Tesis de Posgrado

Un enfoque poliedral del problema de secuenciamiento de tareas en procesadores heterogéneos bajo relaciones de precedencia

Coll, Pablo E.

2002

Tesis presentada para obtener el grado de Doctor en Ciencias de la Computación de la Universidad de Buenos Aires

Tesis Doctoral

Un enfoque poliedral del problema de secuenciamiento de tareas en procesadores heterogéneos bajo relaciones de precedencia

Pablo E. Coll
pecoll@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales - UBA

Director: Celso C. Ribeiro (PUC-Rio)
Co-Director: Cid Carvalho de Souza (UNICAMP)

6 de septiembre de 2002

A mis padres, que supieron incentivar la curiosidad científica que todo niño lleva en su interior

Agradecimientos

- A Celso C. Ribeiro, mi director, por su invaluable guía, apoyo, estímulo y paciencia en todos estos años, aún en los momentos difíciles .
- A Cid Carvalho de Souza, mi co-director, por las múltiples fructiferas discusiones y charlas que mantuvimos y que me enriquecieron en el plano académico y en el humano.
- A mi mujer Beatriz Sauber por la paciencia y apoyo en todos estos años.
- A mis hermanos María Laura, Gustavo y Guillermo y al resto de mi familia, por haberme apoyado en estos años difíciles para poder dedicarme a esta investigación.
- A Irene Loiseau, por haberme iniciado en este camino de mil kilometros incentivándome a dar el primer paso y muchos otros más.
- A Isabel Mendez Díaz y a Javier Marenco por todas las provechosas observaciones que aportaron en numerosas charlas que hemos tenido en estos años.
- A Claudia Justel, Carlos Carballo, Enrique Pujals, Ernesto G. Birgin y Débora Pretti Ronconi por que me hicieron sentir como en su casa en cada uno de mis ocho viajes a Brasil.
- A Eduardo Uchoa y demás alumnos de doctorado y maestrado de Celso y Cid, por la ayuda que me brindaron con gran generosidad en cada viaje.
- A Guillermo Durán, Paula Zabala, Min Chih Lin y en general a todos los docentes, no docentes y alumnos del Departamento de Computación y de la Facultad de Ciencias Exactas y Naturales de la UBA por el buen clima de trabajo que supieron generar.

Resumen

Estudiamos un algoritmo exacto para el problema de secuenciamiento de tareas en procesadores heterogéneos bajo relaciones de precedencia.

En este problema contamos con conjuntos de procesadores y tareas. Las tareas están descriptas por sus duraciones y por un digrafo acíclico de precedencias. El conjunto de procesadores heterogéneos es tal que no pueden establecerse relaciones entre procesadores, tareas y tiempos de procesamiento. No se permitiran las interrupciones de las tareas una vez comenzadas. El objetivo del problema es minimizar el tiempo necesario para completar todas las tareas.

Una aplicación se presenta en el contexto de asignar tareas de programas paralelos en computadoras multiprocesador o sistemas distribuidos.

Se propone una nueva formulación como problema de programación lineal entera. Esta formulación tiene menos restricciones y variables que las formulaciones previas.

Se estudia un poliedro acotado consistente de un subconjunto de desigualdades de la nueva formulación. El poliedro de *partición en ordenes lienales* (PLO) por sus siglas en inglés. es una relajación y una proyección del poliedro original. Se estudia en detalle el poliedro PLO y se encuentra que muchos resultados son una generalización de aquellos hallados para el poliedro de partición en subgrafos completos [21]. Las desigualdades obtenidas para este poliedro es muy probable que jueguen un importante papel en la formulación y resolución exacta a través de algoritmos de bifurcación y corte de una familia de problemas de secuenciamiento de múltiples máquinas.

Finalmente, se desarrolla un algoritmo de bifurcación y corte basado en los cortes específicos desarrollados para el problema en cuestión e igualdades que definen facetas del poliedro PLO y se detallan los resultados computacionales.

Abstract

We study an exact algorithm for the problem of scheduling unrelated processors under precedence constraints.

In this problem we are given sets of tasks and processors. The tasks are described by their processing times and by a directed acyclic graph representing the precedence constraints. The set of unrelated processors is such that no proportional relations can be established between processors, tasks, and processing times. Preemption is not allowed. The goal of the optimization problem is the minimization of the makespan, i.e., time needed to complete all tasks.

An application arises in the context of scheduling tasks of parallel programs in multi-processor computers or distributed systems.

A new integer programming formulation for the problem is proposed. This formulation has less contraints and variables than previous formulations.

A polytope consisting of the subset of inequalities of the new formulation that defines a partition in linear orderings is studied. The *partition in linear orderings* (PLO) polytope is a relaxation and a projection of the original polytope. The PLO polytope is studied in detail and many results are found to be generalizations of those of the clique partition polytope [21]. The inequalities obtained for this polyhedron are likely to play an important role in the formulation and exact solution via branch-and-cut algorithms of a family of multiple machines scheduling problems.

Finally, a branch-and-cut algorithm is developed based on cuts specific to the scheduling problem under consideration and on facet defining inequalities for the PLO polytope. Computational results are reported.

Contents

Chapter 1

Introduction

We study an exact algorithm for the problem of scheduling unrelated processors under precedence constraints. A new formulation for the problem is proposed. A polytope consisting of the subset of inequalities of the new formulation that defines a partition in linear orderings is studied. This polytope named *partition in linear orderings* (PLO) polytope is a relaxation and a projection of the original polytope. PLO polytope is studied in detail and much of these results are found to be a generalization of those of the clique partition polytope [21]. The inequalities obtained for this polyhedron are likely to play an important role in the formulation and exact solution via branch-and-cut algorithms of a huge family of multiple machines scheduling problems.

Finally a branch-and-cut algorithm is developed based on cuts specific to the scheduling problem under consideration and on facet defining inequalities for the PLO polytope. An important part of our test cases instances are solved exactly and for the rest of them good estimates of the optimality gap are obtained. In this work, we refer to the "optimality gap" as the relative difference between the best known feasible solution and the lower bound.

1.1 The Scheduling Problem

In the problem of scheduling unrelated processors under precedence constraints we are given sets of tasks and processors. The set of tasks is described by a directed acyclic graph that gives the precedence constraints between tasks and the duration times. The set of unrelated processors is such that no proportional relations can be established between processors, tasks, and processing times. No task may be interrupted once it begins to be processed in a machine. The goal of the optimization problem is the minimization of the time needed to complete the processing of all tasks.

