is independent of the relaxation to calculate the lower bounds used.

(v) Number of variables: Depending on the relaxation used the number of variables may be increased or decreased. These fluctuations are in turn reflected in overall performance.

(vi) Number of constraints: The number of constraints differs in accordance with the relaxation used. Depending on the sizes of the matrix constraints the performance can be severely affected.

(vii) Number of iterations: The number of iterations is determined by the algorithm and it is independent of the relaxation used.

(viii) Length of the schedule: There is no correlation between the length of the schedule m and the solving time.

(ix) $|\hat{S}|$ and solving time: There is a correlation between the size of $\hat{S}$ and the solving time. Such correlation is given in Table 6.17.

(x) Time-indexed variables: The variables for the problem are time-indexed but with an important characteristic. Instead of being defined for every consecutive time i they are defined only for those times that are worth exploring (see Theorem 6.4.3). This characteristic makes it faster when compared with any other algorithm indexing the variables to all the consecutive values in a time interval.

6.6 Conclusion and future work

Hitherto the best approximation algorithms developed for the scheduling problem represented as $1\,|\,r_j, pmtn\,|\,\sum w_jC_j$ have been provided by Schulz and Skutella (2002) and Afrati et al. (1999). In the former case a solution obtained with their approximation algorithm is guaranteed to be less than $\frac{4}{3}$ (1.33) times the optimal solution. Afrati et al. (1999) provided another approximation algorithm that finds a solution in $\mathcal{O}(2^{poly(1/\epsilon)}n + n\log n)$ time.

The main contribution of this Chapter is the development of a customised branch and bound algorithm. Using a linear programming and a semidefinite programming based approach, three relaxations to calculate the lower bounds of the algorithm are developed. The first relaxation consist in a traditional linear programming relaxation presented in Problem 6.4.5 (LPr). A second relaxation which was shown to be equivalent to a linear programming relaxation was given in Problem 6.4.6 (LPr-2). A third relaxation, based on semidefinite programming presented in Problem 6.4.7 (SDPr) was also provided.

Relaxation LPr and SDPr outperform the other LPr-2 when used to calculate the lower bounds for the branch and bound algorithm. LPr-2 contains matrix variables $\mathsf{Y}_{(j)}$ which are significantly bigger when compared with variables $\mathsf{Y}_{(ij)}$ in SDPr which are only 2×2 matrices. Due to this characteristic the results reported with LPr-2 are worse than the other two. LPr-2 works well with up to 3 maximum number of available jobs at any time. This result may be improved if a specialised solver for linear programming, IBM ILOG CPLEX (2009) for instance, is used. However it is important to notice that the number of variables using this relaxations grows with an increase in the maximum number of available jobs at any time.

Another aspect of the customised branch and bound algorithm presented here, is that it partitions the problem appropriately and it creates new enumeration trees only when required. Additionally, each enumeration tree grows only in those branches that are worth exploring. This characteristic allows the algorithm to obtain solutions in an efficient time for problems with any number of jobs provided the maximum number at any time is 7.

The maximum number of nodes explored per enumeration tree using the algorithm is (recall Lemma 6.4.9)

$$\sum_{i \in |\hat{S}|} \big(h_i k_i^{k_i - 1} + (1 - h_i)\big), \tag{6.16}$$

where k_i denotes the number of available jobs at the time i and h_i indicates when a job has been release at the time i, when the machine is free to process another job or when there is more than one job available. Observe that equation (6.16) is independent of the the total number of jobs n. Thus, the complexity depends on the partitions made for the whole problem.

Numerical experiments indicated that the branch and bound customised algorithm finds a solution in an efficient time suggesting it could be for those instances polynomial for any size of the problem provided that the maximum number of jobs at a particular

time, k_i, is less than or equal to 7 and also $|\hat{S}| \leqslant 30$.

Previous results have provided solutions whose complexity increases directly with the size n of the problem. The results presented here improve those previously reported in the literature in that the complexity is independent of n.

Further experiments can be carried out using LPr for cases with $\max k_i \geqslant 8$. We have shown the advantages of using customised branch and bound algorithms with sophisticated relaxations. Similar relaxations can be developed to formulate and explore the solutions of other scheduling problems.

CHAPTER 7

CONCLUSIONS

In this project two scheduling problems have been addressed. Using convex conic relaxations important results for both cases were obtained.

7.1 $1 \,|\, p_j = p, r_j \,|\, \sum w_j T_j$

The complexity of this problem was still unknown. At first we modelled the problem as an integer program and we study its linear program relaxation. We showed that the constraint matrix is totally unimodular and as a result the linear program relaxation have an integer optimal solution thus solving the integer program to optimality. We conclude that the problem can be solved to optimality in polynomial time. We also studied the case when the schedule has a minimum completion time, or makespan. Following a similar methodology of the above we showed that the constraint matrix of the linear program relaxation for this case is also totally unimodular and as such it can also be solved to optimality in polynomial time.

7.2 $1 \mid r_j, pmtn \mid \sum w_j C_j$

For this $\mathcal{NP}$-hard problem an efficient customised branch and bound algorithm is proposed. We developed three relaxations based on linear and semidefinite programming. Since we are using an enumerative algorithm the solution is guarantee to be optimal.

We have proven the advantage of using the proposed customised branch algorithm which by taking advantage of the characteristics of this scheduling problem partitions the problem creating enumeration trees only when required and also identifies those branches and nodes of each tree that are worth exploring. The latter is decided by determining those times at which a job arrives or when the machine is free to process another job. Numerical experiments have shown that the branch and bound algorithm presented here can be used to solve efficiently instances of any size of the problem provided that two conditions are satisfied, no more than 7 jobs are available at a particular time and the set for the time-indexed variables is less than 30.

PART III

APPENDIX

APPENDIX A

MATRIX ANALYSIS

A.1 Eigenvalues and eigenvectors

An eigenvector of a $n \times n$ matrix A is a nonzero vector $\boldsymbol{x}$ which satisfies

$$\mathsf{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \qquad \boldsymbol{x} \neq 0 \tag{A.1}$$

where λ is a scalar and an eigenvalue of A. The relation

$$(\lambda \mathsf{I} - \mathsf{A})\, \boldsymbol{x} = 0, \qquad \boldsymbol{x} \neq 0$$

may be used to find the values of λ and $\boldsymbol{x}$, where I denotes the identity matrix. Since $\boldsymbol{x} \neq 0$ then in order to satisfy (A.1) the matrix $(\lambda \mathsf{I} - \mathsf{A})$ has to be singular, i.e.

$$\det (\lambda \mathsf{I} - \mathsf{A}) = 0. \tag{A.2}$$

The solution of (A.2) gives a polynomial in λ which is known as the characteristic polynomial. Consequently, the roots of the polynomial (A.2) are the eigenvalues of A, denoted as $\lambda_1, \ldots, \lambda_n$.

Definition A.1.1. (Elementary symmetric polynomials) *Given $\lambda_1, \ldots, \lambda_n$, the elementary symmetric polynomials are defined as*

$$e_k(\lambda_1, \ldots, \lambda_n) = \sum_{1 \leqslant i_1 < \ldots < i_k \leqslant n} \prod \lambda_{i_j}, \qquad k = 0, 1, \ldots, n \tag{A.3}$$

or

$$\begin{aligned}
e_0(\lambda_1, \ldots, \lambda_n) &= 1 \\
e_1(\lambda_1, \ldots, \lambda_n) &= \lambda_1 + \ldots + \lambda_n \\
e_2(\lambda_1, \ldots, \lambda_n) &= \lambda_1\lambda_2 + \lambda_1\lambda_3 + \ldots + \lambda_{n-2}\lambda_n + \lambda_{n-1}\lambda_n \\
e_3(\lambda_1, \ldots, \lambda_n) &= \lambda_1\lambda_2\lambda_3 + \lambda_1\lambda_2\lambda_4 + \ldots + \lambda_{n-3}\lambda_{n-1}\lambda_n + \lambda_{n-2}\lambda_{n-1}\lambda_n \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
e_n(\lambda_1, \ldots, \lambda_n) &= \lambda_1 \ldots \lambda_n.
\end{aligned}$$

Definition A.1.2. (Principal submatrix) *Let $\mathsf{A} = (a_{ij})$ be an $n \times n$ matrix. Then a principal submatrix of A is any $k \times k$ matrix, $1 \leqslant k \leqslant n$, obtained by removing $n-k$ rows and columns of A, such that if the ith row is removed then the ith column is removed too.*

Definition A.1.3. (Principal minor) *A principal minor is the determinant of a given principal submatrix.*

Definition A.1.4. (Sum of principal minors) *Let* $E_k(\mathsf{A})$ *denote the sum of the* $k \times k$ *principal minors of* A*, i.e.*

$$E_k(\mathsf{A}) = \sum_{i_1 < \ldots < i_k} \Delta(i_1, \ldots, i_k) \qquad k = 1, \ldots, n \tag{A.4}$$

where $\Delta(i_1, \ldots, i_k)$ *is the determinant of the matrix composed by the intersection of rows* $i_1, \ldots, i_k$ *and columns* $i_1, \ldots, i_k$*.*

Observe that in (A.4) if $k = 1$ then we have n, 1×1 principal submatrices corresponding to each element in the diagonal of A, whose determinant is the element itself, and thus $E_1(\mathsf{A}) = \sum_{i=1}^{n} a_{ii}$.