An application of this problem arises in the context of scheduling tasks of parallel programs in a multi-processor computer or distributed systems. Parallel programs can be represented as a set of interrelated tasks which are sequential units. In a heterogeneous multiprocessor system, we not only have to determine how many, but also which processors should be allocated to an application, as well as which processors are going to be assigned to each task.

This problem belongs to the class of NP-hard problems [15]. Two essentially different approaches can be used to tackle problems of this class.

Approximate algorithms, or heuristics, provide good, but not necessarily optimal solutions. It

is generally difficult to provide, within these algorithmic schemes, guarantees of the quality for the solutions obtained. The advantage of this approach is that problems of large size can be tackled in reasonable time. Porto and Ribeiro in [40] followed this strategy to tackle the problem of scheduling unrelated processors under precedence constraints. Their results will be extensively used in this thesis as upper bounds for our algorithm.

The other alternative is to use exact approaches, such as a branch-and-cut algorithm. The main drawback of this approach is the limited size of the problems that can be solved in practice. However in the last two decades cutting planes and branch-and-cut algorithms for integer programming have been used successfully to solve reasonable sized instances of hard combinatorial optimization problems such as the traveling salesman problem (TSP) [36, 38], the linear ordering problem [17], the clustering problem [20], the airline crew-scheduling problem [23], the multicommodity flow problem [6], the maximum cardinality stable set problem [42], and the rectilinear Steiner tree problem [45], among others. This success was the motivation for the work developed in this thesis.

Cutting planes and branch-and-cut algorithms are strongly based in the theory of polyhedral combinatorics, whose main concepts are summarized next.

1.2 Polyhedral Combinatorics

Polyhedral theory is one of the most powerful techniques available to understand and solve hard combinatorial optimization problems. The framework of this polyhedral approach can be briefly described as follows. Most often a combinatorial optimization problem can be formulated as a mathematical programming problem in which we have to minimize a linear objective function subject to a set of linear inequalities with some variables restricted to take integer values. This is equivalent to the minimization of this objective function restricted to the integral points within the convex hull of all feasible solutions of the problem. A polynomial time algorithm can be designed to solve the problem when a good description of the convex hull is known. By a good description, we mean that there is a polynomial time algorithm to check whether a given rational point satisfies all linear inequalities defining the convex hull, and if not, to provide a violated inequality.

However, when the combinatorial optimization problem is NP-hard, there is no hope to find a good description of the convex hull, unless P = NP. Nevertheless, even in such cases the polyhedral approach was proved to be quite successful. The main idea is to consider high dimensional faces, preferably facets, of a partial description of the convex hull and to iteratively strengthen the linear formulation by adding these inequalities if they are violated by the current fractional solution. This cutting plane scheme has been particularly successful when embedded in a branch-and-bound framework. The implementation of these so called branch-and-cut algorithms have been greatly simplified in the last years since the appearance of frameworks such as ABACUS [44], designed for the implementation of linear programming based branch-and-bound algorithms. In this thesis, we have found both strong valid inequalities for the scheduling problem and facet defining inequalities for the PLO polytope. These inequalities and other strong valid inequalities specific for the scheduling problem were used as cuts in a branch-and-cut algorithm implemented with the ABACUS software.

1.3 Basic results

In this section, we introduce the basic concepts, definitions and notations required for a better understanding of this thesis. The material presented here includes only the topics in graph theory, linear and integer programming and set theory which are relevant for this text. More thorough presentations of these topics can be found in the references given in the following sections.

1.3.1 Graph Theory

This section summarizes the basic graph theory background. Most of the definitions were extracted from [2, 5, 22].

Graph. A *graph* $G = (V, E)$ consists of a nonempty set V of *vertices* and a set E of *edges*, together with a relation of incidence that associates with each edge two vertices, called its *ends*. A *complete* graph is one in which any two of its vertices are joined by an edge.

Subgraph. A graph $H = (W, F)$ is called a *subgraph* of G if $W \subseteq V, F \subseteq E$, and $F \subseteq W \times W$. A subgraph $H = (W, F)$ of G is called a *induced subgraph* of G if every edge in E with both ends in W is also present in F.

Clique. A *clique* of a graph G is a complete subgraph of G.

Directed graph or digraph. A *directed graph*, or *digraph*, $D = (N, A)$, is a graph in which each edge is assigned a direction, one end being designated its *tail* and the other its *head*. We will use the word *arc* for a directed edge, and denote the set of arcs by A. The set of vertices will be denoted by N. We shall denote the cardinals of sets N and A by n and m, respectively.

Underlying graph. The *underlying graph* $G(D)$ of a digraph D is the graph obtained from D by ignoring the orientations of the arcs.

Complete digraph. A *complete digraph* D_n, is a digraph that has for each pair of vertices u and v two opposite arcs (u, v) and (v, u).

Path and cycle. A *path* of length $h - 1$ is an arc set of the form $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{h-1}, v_h)\}$ where $v_i \neq v_j$ for $1 \leq i < j \leq h$. If $P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{h-1}, v_h)\}$ is a path then $P \cup (v_h, v_1)$ is a *cycle* of length h.

Directed acyclic graph. A directed graph containing no cycles is called a *directed acyclic graph* (DAG).

Tournament. A *tournament* is a directed graph whose underlying graph is simple and complete.

Transitive closure. Let D be a directed graph. The *transitive clousure* of D, denoted D^*, is the digraph with the same set of vertices as D, but which has an arc (u, v) whenever there is a path from u to v in D.

Arc sets. We will define two classes of arc sets. Let $D = (N, A)$ be a directed graph and $B \subseteq N$ a set of vertices. We shall denote by $A(B)$ the subset of arcs with both vertices in B. For $C \subseteq N$, we denote by $\delta(B, C)$ the set of arcs (u, v) with vertex u in B and the vertex v in C. This arc set is called the *directed cut* of B and C.

Matching. Given a digraph $D = (N, A)$, a *matching* is a set $A' \subseteq A$ of arcs such no arcs of A' are adjacent. The size of a matching is the number of arcs in the set.

Hamiltonian cycle. Given a digraph $D = (N, A)$, a *hamiltonian cycle* is a cycle that passes through every vertex exactly once.

Chord and k-chord of a cycle. A *chord* of a cycle C is an arc that joins two non adjacent vertices of C. A *k-chord* of C is an arc that joins two vertices of the cycle separated by $k-1$ intermediate vertices in the cycle.

Inverse arc, inverse set and transposed digraph. Let $a = (u, v)$ be an arc. We will denote the *inverse arc* of a as $\overline{a} = (v, u)$. In general given a arc set A we will denote the *inverse set* of A as the set $\overline{A}$ of all inverse arcs of A. Given a digraph $D = (N, A)$, the *transposed digraph* of D, denoted by D^T, is the digraph whose vertex set is N and whose arc set is $\overline{A}$.

1.3.2 Linear Algebra and Polyhedral Theory

This section summarizes the basic linear algebra and polyhedral theory background necessary for a better understanding of the text. Most of the definitions were extracted from [10, 37, 46].

Convex combination. Given a set $S \subseteq \mathrm{R}^n$, a point $x \in \mathrm{R}^n$ is a *convex combination* of points of S if there exists a finite set of points $\{x^i\}_{i=1}^h$ in S and $\lambda \in \mathrm{R}_+^h$ with $\sum_{i=1}^h \lambda_i = 1$ and $x = \sum_{i=1}^h \lambda_i x^i$.

Convex hull. The *convex hull* of S (denoted by $conv(S)$) is the set of all points that can be written as a convex combination of the elements of S.

Linear independency. A set of points $x^1, \ldots, x^h \in \mathrm{R}^n$ is *linearly independent* if the unique solution of the system $\sum_{i=1}^h \alpha_i x^i = 0$ is $\alpha_i = 0$ for $i = 1, \ldots, h$.

Affine independency. A set of points $x^1, \ldots, x^h \in \mathrm{R}^n$ is *affinely independent* if the unique solution of the system $\{\sum_{i=1}^h \alpha_i x^i = 0,\ \sum_{i=1}^h \alpha_i = 0\}$ is $\alpha_i = 0$ for $i = 1, \ldots, h$.

Polyhedron. A *polyhedron* $P \subseteq \mathrm{R}^n$ is the solution set of a finite system of linear inequalities; i.e. $P = \{x : x \in \mathrm{R}^n, Ax \leq b\}$, where $A \in \mathrm{R}^{m \times n}$ and $b \in \mathrm{R}^m$. A polyhedron $P \subseteq \mathrm{R}^n$ is *bounded* if there exists $M \in \mathrm{R}_+$ such that $P \subseteq \{x : x \in \mathrm{R}^n, -M \leq x_j \leq M, \text{for } j = 1, \ldots, n\}$. A bounded polyhedron is called a *polytope*.

Dimension. A polyhedron P is of dimension k, denoted by $\dim(P)=k$, if the maximum number of affinely independent points in P is $k+1$.

Full-dimension. A polyhedron $P \subseteq \mathrm{R}^n$ is of *full-dimension* if $\dim(P)=n$

Valid Inequality. The inequality $\pi z \leq \pi_0$ is called a *valid inequality* for a set P if it is satisfied by all points in P.

Face and proper face. If $\pi z \leq \pi_0$ is a valid inequality for a polyhedron P, and $F = \{z \in P : \pi z = \pi_0\}$, F is called a *face* of P and we say that $\pi z \leq \pi_0$ *represents* F. A face F is said to be *proper* if $F \neq \emptyset$ and $F \neq P$.

Facet. A face F is a *facet* of a polyhedron P if $\dim(F) = \dim(P) - 1$.

1.3.3 Set Theory

Binary set. We call *binary set* the set $\{0, 1\}$ and denote it by $\mathbf{B}$. $\mathbf{B}^n$ denotes the set resulting of n Cartesian products $\mathbf{B} \times \mathbf{B} \times \cdots \times \mathbf{B}$. Given a set A, we denote by $\mathbf{B}^A$ the set of Cartesian products

$\mathbf{B}^{|A|}$.

Incidence or characteristic vectors. Subsets of a set can be represented by *incidence* or *characteristic* vectors. Thus if $N = \{1, 2, \ldots, h\}$, a subset N' is described by the vector $(n_1, n_2, \ldots, n_h)$, where $n_i = 1$ if $i \in N'$ and $n_i = 0$ otherwise. The subset A' of a set of arcs A of a digraph D can be represented by an incidence vector $z^{A'} = (z_{ij}) \in \mathbf{B}^{n(n-1)}$ whose coordinates are indexed by all possible ordered pairs (i, j), $i \neq j$ of vertices of D. If the arc (i, j) is present in the subset, then $z_{ij} = 1$ otherwise $z_{ij} = 0$. The expresion $z(A')$ denotes the sum of the variables corresponding to the elements of A'.

Partition of a set. A *partition of a set* N is a set of subsets $N_1, \ldots, N_p$ such that $N_i \bigcap N_j = \emptyset$ if $i \neq j$ and $N = \bigcup_{i=1}^{p} N_i$.

1.4 Integer Programming and Branch-and-Cut

In this section we briefly describe the basic ideas in which the branch-and-cut algorithm is based and the context in which it is applied. The branch-and-cut algorithm is based in linear programming and integer programming paradigms for combinatorial optimization problems.

1.4.1 Linear and Mixed Integer Programming

Linear programming is concerned with finding amongst all solutions satisfying a system of linear inequalities, one that minimizes a given linear expression on the same variables. More formally a linear program can be stated as:

$$\min\{c^T x : Ax \leq b,\ x \in \mathrm{R}^n_+\},$$

where $A \in \mathrm{R}^{m \times n}$, $b \in \mathrm{R}^m$, and $c \in \mathrm{R}^n$. Linear programming has been studied extensively since 1947, year in which George B. Dantzig developed this theory and proposed the *simplex algorithm* [12].

In a mixed linear integer programming problem the purpose is the same, except in that some of the variables must take integer values. These problems are notoriously hard to solve, both in theory and in practice. Mixed integer programming problems can be written generically as:

$$\min\{c^T x + d^T y : Ax + By \leq b,\ x \in \mathrm{R}^n_+, y \in \mathrm{Z}^p_+\},$$

where $A \in \mathrm{R}^{m \times n}$, $B \in \mathrm{R}^{m \times p}$, $b \in \mathrm{R}^m$, $c \in \mathrm{R}^n$, $d \in \mathrm{R}^p$.

1.4.2 Problem Relaxation

In certain circumstances, working with a less restrictive or more general problem will furnish useful information in order to solve the original problem. In our case, we will solve a "relaxed" version of MIP. The relaxed problem will have al least one of the following properties:

1. An enlarged set of feasible solutions.
2. A new objective function whose optimal value is at least as good as the original one.