Definition A.1.5. (Trace of a matrix) *The trace of a matrix* A*, denoted by* $\mathrm{tr}(\mathsf{A})$*, is given by*

$$\mathrm{tr}(\mathsf{A}) = a_{11} + a_{22} + \ldots + a_{nn}. \tag{A.5}$$

Remark $E_1(\mathsf{A}) = \mathrm{tr}(\mathsf{A})$.

In the same way, if $k = n$ then we have 1, $n \times n$ matrix which is A and thus $E_n(\mathsf{A}) = \det(\mathsf{A})$.

Lemma A.1.6. *Let* $\mathsf{A} = (a_{ij})$ *be an* $n \times n$ *matrix. Let the entries* a_{ij} *be either constant or linear in* λ*. Then, for all* $m \leqslant n$*, if the matrix has* m *entries with nonzero* λ *coefficients and no two of these entries lie in the same row or same column. Then the characteristic polynomial* $\det(\lambda\mathsf{I} - \mathsf{A})$ *has degree* m*, i.e.*

$$\det(\lambda\mathsf{I} - \mathsf{A}) = \lambda^m + c_1\lambda^{m-1} + c_2\lambda^{m-2} + \ldots + c_n. \tag{A.6}$$

Proof. We will show this by induction on m. As an induction hypothesis suppose we have a $n \times n$ matrix in which exactly m entries have nonzero λ coefficients and no two of these entries lie in the same row or same column. We say that such entries are *independent*. Then we claim that the determinant of such matrix is a polynomial in λ of degree m. If $m = 0$ then the result is trivial. Let us assume that the hypothesis holds for some $k = m-1$ and so we need to check that it works for $k = m$. Assuming that $1 \leqslant m \leqslant n-1$ and using Laplace expansion[1] with a row i containing a λ then every submatrix resulting by deleting the row i and a column j of the original matrix has at most $l \leqslant m-1$ entries with λ coefficients. Thus the determinant of every resulting submatrix in the expansion of row i is a polynomial of degree less than or equal to $m-1$. However if we consider an entry with a λ, since such entries are independent, the submatrix has $l = m-1$, λ entries, and so the determinant of such submatrix is a polynomial of degree $m-1$. In this case when multiplied by the entry in the ith row that includes a λ the polynomial will have a degree equal to m, i.e. Using Laplace expansion by the ith row containing one

[1] Given an $n \times n$ matrix A, the determinant can be calculated expanding by cofactors on row i or column j, i.e.

$$\begin{aligned}\det \mathsf{A} &= a_{i\,1} \cdot \mathsf{B}_{i\,1} + a_{i\,2} \cdot \mathsf{B}_{i\,2} + \cdots + a_{i\,n} \cdot \mathsf{B}_{i\,n} \\ &= a_{1j} \cdot \mathsf{B}_{1j} + a_{2j} \cdot \mathsf{B}_{2j} + \cdots + a_{nj} \cdot \mathsf{B}_{nj}.\end{aligned}$$

With $\mathsf{B}_{ij} = (-1)^{i+j}|\mathsf{M}_{ij}|$ and M_{ij} is the matrix obtained by deleting the ith row and jth column of A.

independent λ we get

$$\begin{aligned}\det(\lambda \mathsf{I} - \mathsf{A}) &= p_1^{(m)}(\lambda) + p_2^{(l_2)}(\lambda) + \ldots + p_n^{(l_n)}(\lambda) \\ &= p^{(m)}(\lambda)\end{aligned}$$

where $l \leqslant m-1$ and $p^{(s)}(\lambda)$ indicates a polynomial in λ of degree s. Therefore given a matrix with m independent λ coefficients then the determinant of such matrix is polynomial in λ of degree m.

□

Lemma A.1.7. *Let* A *be an* $n \times n$ *matrix. Then the characteristic polynomial* $\det(\lambda \mathsf{I} - \mathsf{A})$ *has degree* n*, i.e.*

$$\det(\lambda \mathsf{I} - \mathsf{A}) = \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \ldots + c_n. \tag{A.7}$$

Proof. The $\det(\lambda \mathsf{I} - \mathsf{A})$ has the form

$$\begin{pmatrix} -a_{11} + \lambda & -a_{12} & \ldots & -a_{1n} \\ -a_{21} & -a_{22} + \lambda & \ldots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \ldots & -a_{nn} + \lambda \end{pmatrix}.$$

Observe that the matrix has n independent λ entries. By Lemma A.1.6 with $m = n$ the characteristic polynomial in λ has degree n as claimed.

□

Now we are able to introduce a relation between the elementary symmetric polynomials $e_k(\lambda_1, \ldots, \lambda_n)$ and the sum of principal minors $E_k(\mathsf{A})$.

Theorem A.1.8. *Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of an $n \times n$ matrix A with multiplicity. Then*

$$e_k(\lambda_1, \ldots, \lambda_n) = E_k(\mathsf{A}), \qquad k = 1, \ldots, n. \tag{A.8}$$

Proof. Considering the roots of the polynomial in (A.7) we have the expression

$$(\lambda - \lambda_1)(\lambda - \lambda_2) \ldots (\lambda - \lambda_n). \tag{A.9}$$

Specifically, expanding (A.9) we get

$$\begin{aligned}(\lambda - \lambda_1)(\lambda - \lambda_2) \ldots (\lambda - \lambda_n) = \lambda^n & \\ - (\lambda_1 + \ldots + \lambda_n)\lambda^{n-1} & \\ + (\lambda_1\lambda_2 + \lambda_1\lambda_3 + \ldots + \lambda_{n-2}\lambda_n + \lambda_{n-1}\lambda_n)\lambda^{n-2} & \\ - \ldots & \\ + (-1)^n(\lambda_1 \ldots \lambda_n). &\end{aligned} \tag{A.10}$$

Observe that in (A.10) every coefficient of λ^{n-k} corresponds to $e_k(\lambda_1, \ldots, \lambda_n)$ for $k =$

$1, \ldots, n$ defined in (A.3). Therefore (A.10) can be written as

$$\begin{aligned}(\lambda - \lambda_1)(\lambda - \lambda_2)\ldots(\lambda - \lambda_n) = \lambda^n & \\ - e_1(\lambda_1, \ldots, \lambda_n)\lambda^{n-1} & \\ + e_2(\lambda_1, \ldots, \lambda_n)\lambda^{n-2} & \\ - \cdots & \\ + (-1)^n e_n(\lambda_1, \ldots, \lambda_n). &\end{aligned} \tag{A.11}$$

On the other hand, by Lemma A.1.7

$$\det(\lambda \mathsf{I} - \mathsf{A}) = \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \ldots + c_n.$$

We have seen that

$$\det(\lambda \mathsf{I} - \mathsf{A}) = \begin{pmatrix} -a_{11} + \lambda & -a_{12} & \ldots & -a_{1n} \\ -a_{21} & -a_{22} + \lambda & \ldots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \ldots & -a_{nn} + \lambda \end{pmatrix}.$$

Without loss of generality let us identify the λ in the ith row with λ_i, for $i = 1, \ldots, n$, such that

$$\begin{pmatrix} -a_{11} + \lambda_1 & -a_{12} & \ldots & -a_{1n} \\ -a_{21} & -a_{22} + \lambda_2 & \ldots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \ldots & -a_{nn} + \lambda_n \end{pmatrix}.$$

We claim that the term in $\lambda_{i_1} \dots \lambda_{i_k}$ with $k \leqslant n$ is equal to

$$\det\{\dots\{\{\{-\mathsf{A}\}_{i_1}\}_{i_2}\}\dots\}_{i_k} \tag{A.12}$$

where $\{\mathsf{B}\}_l$ indicates the resultant matrix after deleting the row l and the column l from B. Let (A.12) be our induction hypothesis on k. For $k = 0$ the hypothesis holds. Assume that the hypothesis is true for $k-1$ then we need to show that it also holds for k. Expanding around row i_k the term in $\lambda_{i_1} \dots \lambda_{i_k}$ in $\lambda\mathsf{I}-\mathsf{A}$ is equal to the term in $\lambda_{i_1} \dots \lambda_{i_{k-1}}$ in $\{\lambda\mathsf{I} - \mathsf{A}\}_{i_k}$. By the induction hypothesis this is equal to $\det\{\dots\{\{\{-\mathsf{A}\}_{i_k}\}_{i_1}\}\dots\}_{i_{k-1}}$ which is $\det\{\dots\{\{\{-\mathsf{A}\}_{i_1}\}_{i_2}\}\dots\}_{i_k}$. Then (A.12) holds. Let $d(i_1, \dots, i_k)$ be the term with $\lambda_{i_1} \dots \lambda_{i_k}$. Now to know the coefficient of a term with λ^k we need to sum up all the terms $\lambda_{i_1} \dots \lambda_{i_k}$. Thus, using (A.12) we have

$$\sum_{i_1<\dots<i_k} d(i_1, \dots, i_k) = \sum_{i_1,\dots,i_k} \det\{\dots\{\{\{-\mathsf{A}\}_{i_1}\}_{i_2}\}\dots\}_{i_k}.$$