One possible application of the first of these ideas leads to the concept of *linear programming relaxation* (LP relaxation). Thus, for the integer program

$$\min\{c^T x + d^T y : Ax + By \leq b \; x \in \mathrm{R}^n_+, y \in \mathrm{Z}^p_+\},$$

the LP relaxation is the linear program given by

$$\min\{c^T x + d^T y : Ax + By \leq b \; x \in \mathrm{R}^n_+, y \in \mathrm{R}^p_+\}.$$

Later we shall see how the LP relaxations are used to solve MIPs via brach-and-bound algorithms.

1.4.3 Valid Inequalities and Cuts

A *valid inequality* for a MIP is one that is satisfied by all its feasible solutions. Valid inequalities that are not part of the problem formulation and that are not satisfied by all feasible points of the LP relaxation are called *cuts*.

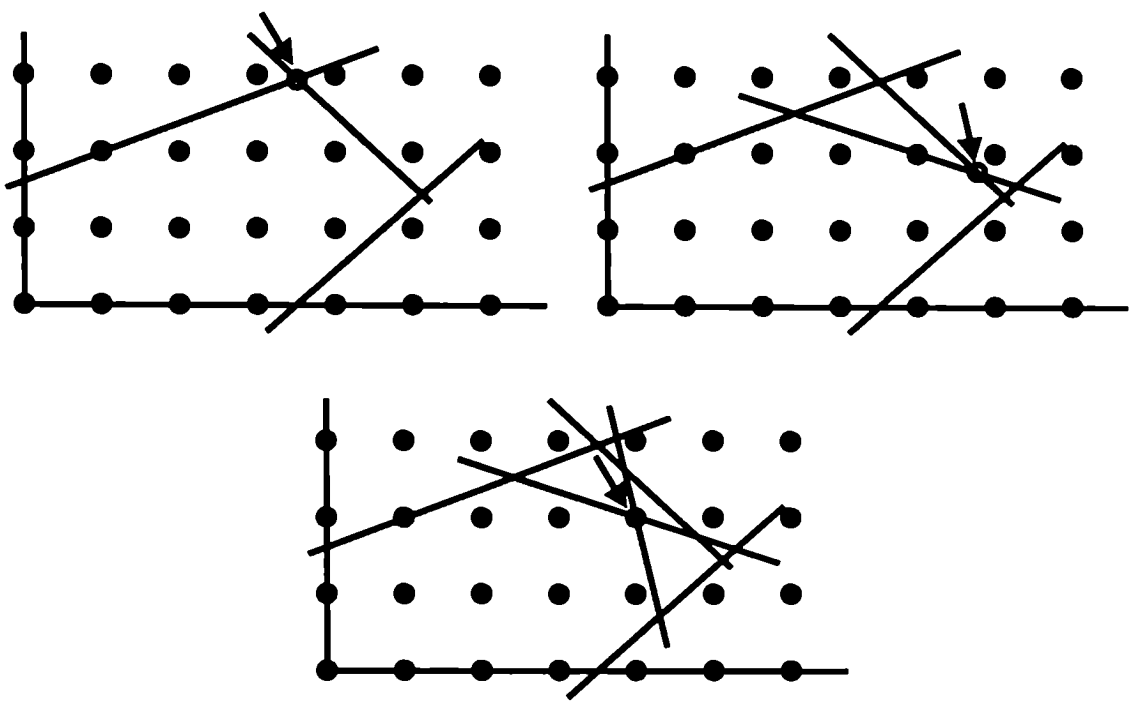

Figure 1.1: Feasible region and violated cuts

A cut that is not satisfied by the given optimal solution of the LP relaxation is a *violated cut*. Violated cuts can be added to the LP relaxation to tighten it. By doing so, we modify the current formulation in such a way that the LP feasible region becomes smaller but the MIP feasible region does not change (see Figure 1.1). Then we can resolve the LP and repeat the process, if necessary, so long as we can find violated cuts. To find violated cuts automatically one has to design and implement a *separation algorithm*. The framework of a simple cutting plane algorithm is given in Figure 1.2.

In case separation algorithms do not find violated inequalities, the obtained solution can still be used, because the final objective function value gives a lower bound to the optimal value. This value

Scheme of the Cutting Plane Algorithm

1. **Initialization:** Initialize the LP relaxation of the MIP formulation.
2. **LP solving/feasible check:** Solve the LP relaxation and let x^* denote its optimal solution. If x^* is integral, stop: an optimal solution of the MIP was found.
3. **Separation:** Run separation algorithms to find violated cuts by x^*. If violated cuts are found, add them to the LP and go to step 2, otherwise stop: the problem cannot be solved with these families of cutting planes.

Figure 1.2: Cutting Plane Algorithm

can be used to assess the quality of a known feasible solution (i.e., an upper bound). In our case, these solutions are provided by the tabu search algorithm due to Porto and Ribeiro [40].

1.4.4 Branch-and-Bound

A *branch-and-bound* or *enumerative search* algorithm refers to a divide-and-conquer method designed for solving integer programs by breaking up the set of feasible solutions into smaller subsets (branching), computing bounds for the objective function on these subsets, and using these bounds (bounding) to discard certain subsets from further consideration. Land and Doig [29] where the first to apply linear programming within this framework. They used the LP relaxation as the main tool for obtaining the bounds.

In the MIP case, branching can be performed by dividing the domain in which a given integer variables can take values into smaller subdomains. The original problem is splitted into smaller subproblems. In $0-1$ programs the simplest branching rule consists of defining two new subproblems. In the first, a fixed variable is required to take the value zero, while in the other it has to take the value one in the solution.

As mentioned earlier, in the MIP case, bounding is done by solving LP relaxations. The LP relaxation gives a *local lower bound* (i.e., it is only valid for the subproblem). If it is also valid for the original problem, then it is a *global lower bound.* If the solution found for the relaxation is feasible for the original problem and its objective function value is smaller than that of any feasible solution found so far, it is memorized and the *global upper bound* for the objective function is updated. At every iteration the algorithm selects a subproblem from the branch-and-bound tree, solves the LP relaxation, and computes a local lower bound for this subproblem.

If the new lower bound is not better than the global upper bound, the current subproblem is pruned from the branch-and-bound tree (*fathomed*). Otherwise, we check if the optimal solution of the subproblem relaxation is a feasible solution of the original problem. In this case, the subproblem is solved and fathomed.

These two properties allow avoiding the enumeration of portions of the branch-and-bound tree, with improvements in the algorithm performance. The tighter these bounds are, the bigger the gain is. This clearly shows the importance of having a good LP formulation for the problem.

When the list of subproblems becomes empty, then the best feasible solution (whose objective

function value is equal to the global upper bound) can be output as the optimal solution.

Branch-and-bound used to be the most usual way to solve MIP programs up to the early eighties. An important step towards the improvement of branch-and-bound came when cutting planes were introduced at the root subproblem of the branch-and-bound tree, to improve the initial formulation given by the LP relaxation. This idea was present for the first time in the mid sixties in the work of Balas [1] about implicit enumeration. This approach lead to the first success in solving some extremely difficult combinatorial optimization problems by Crowder, Johnson and Padberg [11]. They used strong cutting planes arising from knapsack polyhedra to strengthen the LP relaxation of the problem and then applied branch-and-bound.

1.4.5 Branch-and-Cut

The idea of using cutting planes in other subproblem than the root was first introduced by Miliotis [36] for the traveling salesman problem in the mid seventies. The use of facet defining cutting planes and automatic cutting plane generation in combination with branch-and-bound was first published in the mid eighties by Grötschel, Jünger and Reinelt [17] for the linear ordering problem. The term *branch-and-cut* was introduced by Padberg and Rinaldi in [38].

Scheme of the branch-and-cut algorithm

1. **Initialization:** Initialize the branch-and-cut tree (a list of subproblems, with the LP relaxation of the original formulation) and the global upper bound.
2. **Node selection:** If the list of open subproblems of the branch-and-cut tree is not empty choose one of them, otherwise stop.
3. **Lower bounding:** Solve the LP relaxation associated to the subproblem chosen in step 2. If the problem is infeasible go to step 2, otherwise let x^* denote its optimal solution.
   (a) If x^* is a feasible solution of the integer program update the global upper bound if necessary, remove the subproblem from the list, and go back to step 2.
   (b) If the value of the objective function for x^* is greater than the global upper bound, then remove the subproblem from the list, and go back to step 2.
4. **Branching/cutting decision:** If cutting planes should be generated, go to step 5; otherwise go to step 6.
5. **Cut generation:** Run separation algorithms to find cuts violated by x^*. If violated cuts are found add them to the LP and go to step 3.
6. **Branching:** Choose a branching variable and generate new subproblems by fixing the value of this variable. Add these subproblems to the list of open subproblems and go back to step 2.

Figure 1.3: Branch-and-Cut Algorithm

Figure 1.3 describes the basic framework of a branch-and-cut algorithm to solve a minimization problem.

1.5 Outline

The remaining chapters of this thesis are organized as follows.

In Chapter 2, we introduce the problem of *scheduling unrelated processors under precedence constraints*. Two previous mixed integer programming formulations are presented. They are compared with our new integer programming formulation.

In Chapter 3, we study the facial structure of the PLO polytope. It defines all partitions of the set of tasks such that each set of the partition is a linear ordering. Each set of the partition is therefore associated to a processor and the tasks of this set are to be processed in that processor in the corresponding linear ordering. The partition in linear orderings polytope is closely related to the clique partitioning polytope studied in [20, 21] and some order polytopes such as linear ordering [17] and partial ordering [13]. We compare this polyope with other well known order polytopes and we give a characterization of two versions of the PLO polytope. Finally, we define some valid inequalities and we prove that they are facets of the PLO polytope.

In Chapter 4, we describe several families of problem specific cutting planes and its separation algorithms. These cutting planes are used in a branch-and-cut algorithm. We discuss the implementation of this algorithm and the options that arise. Test are performed on a set of 108 instances derived from the work of Kitajima [28]. These instances represent different problems and algorithmic techniques that can be parallelized in a multiprocessor environment. The branch-and-cut algorithm is able to solve 43.52% of the cases to optimality and 27.78% of the cases with a gap smaller than 10%. Only 9.26% of the test cases remain with gap values bigger than 20%.

Finally, Chapter 5 contains our conclusions and suggestions for future research directions.

Chapter 2

Scheduling Unrelated Processors Under Precedence Constraints

We begin this chapter in Section 2.1 introducing the problem of *scheduling unrelated processors under precedence constraints*. Two previous mixed integer programming formulations are presented in Section 2.2. They are compared with our new integer programming formulation, which is introduced in Section 2.3.

2.1 Scheduling Unrelated Processors under Precedence Constraints

Scheduling problems arise from situations that require the assignment of resources over time to perform a set of activities, jobs or tasks. When activities and resources are deterministic, these problems may be viewed as combinatorial optimization problems.

Most of scheduling problems lead to challenging mathematical problems. Many of these scheduling problems are NP-hard [15].

In the case of the problem studied in this work, we consider a set of tasks related by precedence constraints according to a directed acyclic graph which have to be scheduled into a set of $m > 1$ unrelated machines. No preemption (i.e. no interruptions) are allowed once a task begins to be processed. We focus on the minimization of the makespan, which measures the earliest time to complete all tasks.

The problem of scheduling unrelated processors under precedence constraints was first considered by Porto and Menascé in [34, 39]. Fast greedy algorithms for processor assignment of parallel applications on heterogeneous multiprocessor architectures were proposed and compared. Porto and Ribeiro [40] then applied the tabu search metaheuristic to the same problem. The results obtained by tabu search considerably improved by aproximately 25% the makespan of the parallel applications, with respect to schedules generated by the best greedy algorithm. Later, Porto and Ribeiro [41] presented different parallel implementations for this tabu search algorithm, using synchronous message-passing strategies based on domain decompositions.

The minimization of the makespan on two uniform processors, i.e., the problem $Q2||C_{\max}$ according to the notation introduced in [16], is NP-hard [14, 15].

An application of this problem arises in the context of scheduling tasks of parallel programs on a multiprocessor computer or a distributed system. Parallel programs can be represented as a set of interrelated tasks which are sequential units. In a heterogeneous multiprocessor system, not only we have to determine how many, but also which processors should be allocated to an application, as well as which processors are going to be assigned to each task.

2.2 Existing Formulations

We review below two previous integer linear formulations for the problem we are interested in. In the standard notation of [16], this scheduling problem is denoted by $R|pred|C_{\max}$.