Let $d(i_1 < \dots < i_k) = d(S)$ where $S = \{i_1, \dots, i_k\}$, and similarly let $\Delta(i_1, \dots, i_k) = \Delta(S)$ Then $d(S) = \Delta([n]\backslash S)$. Let us define d_k as the term of λ^k given by

$$d_k = \sum_{i_1<\dots<i_k} d(i_1, \dots, i_k)$$

then

$$d_k = \sum_{j_1<\dots<j_{n-k}} \Delta(j_1, \dots, j_{n-k}). \tag{A.13}$$

Thus since the term in λ^k is given by equation (A.13) therefore the term in λ^{n-k} is given by

$$\sum_{i_1<\dots<i_k} \Delta(i_1, \dots, i_k),$$

and so

$$\begin{aligned}\det(\lambda \mathsf{I} - \mathsf{A}) = \lambda^n & \\ & - \left(\sum_{j=1}^{n} \Delta(i_j) \right) \lambda^{n-1} \\ & + \left(\sum_{\substack{i_j < i_k \\ j,k=1}}^{n} \Delta(i_j, i_k) \right) \lambda^{n-2} \\ & - \left(\sum_{\substack{i_j < i_k < i_l \\ j,k,l=1}}^{n} \Delta(i_j, i_k, i_l) \right) \lambda^{n-3} \\ & + \cdots \\ & + (-1)^n \det(\mathsf{A}).\end{aligned} \tag{A.14}$$

Which is equivalent to

$$\begin{aligned}\det(\lambda \mathsf{I} - \mathsf{A}) = \lambda^n & \\ & - E_1(\mathsf{A}) \lambda^{n-1} \\ & + E_2(\mathsf{A}) \lambda^{n-2} \\ & - \cdots \\ & + (-1)^n E_n(\mathsf{A}).\end{aligned} \tag{A.15}$$

Consequently from (A.11) and (A.15) it follows that

$$e_k(\lambda_1, \ldots, \lambda_n) = E_k(\mathsf{A}), \qquad k = 1, \ldots, n.$$

□

In particular the next two corollaries follow from Theorem A.1.8.

Corollary A.1.9. *Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of an $n \times n$ matrix A with multiplicity. Then*

$$\operatorname{tr}(\mathsf{A}) = \sum_{i=1}^{n} \lambda_i. \tag{A.16}$$

Proof. Using Theorem A.1.8 we know that $e_k(\lambda_1, \ldots, \lambda_n) = E_k(\mathsf{A})$ for $k = 1, \ldots, n$. This corollary corresponds to the particular case when $k = 1$.

□

Corollary A.1.10. *Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of an $n \times n$ matrix A with multiplicity. Then*

$$\det(\mathsf{A}) = \prod_{i=1}^{n} \lambda_i. \tag{A.17}$$

Proof. From Theorem A.1.8 $e_k(\lambda_1, \ldots, \lambda_n) = E_k(\mathsf{A})$ for $k = 1, \ldots, n$. This corollary corresponds to the case when $k = n$.

□

A.2 Rank-nullity theorem

Definition A.2.1. (Basis) *Let $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ be n linearly independent vectors. Then a basis will be formed by those n vectors if by taking linear combinations of them it is possible to get every vector in the given vector space.*

Definition A.2.2. (Range and null space of vector spaces) *Let* A *be an* $m \times n$ *matrix representing a linear transformation from* $\mathbb{F}^n$ *to* $\mathbb{F}^m$*. Then*

(i) The range *of* A *is* $\left\{\boldsymbol{y} \in \mathbb{F}^m \mid \mathsf{A}\boldsymbol{x} = \boldsymbol{y}, \text{ for some } \boldsymbol{x} \in \mathbb{F}^n\right\}$*. The range is a subspace of* $\mathbb{F}^m$ *(for a proof see Andrilli and Hecker (2003) page 259). The dimension* (number of linearly independent vectors needed to generate the subspace) *of the range is known as the* rank *of the matrix.*

(ii) The null space *of* A *is* $\left\{\boldsymbol{x} \in \mathbb{F}^n \mid \mathsf{A}\boldsymbol{x} = \mathsf{0}\right\}$*. The null space is a subspace of* $\mathbb{F}^n$ *(for a proof see Andrilli and Hecker (2003) page 259). The dimension of the null space is known as* nullity *of the matrix.*

Theorem A.2.3. (Rank-nullity) *Let* A *be an* $m \times n$ *matrix. Then the rank and the nullity of the matrix add up to the number of columns of the matrix, i.e.*

$$\text{rank}(\mathsf{A}) + \text{nullity}(\mathsf{A}) = n.$$

Proof. Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ be the column vectors of the matrix A. Such columns vectors may form a basis for the range if they are linearly independent, which may or may be not the case. Let $k \leqslant n$ be the maximum number of linearly independent columns of A. We claim that those k vectors, without loss of generality denoted by $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$, form a basis for the range of A and $\text{rank}(\mathsf{A}) = k$. To show that $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ is a basis for the range we need to proof that every element in the range may be represented as a linear combination of those k linearly independent vectors, i.e. those k vectors span the range. The vector space $\mathbb{R}^n$ is equipped with the *standard basis* $\{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_n\}$, in which,

for each $i = 1, \ldots, n$, e_i is a vector with 1 in the ith component and zeros in all the other entries, then $span\{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_n\} = \mathbb{R}^n$ and

$$\mathsf{A}\mathrm{e}_i = \boldsymbol{v}_i \qquad i = 1, \ldots, n$$

and since matrix multiplications preserve linear combinations then

$$\begin{aligned} \text{range}(\mathsf{A}) &= span\{\mathsf{A}\mathrm{e}_1, \mathsf{A}\mathrm{e}_2, \ldots, \mathsf{A}\mathrm{e}_n\} \\ &= span\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}. \end{aligned} \tag{A.18}$$

Therefore the range of A is the set of linear combinations of the columns in A, Observe that $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ forms a maximal set of linearly independent vectors of A then $\{\boldsymbol{v}_{k+1}, \boldsymbol{v}_{k+2}, \ldots, \boldsymbol{v}_n\}$ depend on $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ and so for any column vector $k < j \leqslant n$ there exist coefficients α_{ji} such that

$$\boldsymbol{v}_j = \sum_{i=1}^{k} \alpha_{ji} \boldsymbol{v}_i \qquad k < j \leqslant n.$$

Then $\boldsymbol{v}_j \in span\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ for all $j > k$, and so

$$span\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\} = span\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}. \tag{A.19}$$

Therefore from equations (A.18) and (A.19)

$$\text{range}(\mathsf{A}) = span\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}.$$

Thus $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ forms a basis for the range and so $\text{rank}(\mathsf{A}) = k$. Now we need to show that there exists a set $\{\boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$ that forms a basis for the null space.

Let

$$\boldsymbol{w}_j = \mathrm{e}_j - \sum_{i=1}^{k} \alpha_{ji}\mathrm{e}_i \qquad k < j \leqslant n.$$

Observe that defined in this way each $\boldsymbol{w}_j$ contains one e_j and no other e_l for $j, l > k$. Consequently the set of column vectors $\{\boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$ is linearly independent and indeed $\{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_k, \boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$. is linearly independent and so forms a basis for $\mathbb{R}^n$. Now we have

$$\begin{aligned}
\mathsf{A}(\boldsymbol{w}_j) &= \mathsf{A}\mathrm{e}_j - \sum_{i=1}^{k} \alpha_{ji}\mathsf{A}\mathrm{e}_i, \qquad k < j \leqslant n, \\
&= \boldsymbol{v}_j - \sum_{i=1}^{k} \alpha_{ji}\boldsymbol{v}_i, \qquad k < j \leqslant n, \\
&= \boldsymbol{v}_j - \boldsymbol{v}_j, \qquad k < j \leqslant n, \\
&= \mathsf{0},
\end{aligned}$$

and so $\mathsf{A}\boldsymbol{w}_j = \mathbf{0}$ for $j = k+1, \ldots, n$. Thus

$$\text{nullity}(\mathsf{A}) \geqslant n - k. \tag{A.20}$$

Now we need to show that $\{\boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$ spans the null space of A. We know that $\{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_k, \boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$ is a basis for $\mathbb{F}^n$. Suppose $\boldsymbol{x}$ is in the null space of A and suppose

$$\boldsymbol{x} = \sum_{i=1}^{k} \beta_i \mathrm{e}_i + \sum_{j=k+1}^{n} \beta_j \boldsymbol{w}_j.$$

Observe that since $\sum_{j=k+1}^{n} \beta_j \boldsymbol{w}_j$ is in the null space, also $\sum_{i=1}^{k} \beta_i \mathrm{e}_i$ is in the null space,

i.e.

$$\begin{aligned} \mathsf{A}\boldsymbol{x} = \mathsf{A}\left(\sum_{i=1}^{k} \beta_i \mathrm{e}_i + \sum_{j=k+1}^{n} \beta_j \boldsymbol{w}_j \right) &= \mathsf{0} \\ \mathsf{A}\left(\sum_{i=1}^{k} \beta_i \mathrm{e}_i \right) &= \mathsf{0} - \mathsf{A}\left(\sum_{j=k+1}^{n} \beta_j \boldsymbol{w}_j \right) \\ \sum_{i=1}^{k} \beta_i \boldsymbol{v}_i &= \mathsf{0} - \mathsf{0} \\ &= \mathsf{0}. \end{aligned}$$

But $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ is linearly independent, so $\beta_i = 0$ for $i = 1, \ldots, k$ and

$$\boldsymbol{x} = \sum_{j=k+1}^{n} \beta_j \boldsymbol{w}_j.$$

Then $\{\boldsymbol{w}_{k+1}, \boldsymbol{w}_{k+2}, \ldots, \boldsymbol{w}_n\}$ is a basis for the null space of A therefore

$$\text{nullity}(\mathsf{A}) \leqslant n - k. \tag{A.21}$$

Equations (A.20) and (A.21) indicate that $\text{null}(\mathsf{A}) = n - \text{rank}(\mathsf{A})$ which completes the proof.

□

Now we will see that the rank of A is at least the number of nonzero eigenvalues of A as explained in Lemma A.2.5.