Let $T = \{t_0, \ldots, t_{n-1}\}$ be the set of partially ordered tasks and $P = \{p_0, \ldots, p_{m-1}\}$ the set of processors. We work with two set of indices $N = \{0, \ldots, n-1\}$ and $M = \{0, \ldots, m-1\}$ associated respectively with these sets. We are also given an acyclic directed precedence graph $D = (N, A)$ associated with the set of tasks, such that $(t_i, t_j) \in A$ if and only if task t_i must be executed before task t_j. Each task has to be assigned to exactly one processor, in which it is entirely executed without preemption.

2.2.1 Formulation with a Non-polynomial Number of 0-1 Variables

The first formulation of this problem as an integer linear program was given by Blazewicz et al. [4] in the context of deterministic models for scheduling under resource constraints.

Let H denote a known heuristics schedule length, called the time horizon, and assume that H and all processing times d_{jk} are integer values. Notice that a trivial upper bound for H is given by:

$$\sum_{j=0}^{n-1} \max_{k=0,\ldots,m-1} \{d_{jk}\}.$$

If the time horizon is splitted into H unit time-periods, we may define the following $0-1$ variable:

$$x_{jkt} = \begin{cases} 1 & \text{if task } t_j \text{ scheduled to processor } p_k \text{ in the } t\text{-th time period,} \\ 0 & \text{otherwise,} \end{cases}$$

for all $j \in N, k \in M$, and $t = 1, \ldots, H$. Without loss of generality, task t_{n-1} is assumed to be a unique terminal task with no successors and all other tasks should be processed before the processing of task t_{n-1} begins. Associated with each task t_j are its early finish time a_j, which corresponds to the length of the critical path associated to this task in the precedence graph, and the latest finish time ℓ_j, which may be computed a priori by forward and reverse traversals of the precedence graph. Obviously, 0 and H are valid lower and upper bounds for a_j and ℓ_j, respectively, for all $t_j \in T$.

We denote by $\Gamma(j)$ the set of indices of the immediate predecessors of task t_j, i.e. $\Gamma(j) = \{i \in N, (t_i, t_j) \in A\}$. This scheduling problem under precedence constraints may be formulated in the following way:

$$\text{minimize } C_{\max} = \sum_{k=0}^{m-1} \sum_{t=a_n}^{\ell_n} t\, x_{nkt}, \tag{2.1}$$

subject to:

$$\sum_{k=0}^{m-1} \sum_{t=a_j}^{\ell_j} x_{jkt} = 1, \qquad \forall j \in N, \tag{2.2}$$

$$\sum_{k=0}^{m-1} \sum_{t=a_j}^{\ell_j} t\, x_{jkt} - \sum_{k=0}^{m-1} \sum_{t=a_f}^{\ell_f} (t + d_{fk})\, x_{fkt} \geq 0, \qquad \forall j \in N, \forall f \in \Gamma(j), \tag{2.3}$$

$$\sum_{j=0}^{n-1} \sum_{q=t}^{t-d_{jk}-1} x_{jkq} \leq 1, \qquad t = 1, \ldots, H, \forall k \in M, \tag{2.4}$$

$$x_{jkt} \in \{0, 1\}, \qquad \forall j \in N,\ k \in M,\ t = 1, \ldots, H. \tag{2.5}$$

Equations (2.2) ensure that each task is completed and assigned to exactly one processor, inequalities (2.3) express precedence constraints (no task may be started unless all its predecessors have already completed their execution), and inequalities (2.4) state that each machine can be used by at most one task in each time-period.

Regardless of more practical issues concerning its solvability as a large integer programming problem, this formulation already has two main drawbacks. First it requires that the processing times be integer valued. Second, the number of $0-1$ variables is not polynomially bounded by the number of machines and tasks, due to the splitting of the schedule horizon into unit time periods.

2.2.2 A Formulation with Polynomial Number of $0-1$ Variables

Maculan et al. [32] give a more natural formulation with a polynomial number of variables for the problem of task scheduling in unrelated processors under precedence constraints. They define a set of $0-1$ variables:

$$y_{jk}^{s} = \begin{cases} 1 & \text{if task } t_j \text{ scheduled to processor } p_k \text{ is the } s\text{-th task executed in } p_k, \\ 0 & \text{otherwise,} \end{cases}$$

for all $j \in N$, $k \in M$, and $s = 1, \ldots, n$. For every task $t_j \in T$, we denote by $e_j \geq 0$ the starting time of its execution. For each task $t_j \in T$ and each processor $p_k \in P$, we denote by d_{jk} the total processing time of task t_j in case it is assigned to processor p_k. Then, the scheduling problem under precedence constraints may be formulated as follows:

minimize $C_{\max}$,

subject to:

$$\sum_{k=0}^{m-1}\sum_{s=1}^{n} y_{jk}^{s} = 1, \qquad \forall j \in N, \tag{2.6}$$

$$\sum_{j=0}^{n-1} y_{jk}^{1} \leq 1, \qquad \forall k \in M, \tag{2.7}$$

$$\sum_{j=0}^{n-1} y_{jk}^{s} - \sum_{j=0}^{n-1} y_{jk}^{s-1} \leq 0, \qquad \forall k \in M,\ s = 2, \ldots, n, \tag{2.8}$$

$$e_i - e_j + \sum_{k=0}^{m-1}\sum_{s=1}^{n} d_{ik} y_{ik}^{s} \leq 0, \qquad \forall i \in \Gamma(j),\ \forall j \in N, \tag{2.9}$$

$$e_i - e_j + d_{ik} - \mu[2 - (\sum_{r=1}^{s} y_{ik}^{r} + \sum_{r=s+1}^{n} y_{jk}^{r})] \leq 0, \qquad \forall (i,j) \in N \times N,\ \forall k \in M,\ s = 1, \ldots, n-1, \tag{2.10}$$

$$e_j - C_{\max} + \sum_{k=0}^{m-1}\sum_{s=1}^{n} d_{jk} y_{jk}^{s} \leq 0, \qquad \forall j \in N, \tag{2.11}$$

$$-e_j \leq 0, \qquad \forall j \in N, \tag{2.12}$$

$$y_{jk}^{s} \in \{0,1\}, \qquad \forall j \in N,\ \forall k \in M,\ s = 1, \ldots, n.$$

Equations (2.6) ensure that each task is completed and assigned to exactly one processor. Inequalities (2.7) and (2.8) express that a processor cannot process a task in position s until it has processed a task in position $s-1$. Inequalities (2.9) express precedence constraints: no task may be started unless all its predecessors have already completed their execution. Inequalities (2.10) define the sequence of starting times of the tasks assigned to the same processor, ensuring that no overlap will occur, i.e., each machine can be used by at most one task at a time. The constant μ s such that, if tasks t_i and t_j are not executed in processor p_k in that order then inequality (2.10) is allways satisfied. Finally, inequalities (2.11) define the makespan.

Though this formulation is much lighter than the precedent one, since it has less variables, constraints and nonzero entries, preliminary computational experiments have shown that it is not suitable to solve even very small instances to optimality.

2.3 A New Formulation

In this section, we present a new formulation for the problem of scheduling unrelated processors under precedence constraints. This formulation reveals as part of it a polytope we called *partition in linear orderings*. We expect that the study of the facial structure of this polytope will have a major impact to formulate and solve other multiple machines scheduling problems. Chapter 3 is devoted to the study of this polytope. Also, this formulation allows solving bigger problems than the previous ones. These results are developed in Chapter 4.

Given the directed acyclic precedence graph $D = (T, A)$, we define the following sets for every task $t_j \in T$:

$P_j = \{t_i \in T:$ there exists a path in D from t_i to $t_j\}$, i.e., P_j is the set of predecessors of t_j;

$Q_j = \{t_i \in T:$ there exists a path in D from t_j to $t_i\}$, i.e., Q_j is the set of sucessors of t_j; and

$R_j = \{t_i \in T:$ there is no path in D either from t_i to t_j or from t_j to $t_i\}$.

These sets will play an important role in the generation of new valid inequalities, as well as in the reduction on the number of $0-1$ variables.

This new formulation makes use of two types of $0-1$ variables:

$$y_{jk} = \begin{cases} 1, & \text{if task } t_j \text{ is scheduled to processor } p_k, \\ 0, & \text{otherwise,} \end{cases}$$

for all $j \in N = \{0, \ldots, n-1\}, k \in M = \{0, \ldots, m-1\}$, and

$$z_{ij} = \begin{cases} 1, & \text{if task } t_i \text{ is scheduled before task } t_j \text{ and to the same processor,} \\ 0, & \text{otherwise,} \end{cases}$$

for all $j \in N, i \in R_j \cup P_j$. Moreover, for every task $t_j \in T$, we denote by $e_j \geq 0$ the starting time of its execution. The problem of scheduling unrelated processors under precedence constraints may be formulated as follows:

minimize $C_{\max}$

subject to:

$$\sum_{k=0}^{m-1} y_{jk} = 1 \qquad \forall j \in N, \tag{2.13}$$

$$z_{ij} + z_{ji} + y_{ik} - y_{jk} \leq 1 \qquad \forall j \in N, \forall i \in R_j, \forall k \in M, \tag{2.14}$$

$$z_{ij} + y_{ik} - y_{jk} \leq 1 \qquad \forall j \in N, \forall i \in P_j, \forall k \in M, \tag{2.15}$$

$$y_{ik} + y_{jk} - z_{ij} - z_{ji} \leq 1 \qquad \forall j \in N, \forall i \in R_j, \forall k \in M, \tag{2.16}$$

$$y_{ik} + y_{jk} - z_{ij} \leq 1 \qquad \forall j \in N, \forall i \in P_j, \forall k \in M, \tag{2.17}$$

$$e_i - e_j + \sum_{k=0}^{m-1} d_{ik} \cdot y_{ik} \leq 0 \qquad \forall j \in N, \forall i \in \Gamma(j), \tag{2.18}$$

$$e_j - C_{\max} + \sum_{k=0}^{m-1} d_{jk} \cdot y_{jk} \leq 0 \qquad \forall j \in N, \tag{2.19}$$

$$e_i - e_j + \sum_{k=0}^{m-1} d_{ik} \cdot y_{ik} \leq \mu_{ij} \cdot (1 - z_{ij}) \qquad \forall j \in N, \forall i \in R_j, \tag{2.20}$$

$$e_j \geq 0 \qquad \forall j \in N, \tag{2.21}$$

$$y_{jk} \in \{0,1\} \qquad \forall j \in N, \forall k \in M, \text{ and}$$

$$z_{ij} \in \{0,1\} \qquad \forall j \in N, \forall i \in R_j \cup P_j.$$

Equations (2.13) ensure that each task is processed and assigned to exactly one processor. Inequalities (2.18) express the precedence constraints: no task may be started unless all its predecessors have already completed their execution. Inequalities (2.20) define the sequence of starting times of the tasks

assigned to the same processor, ensuring that no overlap occurs. The constant μ_{ij} is such that if t_i and t_j are not executed in the same processor in that order, then inequality (2.20) is always satisfied. Though knowing the smaller possible value of this constant is as difficult as solving the original problem itself, some good approximations can be obtained. One good estimation is $\widehat{\mu_{ij}} = \overline{f_i} - \underline{e_j}$ ($\widehat{\mu_{ij}} \leq \mu_{ij}$), where $\overline{f_i}$ is an overestimate of the latest time task t_i could finish to be processed, and $\underline{e_j}$ is an underestimate of the least time at which t_j could be processed. The tighter these estimations are, the tighter this inequality will be. In Section 4.1.7 we discuss how these estimators are calculated. Inequalities (2.19) define the makespan.

The correct relation between the z and y variables are assured by inequalities (2.14)–(2.17). Since the formulation with the variables defined earlier seem to be a novelty in the scheduling literature, we now discuss these inequalities in detail. First we consider the inequalities (2.14) and (2.16) for fixed tasks t_i and t_j and a machine k. In this case we have that $i \in R_j$ and $j \in R_i$. Therefore, inequality (2.14) can also be written for when the roles of i and j are interchanged. Table 2.3 summarizes the outcomes of inequalities (2.16) and (2.14 for the four possible combinations of the values of y_{ik} and y_{jk}. In this table, when inequality (2.14) written for $j \in R_i$, it is denoted by (2.14)'.

y_{ik}	y_{jk}	(2.16)	(2.14)	(2.14)'
0	0	$z_{ij} + z_{ji} \geq -1$	$z_{ij} + z_{ji} \leq 1$	$z_{ij} + z_{ji} \leq 1$
0	1	$z_{ij} + z_{ji} \geq 0$	$z_{ij} + z_{ji} \leq 2$	$z_{ij} + z_{ji} \leq 0$
1	0	$z_{ij} + z_{ji} \geq 0$	$z_{ij} + z_{ji} \leq 0$	$z_{ij} + z_{ji} \leq 2$
1	1	$z_{ij} + z_{ji} \geq 1$	$z_{ij} + z_{ji} \leq 1$	$z_{ij} + z_{ji} \leq 1$

Table 2.1: Relation between z and y variables when $i \in R_j$.