Definition A.2.4. (Number of nonzero eigenvalues) *Let $e(\mathsf{A})$ be the number of nonzero eigenvalues of A with multiplicity.*

Lemma A.2.5. *Let* A *be an* $n \times n$ *matrix. Then it holds that the number of nonzero eigenvalues, with multiplicity, is less than or equal to the rank of the matrix, i.e.*

$$e(\mathsf{A}) \leqslant \operatorname{rank}(\mathsf{A}). \tag{A.22}$$

Proof. Let $\operatorname{nullity}(\mathsf{A}) = k$. Then there exist k vectors in the basis for the null space of A, and so there exist k zero eigenvalues. Also there are at most n eigenvalues in total, and so

$$e(\mathsf{A}) \leqslant n - k.$$

However by Theorem A.2.3 (rank-nullity) we know that

$$\operatorname{rank}(\mathsf{A}) = n - k$$

thus

$$e(\mathsf{A}) \leqslant \operatorname{rank}(\mathsf{A}).$$

□

A.3 Positive definite and semidefinite matrices

A.3.1 Symmetric matrices

Definition A.3.1. (Symmetric matrix) *Let* $\mathsf{A} \in \mathbb{R}^n$. *Then* A *is symmetric if* $\mathsf{A} = \mathsf{A}^T$.

Definition A.3.2. (Unitary matrix) *Let* $\mathsf{U} \in \mathbb{R}^n$. *Then* U *is unitary if* $\mathsf{U}^T\mathsf{U} = \mathsf{I}$, *where* I *is the* $n \times n$ *identity matrix.*

Theorem A.3.3. (Schur factorisation) *Let* A *be a* $n \times n$ *symmetric matrix with real entries and let* λ_i *for* $i = 1, \ldots, n$ *be the eigenvalues of* A *in any order. Then*

$$\mathsf{A} = \mathsf{U}\Lambda\mathsf{U}^T \tag{A.23}$$

where U *is an* $n \times n$ *unitary matrix and* $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$.

Proof. For a proof see Horn and Johnson (1999) page 79. □

A.3.2 Characterization of positive definite and semidefinite matrices

Definition A.3.4. (Positive definite and semidefinite matrix) *Let* A *be an* $n \times n$ *real symmetric matrix. Then* A *is* positive definite*, denoted by* $\mathsf{A} \succ 0$*, if for all* $\boldsymbol{x} \in \mathbb{R}^n$

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} > 0, \qquad \boldsymbol{x} \neq 0. \tag{A.24}$$

A *is* positive semidefinite*, denoted by* $\mathsf{A} \succeq 0$*, if for all* $\boldsymbol{x} \in \mathbb{R}^n$

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} \geqslant 0, \qquad \boldsymbol{x} \neq 0. \tag{A.25}$$

Definition A.3.5. (*Quadratic form*) *Let* $f : \mathbb{R}^n \to \mathbb{R}$*. Then* f *is a quadratic form if can be expressed as*

$$f(\boldsymbol{x}) = \boldsymbol{x}^T \mathsf{A} \boldsymbol{x} = \sum_{i,j=1}^{n} a_{ij} x_i x_j. \tag{A.26}$$

Theorem A.3.6. (Cholesky factorisation) *Let* $\mathbf{A}$ *be a* $n \times n$ *positive definite matrix. Then there exist a nonsingular lower triangular matrix* $\mathsf{L} \in \mathbb{R}^{n \times n}$ *with positive diagonal entries, such that*

$$\mathsf{A} = \mathsf{L}\mathsf{L}^T. \tag{A.27}$$

is a unique cholesky factorisation of A*.*

Proof. For a proof see Horn and Johnson (1999) page 407.

□

Lemma A.3.7. *Let* A *be a positive definite* (*respectively semidefinite*) *matrix. Then all principal submatrices of* A *are positive definite* (*respectively semidefinte*) *matrices.*

Proof. Let A_k be any $k \times k$ principal submatrix of A. Since A_k is obtained by deleting $n-k$ rows and columns of A such that if the ith row is removed then the ith column is removed too, it follows that A_k is symmetric. Let $\boldsymbol{x} \in \mathbb{R}^k$ such that $\boldsymbol{x} \neq 0$. Now extend the vector by including zeros in all the $n-k$ entries to create a vector with n-entries, then

$$\boldsymbol{x}_k^T \mathsf{A}_k \boldsymbol{x}_k = \boldsymbol{x}^T \mathsf{A} \boldsymbol{x} > 0, \tag{A.28}$$

where the second inequality in (A.28) follows by the assumption that $\mathsf{A} \succ 0$. The same arguments can be used to prove the Lemma when A is positive semidefinite.

□

Corollary A.3.8. *The principal minors of a positive definite* (*respectively semidefinite*) *matrix are positive* (*respectively nonnegative*).

Proof. Using Lemma A.1.8 we know that all principal submatrices of a positive definite matrix are also positive definite, and so by Lemma A.3.11 it follows that the principal minors are positive. The same analysis works for positive semidefinite matrices.

□

Theorem A.3.9. *Let* A *be an* $n \times n$ *real symmetric matrix. Then* $\mathsf{A} \succ 0$

$$\iff \quad \boldsymbol{x}^T \mathbf{A} \boldsymbol{x} > 0 \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \tag{A.29a}$$

$$\iff \quad \lambda_i > 0, \text{ for } i = 1, \ldots, n, \text{ where } \lambda_i \text{ are the eigenvalues of } \mathsf{A} \tag{A.29b}$$

$$\iff \quad \text{All principal minors of } \mathsf{A} \text{ are positive} \tag{A.29c}$$

$$\iff \quad \mathsf{A} = \mathsf{L}\mathsf{L}^T \text{ for a nonsingular lower triangular matrix } \mathsf{L} \in \mathbb{R}^{n \times n}, \text{ with } l_{ii} > 0. \tag{A.29d}$$

Proof. If $\mathsf{A} \succ 0$ then (A.29a) follows by definition A.3.4. To prove (A.29b) we know that an eigenvalue and an eigenvector are a scalar λ and a vector $\boldsymbol{x}$ satisfying

$$\mathsf{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \qquad \boldsymbol{x} \neq 0,$$

thus

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} = \boldsymbol{x}^T \lambda \boldsymbol{x}$$

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} = \lambda \boldsymbol{x}^T \boldsymbol{x}$$

and so

$$\lambda = \frac{\boldsymbol{x}^T \mathsf{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}. \tag{A.30}$$

Observe that in (A.30) since $\boldsymbol{x} \neq 0$ the denominator is always a positive quantity. If $\mathsf{A} \succ 0$ then the numerator in (A.30) is positive and so λ (i.e. the eigenvalue) is also positive. On the other hand assume that $\lambda_i > 0$ for $i = 1, \ldots, n$. By the Theorem A.3.3 $\mathsf{A} = \mathsf{U}\Lambda\mathsf{U}^T$,

where U is unitary and $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, thus

$$\begin{aligned}
\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} &= \boldsymbol{x}^T \mathsf{U} \Lambda \mathsf{U}^T \boldsymbol{x} \\
&= (\mathsf{U}^T \boldsymbol{x})^T \Lambda (\mathsf{U}^T \boldsymbol{x}) \\
&= \sum_{i=1}^{n} \lambda_i \left(\mathsf{U}^T \boldsymbol{x}\right)_i^2 \\
&= \sum_{i=1}^{n} \lambda_i \left|\mathsf{U}^T \boldsymbol{x}\right|_i^2 > 0,
\end{aligned}$$

which indicates that $\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} > 0$ (i.e. $\mathsf{A} \succ 0$). Equation (A.29c) follows from Lemma A.3.8 and equation (A.29d) is just the cholesky factorization of A defined in Theorem A.3.6.

□

Similar results hold in the case of positive semidefinite matrices as explained in Theorem A.3.10.

Theorem A.3.10. *Let* A *be an* $n \times n$ *real symmetric matrix. Then* $\mathsf{A} \succeq 0$

$$\iff \quad \boldsymbol{x}^T \mathbf{A} \boldsymbol{x} \geqslant 0 \text{ for all } \boldsymbol{x} \in \mathbb{R}^n \tag{A.31a}$$

$$\iff \quad \lambda_i \geqslant 0, \text{ for } i = 1, \ldots, n, \text{ where } \lambda_i \text{ are the eigenvalues of } \mathsf{A} \tag{A.31b}$$

$$\iff \quad \text{All principal minors of } \mathsf{A} \text{ are nonnegative} \tag{A.31c}$$

$$\iff \quad \mathsf{A} = \mathsf{L}\mathsf{L}^T \text{ for some lower triangular matrix } \mathsf{L} \in \mathbb{R}^{n \times n}, \text{ with } l_{ii} \geqslant 0. \tag{A.31d}$$

Proof. The same analysis used in the proof of Theorem A.3.9 can be used to prove this theorem.

□

Remark To test if a matrix is positive definite or semidefinite either the eigenvalues or the principal minors can be checked.

Lemma A.3.11. *Let* A *be an* $n \times n$ *positive definite* (*respectively semidefinite*) *matrix. Then the trace and the determinant of* A *are positive* (*respectively nonnegative*).

Proof. From Corollary A.1.9

$$\operatorname{tr}(\mathsf{A}) = \sum_{i=1}^{n} \lambda_i$$

and from Corollary A.1.10

$$\det(\mathsf{A}) = \prod_{i=1}^{n} \lambda_i.$$

Since from Theorem A.3.9 we know that $\lambda_i > 0$, for $i = 1, \ldots, n$, then both $\operatorname{tr}(\mathsf{A})$ and $\det(\mathsf{A})$ are positive numbers. In the same way when dealing with positive semidefinite matrices we have $\lambda_i \geqslant 0$, for $i = 1, \ldots, n$, and so in that case both quantities are nonnegative numbers.

□

Definition A.3.12. (*Diagonal matrix*) *Let* $\mathsf{A} = (a_{ij})$ *be an* $m \times n$ *matrix. Then* A *is a* diagonal matrix *if the entries* $a_{ij} = 0$, *for all* $i \neq j$.

Lemma A.3.13. *Let* A *be an* $n \times n$ *diagonal matrix. Then* A *is positive definite* (*respectively semidefinite*) *if and only if the diagonal entries are positive* (*respectively nonnegative*).

Proof. Recall that the eigenvalues of a matrix are found by solving

$$\det(\lambda \mathsf{I} - \mathsf{A})$$

In the case of the diagonal matrix A

$$\begin{pmatrix} -a_{11} + \lambda & 0 & \dots & 0 \\ 0 & -a_{22} + \lambda & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & -a_{nn} + \lambda \end{pmatrix}.$$

The determinant of the last matrix by expansion of the row with the nonzero element each time

$$\det(\mathsf{A}) = (-a_{11} + \lambda) \dots (-a_{nn} + \lambda)$$

which indicates that the eigenvalues are equal to the elements in the main diagonal. Using Theorem A.3.9 A is positive definite if and only if the entries in the main diagonal (i.e. the eigenvalues) are positive. The same holds for the case when A is positive semidefinite using Theorem A.3.10. .

□

A.3.3 The frobenius norm

Definition A.3.14. (*Frobenius norm*)

$$\| \mathsf{A} \|^2 = \langle \mathsf{A}, \mathsf{A} \rangle = \mathrm{tr}(\mathsf{A}\mathsf{A}^T) = \sum_{i,j=1}^{n} a_{ij}^2 \tag{A.32}$$

APPENDIX B

CONIC PROGRAMMING

B.1 Convex optimization

An optimization problem is defined as

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & f_0(\boldsymbol{x}) \\ \text{s. t.} \quad & f_i(\boldsymbol{x}) \leqslant 0, \qquad i = 1, \ldots, m, \\ & h_i(\boldsymbol{x}) = 0, \qquad i = 1, \ldots, p, \end{aligned} \tag{B.1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is the optimization variable, $f_0(\boldsymbol{x})$ is the objective function with $f_0 : \mathbb{R}^n \to \mathbb{R}$, $f_i(\boldsymbol{x}) \leqslant 0$ are the inequality constraints with $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \ldots, m$ and $h_i(\boldsymbol{x}) = 0$ are the equality constraints with $h_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \ldots, p$.

Definition B.1.1. (Affine function) *Let $f : \mathbb{R}^n \to \mathbb{R}^m$. Then f is affine if there exists an $n \times m$ matrix A and a vector $\boldsymbol{b}$ in $\mathbb{R}^m$ such that*

$$f(\boldsymbol{x}) = \mathsf{A}\boldsymbol{x} + \boldsymbol{b}, \tag{B.2}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$.

When the functions $f_0, \ldots, f_m$ are convex and also the equality constraints $h_i(\boldsymbol{x})$ are affine, i.e. $h_i(\boldsymbol{x}) = a_i^T \boldsymbol{x} - b_i$, for $i = 1, \ldots, p$, then the optimization problem is convex and is given by

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^n} \quad & f_0(\boldsymbol{x}) \\ \text{s. t.} \quad & f_i(\boldsymbol{x}) \leqslant 0, \qquad i = 1, \ldots, m, \\ & a_i^T \boldsymbol{x} = b_i, \qquad i = 1, \ldots, p. \end{aligned} \tag{B.3}$$

In the convex optimization problem defined in (B.3) the functions $f : \mathbb{R}^n \to \mathbb{R}$ for $i = 0, \ldots, m$. Now given a subset of $\mathbb{R}^n$, i.e. $K \subset \mathbb{R}^n$, with certain characteristics, the associated problem with $f : K \to \mathbb{R}$ is a conic program. The mentioned K is a cone and its characteristics are analysed next.

B.2 Ordering in $\mathbb{R}^m$ and cones

Definition B.2.1. (Partial ordering) *A partial ordering in $\mathbb{R}^m$, denoted by $\succcurlyeq$ for all $\boldsymbol{v}$, $\boldsymbol{w} \in \mathbb{R}^m$, satisfies*

(i) Translation invariant, i.e. if $\boldsymbol{v} \succcurlyeq \boldsymbol{w}$ then $\boldsymbol{v} + \boldsymbol{u} \succcurlyeq \boldsymbol{w} + \boldsymbol{u}$, for all $\boldsymbol{u} \in \mathbb{R}^m$.

(ii) Scale invariant, i.e. if $\boldsymbol{v} \succcurlyeq \boldsymbol{w}$ and $\lambda \geqslant 0$ then $\lambda\boldsymbol{v} \succcurlyeq \lambda\boldsymbol{w}$, for all $\lambda \in \mathbb{R}^+$.

Orderings have several properties.

Definition B.2.2. (Properties of partial ordering) *Let $\succcurlyeq$ be the partial ordering defined in* B.2.1. *Then for all vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d} \in \mathbb{R}^m$ the following holds*

(i) The order is reflexive, i.e. $\boldsymbol{a} \succcurlyeq \boldsymbol{a}$.

(ii) The order is antisymmetric, i.e. if $\boldsymbol{a} \succcurlyeq \boldsymbol{b}$ and $\boldsymbol{b} \succcurlyeq \boldsymbol{a}$ then $\boldsymbol{a} = \boldsymbol{b}$.

(iii) The order is transitive, i.e. if $\boldsymbol{a} \succcurlyeq \boldsymbol{b}$ and $\boldsymbol{b} \succcurlyeq \boldsymbol{c}$ then $\boldsymbol{a} \succcurlyeq \boldsymbol{c}$.

(iv) The order is compatible with linear operators, that is

(a) Homogeneity: if $\boldsymbol{a} \succcurlyeq \boldsymbol{b}$ and $\lambda \geqslant 0$ then $\lambda\boldsymbol{a} \succcurlyeq \lambda\boldsymbol{b}$.

(b) Additivity: if $\boldsymbol{a} \succcurlyeq \boldsymbol{b}$ and $\boldsymbol{c} \succcurlyeq \boldsymbol{d}$ then $\boldsymbol{a} + \boldsymbol{c} \succcurlyeq \boldsymbol{b} + \boldsymbol{d}$.

The definition of a pointed convex cone is introduced next.

Definition B.2.3. (Pointed convex cone) *Let $K \in \mathbb{R}^m$ be a set. Then K is a pointed convex cone if it satisfies the following*

(i) K has nonempty interior and is closed under addition, i.e.

$$\boldsymbol{a}, \boldsymbol{b} \in K \Longrightarrow \boldsymbol{a} + \boldsymbol{b} \in K. \tag{B.4}$$

(ii) K is a conic set, i.e.

$$\boldsymbol{a} \in K, \lambda \geqslant 0 \Longrightarrow \lambda \boldsymbol{a} \in K. \tag{B.5}$$

(iii) K is pointed, i.e.

$$\boldsymbol{a} \in K, -\boldsymbol{a} \in K \Longrightarrow \boldsymbol{a} = 0. \tag{B.6}$$

Using both, the definition of partial ordering and the definition of a pointed convex cone the following definition is of particular interest.

Definition B.2.4. (Partial ordering induced by K) *Let K be a pointed convex cone. Then a partial ordering induced by $K \in \mathbb{R}^m$ is defined as*

$$\boldsymbol{a} \geqslant_K \boldsymbol{b} \iff \boldsymbol{a} - \boldsymbol{b} \geqslant_K 0 \iff \boldsymbol{a} - \boldsymbol{b} \in K. \tag{B.7}$$

Lemma B.2.5. *Let $\geqslant_K$ be the partial ordering induced by K. Then $\geqslant_K$ satisfies the properties in* B.2.2.

Proof. Let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ and $\boldsymbol{w} \in K$.

(i) Reflexivity. Let $\boldsymbol{x} \geqslant_K \boldsymbol{x}$. Then

$$\begin{aligned} \boldsymbol{x} &\geqslant_K \boldsymbol{x} \\ \boldsymbol{x} - \boldsymbol{x} &\in K \\ \mathbf{0} &\in K. \end{aligned}$$

Observe that if in equation (B.5) λ is chosen to be zero then for any $\boldsymbol{a}$ we have that $\mathbf{0} \in K$.

(ii) Antisymmetry. Let $\boldsymbol{x} \geqslant_K \boldsymbol{y}$ and $\boldsymbol{y} \geqslant_K \boldsymbol{x}$. Thus

$$\begin{aligned} \boldsymbol{x} &\geqslant_K \boldsymbol{y}, & \boldsymbol{y} &\geqslant_K \boldsymbol{x} \\ \boldsymbol{x} - \boldsymbol{y} &\in K, & \boldsymbol{y} - \boldsymbol{x} &\in K \\ & & -\boldsymbol{x} + \boldsymbol{y} &\in K. \end{aligned}$$

Since $\boldsymbol{x} - \boldsymbol{y} \in K$ and also $-\boldsymbol{x} + \boldsymbol{y} \in K$ by (B.6) $\boldsymbol{y} - \boldsymbol{x} = 0$ and so $\boldsymbol{y} = \boldsymbol{x}$.

(iii) Transitivity. Let $\boldsymbol{x} \geqslant_K \boldsymbol{y}$ and $\boldsymbol{y} \geqslant_K \boldsymbol{z}$, thus

$$\begin{aligned} \boldsymbol{x} &\geqslant_K \boldsymbol{y}, & \boldsymbol{y} &\geqslant_K \boldsymbol{z} \\ \boldsymbol{x} - \boldsymbol{y} &\in K, & \boldsymbol{y} - \boldsymbol{z} &\in K. \end{aligned}$$

Using (B.4)

$$\begin{aligned} \boldsymbol{x} - \boldsymbol{y} + \boldsymbol{y} - \boldsymbol{z} &\in K \\ \boldsymbol{x} - \boldsymbol{z} &\in K \\ \boldsymbol{x} &\geqslant_K \boldsymbol{z}. \end{aligned}$$

(iv) Compatibility with linear operators.

(a) Homogeneity. Let $\boldsymbol{x} \geqslant_K \boldsymbol{y}$ and $\lambda \geqslant 0$. Thus $\boldsymbol{x} - \boldsymbol{y} \in K$ and so using (B.5) $\lambda\boldsymbol{x} - \lambda\boldsymbol{y} \in K$ and finally $\lambda\boldsymbol{x} \geqslant_K \lambda\boldsymbol{y}$.

(b) Additivity. Let $\boldsymbol{x} \geqslant_K \boldsymbol{y}$ and $\boldsymbol{z} \geqslant_K \boldsymbol{w}$, then

$$\begin{aligned} \boldsymbol{x} \geqslant_K \boldsymbol{y}, \qquad & \boldsymbol{z} \geqslant_K \boldsymbol{w} \\ \boldsymbol{x} - \boldsymbol{y} \in K, \quad & \boldsymbol{z} - \boldsymbol{w} \in K. \end{aligned}$$

Using (B.4) we get

$$\begin{aligned} \boldsymbol{x} - \boldsymbol{y} + \boldsymbol{z} - \boldsymbol{w} &\in K \\ \boldsymbol{x} + \boldsymbol{z} &\geqslant_K \boldsymbol{y} + \boldsymbol{w}. \end{aligned}$$

□

Definition B.2.6. (Proper cone) *Let K be a pointed convex cone. Then K is a proper cone if it is closed and has nonempty interior.*

B.3 Definition of conic programming

B.3.1 Primal

Definition B.3.1. (Conic program) *Let K be a proper cone. Then a conic program is the optimization problem*

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ s.t. \quad & \mathsf{A}\boldsymbol{x} \geqslant_K \boldsymbol{b}, \end{aligned} \tag{B.8}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the optimization vector, $\boldsymbol{c} \in \mathbb{R}^n$ is the coefficients vector of the objective function, $\mathsf{A} \in \mathbb{R}^{m\times n}$ is the constraint matrix, and $\boldsymbol{b} \in \mathbb{R}^m$ is the right hand side for the constraints.

Examples of conic optimization are linear, conic quadratic and semidefinite programming.

B.3.2 Dual

Definition B.3.2. (Dual cone) *Let $K \in \mathbb{R}^m$ be a nonempty set. Then the dual cone of K, denoted by K_*, is given by*

$$K_* = \left\{\lambda \in \mathbb{R}^m \mid \lambda^T\boldsymbol{a} \geqslant 0,\ \forall \boldsymbol{a} \in K\right\}. \tag{B.9}$$

Let $\boldsymbol{x}$ be feasible and $\boldsymbol{\lambda} \in K_*$ for (B.8) then

$$\boldsymbol{\lambda}^T(\mathsf{A}\boldsymbol{x} - \boldsymbol{b}) \geqslant 0.$$

Thus given a $\boldsymbol{\lambda} \in K_*$ satisfying $\mathsf{A}^T\boldsymbol{\lambda} = \boldsymbol{c}$ it holds that

$$\boldsymbol{c}^T\boldsymbol{x} = (\mathsf{A}^T\boldsymbol{\lambda})^T\boldsymbol{x} = \boldsymbol{\lambda}^T\mathsf{A}\boldsymbol{x} \geqslant \boldsymbol{\lambda}^T\boldsymbol{b} = \boldsymbol{b}^T\boldsymbol{\lambda}.$$

Definition B.3.3. (Dual of a conic program) *The dual of a conic program is given by*

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & \boldsymbol{b}^T\boldsymbol{\lambda} \\ s.t. \quad & \mathsf{A}^T\boldsymbol{\lambda} = \boldsymbol{c}, \\ & \boldsymbol{\lambda} \geqslant_{K_*} \boldsymbol{0}, \end{aligned} \tag{B.10}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the optimization vector, $\boldsymbol{b} \in \mathbb{R}^m$ is the coefficients vector of the objective function, $\mathsf{A}^T \in \mathbb{R}^{n \times m}$ is the constraint matrix, and $\boldsymbol{c} \in \mathbb{R}^n$ is the right hand side for the constraints.

B.3.3 Conic duality theorem

Theorem B.3.4. (Conic duality theorem) *Let a conic program and its dual be given by*

$$c^* = \min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \boldsymbol{c}^T\boldsymbol{x} \mid \mathsf{A}\boldsymbol{x} \geqslant_K \boldsymbol{b} \right\}, \tag{\mathbf{P}}$$

$$b^* = \max_{\boldsymbol{\lambda}\in\mathbb{R}^m} \left\{ \langle \boldsymbol{b}, \boldsymbol{\lambda} \rangle \mid \mathsf{A}^*\boldsymbol{y} = \boldsymbol{c}, \boldsymbol{\lambda} \geqslant_{K_*} 0 \right\}. \tag{\mathbf{D}}$$

Then the following holds

(1) The duality is symmetric, i.e. the dual of the dual is the primal.

(2) The duality gap $\boldsymbol{c}^T\boldsymbol{x} - \langle \boldsymbol{b}, \boldsymbol{\lambda} \rangle$ is nonnegative for every feasible pair $(\boldsymbol{x}, \boldsymbol{\lambda})$.

(3a) If $(\mathbf{P})$ is bounded below and strictly feasible then $(\mathbf{P})$ is solvable and $c^* = b^*$.

(3b) If $(\mathbf{D})$ is bounded above and strictly feasible then $(\mathbf{P})$ is solvable and $c^* = b^*$.

(4) Assume that at least one of the problems $(\mathbf{P})$ or $(\mathbf{D})$ is bounded and strictly feasible. Then a primal-dual feasible pair $(\boldsymbol{x}, \boldsymbol{\lambda})$ is a pair of optimal solutions to the respective problems.

(4a) if and only if there is a zero duality gap, i.e. $\langle \boldsymbol{b}, \boldsymbol{\lambda} \rangle = \boldsymbol{c}^T\boldsymbol{x}$.

(4b) if and only if the complementary slackness is equal to zero, $\langle \boldsymbol{\lambda}, \mathsf{A}\boldsymbol{x} - \boldsymbol{b} \rangle = 0$.

Proof. For a proof see Nemirovski (2005) page 33. □

B.4 Examples of conic programs

B.4.1 Linear program

Definition B.4.1. (Linear program) *Let* $K = \mathbb{R}^n_+$. *Then the resultant optimization problem in* Definition B.3.1 *is a linear program.*

Lemma B.4.2. *Let* $K = \mathbb{R}^m_+$. *Then* $K_* = \mathbb{R}^m_+$, *i.e.* $\mathbb{R}^m_+$ *is self dual.*

Proof. Proving that $\mathbb{R}^m_+$ is self dual is equivalent to showing that given $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} \geqslant 0$ if and only if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geqslant 0$ for all $\boldsymbol{y} \geqslant 0$. If $\boldsymbol{x} \geqslant 0$ and $\boldsymbol{y} \geqslant 0$ then $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geqslant 0$. On the other hand if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geqslant 0$ for all $\boldsymbol{y} \geqslant 0$ then $\boldsymbol{x} \geqslant 0$ which completes the proof.

□

Thus similarly for the dual the following holds.

Definition B.4.3. (Dual of a linear program) *Let* $K_* = \mathbb{R}^m_+$. *Then the problem in* Definition B.3.3 *is the dual of the linear program given in* Definition B.4.1.

The Duality Theorem B.3.4 has stronger conditions for linear programming.

Theorem B.4.4. (Duality theorem for linear programming) *Let a primal linear program and its dual be given by* Definition B.4.1 *and* Definition B.4.3 *respectively. Then*

(1) The duality is symmetric, i.e. the dual of the dual is the primal.

(2) The duality gap $\boldsymbol{c}^T\boldsymbol{x} - \langle \boldsymbol{b}, \boldsymbol{\lambda} \rangle$ *is nonnegative for every feasible pair* $(\boldsymbol{x}, \boldsymbol{\lambda})$.

(3) The following are equivalent

(3a) Primal is feasible and bounded below

(3b) Dual is feasible and bounded above

(3c) Primal is solvable

(3d) Dual is solvable

(3e) Both primal and dual are feasible.

If one of the conditions in (3) holds then the optimal values of the primal and the dual are equal to each other.

Proof. For a proof see Nemirovski (2005) page 21.

□

B.4.2 Quadratic program

Definition B.4.5. (Lorentz Cone) *The Lorentz cone, also known as n-dimensional ice cream is defined by*

$$L^n = \left\{ \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n \mid \left(x_1^2 + \ldots + x_{n-1}^2 \right) \leqslant x_n^2 \right\}, n \geqslant 2.$$

Definition B.4.6. (Quadratic program) *Let K be a proper cone given by*

$$K = L^{n_1} \times L^{n_2} \times \ldots \times L^{n_k} = \left\{ y = \begin{pmatrix} y[1] \\ \vdots \\ y[k] \end{pmatrix} \mid y[i] \in L^{n_i}, i = 1, \ldots, k \right\}.$$

Then the optimization problem in (B.8) *is a quadratic program.*

Using

$$(\mathsf{A}\,;\boldsymbol{b}\,) = \begin{pmatrix} [\,\mathsf{A}_1\,;\boldsymbol{b}_1\,] \\ \vdots \\ [\,\mathsf{A}_k\,;\boldsymbol{b}_k\,] \end{pmatrix},$$

such quadratic program is equivalent to

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & \mathsf{A}_i \boldsymbol{x} \geqslant_{L^{n_i}} \boldsymbol{b}_i, \qquad i = 1, \ldots, k. \end{aligned}$$

Also, defining

$$(\mathsf{A}_i\,;\boldsymbol{b}_i\,) = \begin{pmatrix} \mathbf{D}_i & \mathbf{d}_i \\ p_i^T & q_i \end{pmatrix},$$

the quadratic program is now

$$\begin{aligned} \min_{\boldsymbol{x}\,\in\,\mathbb{R}^n} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ s.t. \quad & \|\mathsf{D}_i\boldsymbol{x} - d_i\|_2 \leqslant \boldsymbol{p}_i^T\boldsymbol{x} - q_i, \qquad i = 1,\ldots,k. \end{aligned} \tag{B.11}$$

To define the dual of a quadratic program the dual of a Lorentz cone is determined first.

Lemma B.4.7. *Let $K = \mathbb{L}^n$. Then $K_* = \mathbb{L}^n$, i.e. $\mathbb{L}^n$ is self dual.*

Proof. Proving that L^n is self dual is equivalent to showing that given

$$\begin{aligned} L^n &= \{(\boldsymbol{x},t) \mid \|\boldsymbol{x}\|_2 \leqslant t\}\,, \\ L_*^n &= \{(\boldsymbol{z},\tau) \mid \langle \boldsymbol{x}, z\rangle + t\tau \geqslant 0\}\,, \text{ for all } (\boldsymbol{x},t) \in L^n, \end{aligned} \tag{B.12}$$

it holds that $(z,\tau) \in L^n$ and $(z,\tau) \in L_*^n$. By contradiction assume $(z,\tau) \notin L^n$. By (B.12) for all $(\boldsymbol{x},\tau)$ it holds that $\|\boldsymbol{x}\|_2 \leqslant \tau$. Let $\overline{\boldsymbol{x}} = -\frac{\tau}{\|z\|}z$, thus

$$\langle \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}\rangle = -\frac{\tau^2}{\|z\|^2}\langle z, z\rangle,$$

and $\|\hat{\boldsymbol{x}}\| = \tau$, which is $(\hat{\boldsymbol{x}},\tau) \in L^n$. Using the definition of a dual cone

$$0 \leqslant \langle \hat{\boldsymbol{x}}, z\rangle + \tau^2 = -\frac{\tau}{\|z\|}\langle z, z\rangle + \tau^2 = -\tau\|z\| + \tau^2 < 0$$

where the second inequality follows from the fact that $\|z\| > \tau$ leading to a contradiction.

□

The dual of a conic program has the form

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & \boldsymbol{b}^T \boldsymbol{\lambda} \\ s.t. \quad & \mathsf{A}^T \boldsymbol{\lambda} = \boldsymbol{c}, \\ & \boldsymbol{\lambda} \succcurlyeq_{K_*} \mathbf{0}, \end{aligned}$$

where $\lambda = (\lambda_1, \ldots, \lambda_k^T)$. Letting $\lambda_i = \begin{pmatrix} \mu_i \\ \nu_i \end{pmatrix}$ with $\nu_i \in \mathbb{R}$ the dual of a quadratic program is obtained.

Definition B.4.8. (Dual of a quadratic program) *The dual of the quadratic program* (B.11) *is*

$$\begin{aligned} \max_{\mu_i \nu_i \in \mathbb{L}^n} \quad & \sum_{i=1}^{k} \left(\mu_i^T d_i + \nu_i q_i \right) \\ s.t. \quad & \sum_{i=1}^{k} \left(D_i^T \mu_i + \nu_i p_i \right) = c, \\ & \|\mu_i\|_2 \leqslant \nu_i, \qquad i = 1, \ldots, k. \end{aligned} \tag{B.13}$$

Quadratic programs appear sometimes in the form of convex quadratic forms.

Lemma B.4.9. *A convex quadratic form* $\boldsymbol{x}^T\mathsf{Q}\boldsymbol{x}+\boldsymbol{q}^T\boldsymbol{x}+r$ *where* Q *is an* $n\times n$ *symmetric matrix, is conic quadratic representable, i.e.*

$$\left\{(\boldsymbol{x},t) \mid \boldsymbol{x}^T\mathsf{Q}\boldsymbol{x}+\boldsymbol{q}^T\boldsymbol{x}+r \leqslant t\right\}= \left\{(\boldsymbol{x},t) \left\| \frac{\mathsf{D}\boldsymbol{x}}{\frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r-1)}{2}} \right\|_2 \leqslant \frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r+1)}{2} \right\}. \tag{B.14}$$

Proof. The matrix $\mathsf{Q}\in\mathbb{S}^n$ and therefore by Theorem A.3.3 can be factorise as $\mathsf{Q}=\mathsf{D}^T\mathsf{D}$. Observe that

$$\begin{aligned} \left\| \frac{\mathsf{D}\boldsymbol{x}}{\frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r-1)}{2}} \right\|_2 &\leqslant \frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r+1)}{2} \\ \iff \|\,\mathsf{D}\boldsymbol{x}\,\|_2^2 + \frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r-1)^2}{4} &\leqslant \frac{(t-\boldsymbol{q}^T\boldsymbol{x}-r+1)^2}{4} \\ \iff \|\,\mathsf{D}\boldsymbol{x}\,\|_2^2 &\leqslant t-\boldsymbol{q}^T\boldsymbol{x}-r \end{aligned}$$

where $\|\,\mathsf{D}\boldsymbol{x}\,\|_2^2 = \boldsymbol{x}^T\mathsf{D}^T\mathsf{D}\boldsymbol{x}$.

□

B.4.3 Semidefinite program

Denote the linear mapping $\mathcal{A}:\mathbb{R}^n\to\mathbb{S}^m$ with

$$\mathcal{A}\boldsymbol{x}=\sum_{i=1}^{n} x_i\mathsf{A}_i,$$

where $\boldsymbol{x}=(x_1,\ldots,x_n)\in\mathbb{R}^n$ and $\mathsf{A}_1,\ldots,\mathsf{A}_n$ are matrices from $\mathbb{S}^m$.

Also denote the linear mapping $\mathcal{A}^* : \mathbb{S}^m \to \mathbb{R}^n$ which corresponds to the conjugate transpose of $\mathcal{A}$ and is given by

$$\langle \mathcal{A}^*(\mathsf{X}), \boldsymbol{y} \rangle = \langle \mathcal{A}(\boldsymbol{y}), \mathsf{X} \rangle, \qquad \text{for all } \boldsymbol{y} \in \mathbb{R}^n.$$

Definition B.4.10. (Semidefinite program) *Let $K = \mathbb{S}^m$ be a proper cone. Then the optimization problem in* (B.8) *is a semidefinite program, i.e.*

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & \mathcal{A}\boldsymbol{x} - \mathsf{B} \succeq 0, \end{aligned} \tag{B.15}$$

where $\succeq$ stands for $\geqslant_{\mathbb{S}^m_+}$.

In explicit form a semidefinite program is

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & x_1 \mathsf{A}_1 + \ldots, + x_n \mathsf{A}_n - \mathsf{B} \succeq 0. \end{aligned}$$

Similarly the dual of a semidefinite program can be defined, noting first that the cone of semidefinite matrices is self dual.

Lemma B.4.11. *Let $K = \mathbb{S}^m_+$. Then $K_* = \mathbb{S}^m_+$, i.e. $\mathbb{S}^m_+$ is self dual.*

Proof. Proving that $\mathbb{S}^m_+$ is self dual is equivalent to showing that $\mathsf{X} \in \mathbb{S}^n_+$ is positive

semidefinite if and only if $\langle \mathsf{X}, \mathsf{Y} \rangle \geqslant 0$ for all $\mathsf{Y} \in \mathbb{S}^m_+$. If $\mathsf{X} \geqslant 0$ and $\mathsf{Y} \geqslant 0$ then

$$\langle \mathsf{X}, \mathsf{Y} \rangle = \operatorname{tr}(\mathsf{XY}) = \operatorname{tr}(\mathsf{X}^{\frac{1}{2}}\mathsf{X}^{\frac{1}{2}}\mathsf{Y}^{\frac{1}{2}}\mathsf{Y}^{\frac{1}{2}}) = \operatorname{tr}(\mathsf{X}^{\frac{1}{2}}\mathsf{Y}^{\frac{1}{2}}\mathsf{Y}^{\frac{1}{2}}\mathsf{X}^{\frac{1}{2}}) = \|\mathsf{X}^{\frac{1}{2}}\mathsf{Y}^{\frac{1}{2}}\| \geqslant 0.$$

On the other hand $\langle \mathsf{X}, \mathsf{Y} \rangle \geqslant 0$ for all $\mathsf{X} \geqslant 0$. Let $\mathsf{X} = \boldsymbol{x}\boldsymbol{x}^T \succeq 0$ with $\boldsymbol{x} \in \mathbb{R}^n$ and

$$0 \leqslant \langle \mathsf{X}, \mathsf{Y} \rangle = \operatorname{tr}(\mathsf{Y}\boldsymbol{x}\boldsymbol{x}^T) = \sum_{i,j=1}^{n} Y_{ij} x_i x_j = \boldsymbol{x}^T \mathsf{Y} \boldsymbol{x},$$

which indicates that $\mathsf{Y} \succeq 0$ which completes the proof.

□

Definition B.4.12. (Dual of a semidefinite program) *The dual of the semidefinite program* (B.15) *is*

$$\begin{array}{ll} \max\limits_{\Lambda \in \mathbb{S}^m} & \langle \mathsf{B}, \Lambda \rangle \\ s.t. & \mathcal{A}^*\Lambda = \boldsymbol{c}, \\ & \Lambda \succeq 0. \end{array} \tag{B.16}$$

Observe that $\langle \mathcal{A}^*, \Lambda \rangle_{\mathbb{S}^m} = \langle \mathcal{A}^*(\Lambda), \boldsymbol{x} \rangle_{\mathbb{R}^n}$, for all Λ, which is

$$\begin{aligned} \operatorname{tr}\big((\mathcal{A}\boldsymbol{x})\Lambda\big) &= \operatorname{tr}\left(\left(\sum_{i=1}^{n} x_i \mathsf{A}_i\right)\Lambda\right) \\ &= \sum_{i=1}^{n} x_i \operatorname{tr}(\mathsf{A}_i\Lambda) \\ &= \big(\operatorname{tr}(\mathsf{A}_1\Lambda), \ldots, (\mathsf{A}_n\Lambda)\big) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}. \end{aligned} \tag{B.17}$$

Using (B.17) the dual of a semidefinite program can also be given in explicit form

$$\begin{aligned} \max_{\Lambda \in \mathbb{S}^m} \quad & \langle \mathsf{B}, \Lambda \rangle \\ s.t. \quad & \langle \mathsf{A}_j, \Lambda \rangle = c_j, \qquad j = 1, \ldots, n, \\ & \Lambda \succeq 0. \end{aligned}$$

Next the simple lemma that specifies when a quadratic function is a nonnegative polynomial is introduced.

Lemma B.4.13. (Simple Lemma) *Let* $\mathsf{A} \in \mathbb{S}^n$, $\boldsymbol{b} \in \mathbb{R}^n$ *and* $\boldsymbol{c} \in \mathbb{R}$. *Then*

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} + 2\boldsymbol{b}^T \boldsymbol{x} + c \geqslant 0 \tag{B.18}$$

if and only if

$$\begin{pmatrix} \boldsymbol{c} & \boldsymbol{b}^T \\ \boldsymbol{b} & \mathsf{A} \end{pmatrix} \succeq 0 \tag{B.19}$$

Proof. Let $\begin{pmatrix} t \\ \boldsymbol{x} \end{pmatrix}$ be a vector with $t \in \mathbb{R}$, $t \neq 0$ and $\boldsymbol{x} \in \mathbb{R}^n$. Assume that the matrix (B.19) is positive semidefinite. Then

$$\begin{pmatrix} t & \boldsymbol{x} \end{pmatrix} \begin{pmatrix} \boldsymbol{c} & \boldsymbol{b}^T \\ \boldsymbol{b} & \mathsf{A} \end{pmatrix} \begin{pmatrix} t \\ \boldsymbol{x} \end{pmatrix} \geqslant \mathbf{0}$$

which is

$$\boldsymbol{x}^T \mathsf{A} \boldsymbol{x} + 2\boldsymbol{b}^T \boldsymbol{x} + \boldsymbol{c} \geqslant \mathbf{0}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}$ and in particular if $t = 1$.

$\square$

BIBLIOGRAPHY

Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., and Sviridenko, M. (1999). Approximation schemes for minimizing average weighted completion time with release dates. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 32 –43.

Andrilli, S. F. and Hecker, D. (2003). *Elementary linear algebra*. Gulf Professional Publishing, 3 edition.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Baker, K. R., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1983). Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 31:381–386.

Beale, E. and Tomlin, J. (1970). Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J.Lawrence, editor, *Proceedings of the Fifth International Conference on Operational Research*, pages 447–454, London. Tavistock Publications.

Boyd, S. and Vandenberghe, L. (2008). *Convex optimization*. Cambridge university press, sixth edition.

Brucker, P., Garey, M. R., and Johnson, D. S. (1977). Scheduling equal-length tasks under treelike precedence constraints to minimize maximun lateness. *Mathematics of Operations Research*, 2(3):275–284.

Brucker, P. and Knust, S. (2006). Complexity results for scheduling problems. Link http://www.mathematik.uni-osnabrueck.de/research/OR/class.

Camion, P. (1965). Charaterization of totally unimodular matrices. *Proceedings of the American Mathematical Society 16*, pages 1068–1073.

Chor, B. and Sudan, M. (1998). A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11:511–523.

Du, J. and Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495.

Du, J., Leung, J. Y.-T., and Young, G. H. (1991). Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2):219–236.

Garey, M. R. and Johnson, D. S. (1978). "strong"np-completeness results: Motivation, examples, and implications. *Journal of Association for Computing Machinery*, 25(3):499–508.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326.

Gröetschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method an dits consequences in combinatorial optimization. *Combinatorica*, 1:169–197.

Hall, L. A. (1997). *Approximation algorithms for NP-hard problems*, chapter Approximation algorithms for scheduling, pages 1–45. PWS Publishing Co., Boston, MA, USA.

Horn, R. A. and Johnson, C. R. (1999). *Matrix analysis*. Cambridge University Press.

IBM ILOG CPLEX (2009). IBM ILOG CPLEX Optimizer: High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Jinjiang, J. (1992). The NP-hardness of the single machine common due date weighted tardiness problem. *System Science and Mathematical Sciences*, 5(4):328–333.

Karger, D. R., Motwani, R., and Sudan, M. (1994). Approximate graph coloring by semidefinite programming. In *IEEE Sumposium on Foundations of Computer Science*, pages 2–13.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103, New York.

Kočvara, M. and Stingl, M. (2003). PENNON a code for convex nonlinear and semidenite programming. *Optimization Methods and Software*, 8(3):317–333.

Labetoulle, J., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1984). Preemptive scheduling of uniform machines subject to release dates. *Progress in combinatorial optimization (Waterloo, Ont., 1982)*, pages 245–261.

Lawler, E. L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546.

Lawler, E. L. (1977). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1993). Sequencing and scheduling: algorithms and complexity. In *Handbook in Operations Research and Management Science*, volume 4. Elsevier Science Publishers B. V., Amsterdam.

Lawler, E. L. and Moore, J. M. (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16:77–84.

Lenstra, J. K., Kan, A. H. G. R., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Marhematics*, 1:343–362.

Leung, J. Y.-T., editor (2004). *Handbook of Scheduling. Algorithms, Models, and Performance Analysis*, Computer and Information Science. Chapman & Hall/CRC.

Löfberg, J. (2004). YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan.

Lovász, L. (1979). On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7.

Morton, T. E. and Pentico, D. W. (1993). *Heuristic scheduling systems.* John Wiley & Sons, Inc.

Nemirovski, A. (2005). Lectures on modern convex optimization. Department ISYE, Georgia Institute of Technology.

Sahni, S. (1976). Algorithms for scheduling independent tasks. *Association for Computing Machinery Journal*, 23:116–127.

Schrijver, A. (1986). *Theory of Linear and Integer Programming.* John Wiley & Sons Ltd, New York.

Schulz, A. and Skutella, M. (2002). The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133.

Sethuraman, J. and Squillante, M. S. (1999a). Optimal scheduling of multiclass parallel machines. In *Proceeding of 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 963–964.

Sethuraman, J. and Squillante, M. S. (1999b). Optimal stochastic scheduling in multiclass parallel queues. In *ACM Sogmetrics Conference on Measurement and Modeling of Computer Systems*, pages 93–102.

Shor, N. Z. (1987). Quadratic optimization problems. *Tekhnicheskaya Kibernetika*, 1:128–139.

Sitters, R. (2004). *Complexity and approximation in routing and scheduling*. PhD thesis, Eindhoven University of Technology.

Skutella, M. (1998). Semidefinite relaxations for parallel machine scheduling. In *In IEEE Symposium Foundations of Computer Science*, pages 472–481.

Skutella, M. (1999). Convex quadratic programming relaxatins for network scheduling problems. In *European Symposium on Algorithms*, pages 127–138.

Skutella, M. (2001). Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM*, 48(2):206–242.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Q.*, 3(59-66).

Sturm, J. F. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653.

Tamir, A. (1976). On totally unimodular matrices. *Networks 6*, pages 373–382.

Tomlin, J. A. (1988). Special ordered sets and an application to gas supply operations planning. *Mathematical Programming*, 42:69–84.

Truemper, K. (1998). *Matroid decomposition*. Leibniz, Texas.

Ullman, J. D. (1975). Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393.

van den Akker, J., Diepen, G., and Hoogeveen, J. (2010). Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs. *Journal of Scheduling*, pages 1–16.

Veinoot, A. F. J. and Dantzig, G. (1968). Integral extreme points. *SIAM Review*, 3(10):371–372.

Wright, S. J. (1997). *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia.

Ye, Y. (2001). A 0.699 approximation algorithm for max-bisection. *Mathematical Programming*, 90(1):101–111.