In the first row of Table 2.3, we cannot determine the exact values of z_{ij} and z_{ji}. However, due to constraints (2.13), there must be a machine $\ell \neq k$ for which at least one of the variables $y_{i\ell}$ or $y_{j\ell}$ is 1. Thus, for machine ℓ, one of the cases in the three remaining rows of Table 2.3) must hold which forces the z variables to assume the right values. In the third (fourth) row of Table 2.3, the condition on the fifth (fourth) column forces both z_{ij} and z_{ji} to be 0, which is correct since, in both cases, i and j are not assigned to the same machine. Finally, in the last row of the table, the three conditions forces either z_{ij} or z_{ji} to be set to 1 which is correct since both tasks are assigned to machine k and one must precede the other.

Now we investigate the relation between y and z variables when $i \in P_j$. For fixed values of i, j and k, we are left only with the 2 inequalities (2.15) and (2.17) since j is clearly not in P_i. As for the previous case, we build Table 2.3 in which the outcomes of these inequalities are given for all the possible values of y_{ik} and y_{jk}.

y_{ik}	y_{jk}	(2.17)	(2.15)
0	0	$z_{ij} \geq -1$	$z_{ij} \leq 1$
0	1	$z_{ij} \geq 0$	$z_{ij} \leq 2$
1	0	$z_{ij} \geq 0$	$z_{ij} \leq 0$
1	1	$z_{ij} \geq 1$	$z_{ij} \leq 1$

Table 2.2: Relation between z and y variables when $i \in P_j$.

As for the previous table, the conditions in row 1 of Table 2.3 are inconclusive with respect to

the value of z_{ij}. Once again, this is not a difficulty since, like when $i \in R_j$, there must exist another machine for which we are in one of the cases expressed in the remaining rows of the table. Now, notice that row 2 is also inconclusive. However, from (2.13), there exists another machine $\ell \neq k$ for which the third row (with k replaced by ℓ) must hold. The last column of the third row forces z_{ij} to be equal 1. This is in accordance with the definition of the z variables since i and j are not in the same machine. Finally, the last row of Table 2.3 imposes that $z_{ij} = 1$ which is correct since, in this situation, i precedes j in machine k.

Inequalities (2.14) to (2.17) determine the relation between variables z and y. Inequalities (2.14) and (2.15) ensure that if task t_i is processed in p_k but task t_j is not, then both z_{ij} and z_{ji} are equal to zero. In case both t_i and t_j are processed in the same processor, these inequalities become redundant. Inequalities (2.16) and (2.17) ensure that if t_i and t_j are both processed in machine p_k, then z_{ij} or z_{ji} is forced to be one. In case t_i is processed in p_k and t_j is not, then the inequalities become redundant.

Equations (2.13) and inequalities (2.14) to (2.17) deserve a more exhaustive analysis, because they represent an underlying common structure to different multiple processor scheduling problems. This study is performed in the next chapter.

Chapter 3

The Partition in Linear Orderings Polytope

In this chapter we consider the problem of partitioning a directed graph in linear orderings (PLO). Formally the problem can be stated as follows: given a directed graph $D = (N, A)$ with weights $w_a \in \mathrm{R}$ for all arc $a \in A$, a feasible solution is defined by any set of subgraphs partitioning the vertices of D each of them defining a linear ordering on its vertices. An optimal solution is one in which the sum of the weights of the arcs in the subgraphs is minimized.

In the previous chapter we have presented a new formulation for the problem of scheduling tasks with precedence constraints in unrelated processors. A subset of the inequalities in this model defines a linear ordering for the set of tasks that is assigned to each processor. Thus, to tighten this model we investigate the facial structure of the *polytope of partitions in linear orderings.*

It should be noticed that most of the results in this chapter hold for the particular case where the graph is complete. Though the precedence graph of the scheduling problem treated here are not necessarily complete, this assumption makes the proofs much easier.

The *PLO* polytope is closely related to the clique partitioning polytope studied in [20, 21] and some ordering polytopes such as linear ordering [17] and partial ordering [13]. In section 3.1 we compare the PLO polytope with some other well studied ordering polytopes. In section 3.2 we present two alternative characterizations of the *PLO* polytope. Finally, in section 3.3 introduce some valid and facet defining inequalities for the polytope.

3.1 Ordering Polytopes

Several ordering polytopes have been extensively studied in the recent years. In this section we study of the relations between *PLO* polytope and some of the most well known ordering polytopes. We will consider different ordering polytopes ranging from the most inclusive (i.e. *acyclic subgraph*), to the most restrictive (i.e. *linear ordering*). To this, we start by giving some basic definitions which allow us to describe the order relations.

Given a binary relation B over N ($B \subseteq N \times N$), the following properties are defined:

- **Symmetry:** $(i, j) \in B$ then $(j, i) \in B \; \forall \; i, j \in N$.

- **Antisymmetry:** $(i,j) \in B$ and $(j,i) \in B$ then $i = j$, $\forall\ i,j \in N$.
- **Transitivity:** $(h,i) \in B$ and $(i,j) \in B$ then $(h,j) \in B$, $\forall\ h,i,j \in N$.
- **Comparability:** $(i,j) \in B$ or $(j,i) \in B$, $\forall\ i,j \in N$.
- **Weak comparability:** $(h,i) \in B$ and $(h,j) \in B$, or, $(i,h) \in B$ and $(j,h) \in B$, then $(i,j) \in B$ or $(j,i) \in B\ \forall\ h,i,j \in N$.

3.1.1 Preordering

A preordering is a binary relation satisfying the transitivity property. The *preordering polytope* can be described as the convex hull of the characteristic vectors $z^{A'}$ of $A' \subseteq A$ of a given digraph $D = (N, A)$, such that the arcs of A' determines a preordering in the vertices of D. The set of preorderings of D is given by:

$$\mathcal{P}(D) = \{A' \subseteq A : A' \text{ is a preordering in } D\}.$$

Thus, the preordering polytope $P_{Pre}(D)$ for D is defined as:

$$P_{Pre}(D) = conv(\{z^{A'} : z^{A'} \in \mathbf{B}^A,\ A' \in \mathcal{P}(D)\})$$

This polytope has been studied in [13].

3.1.2 Complete Preordering

A complete preordering is a binary relation satisfying the transitivity and comparability properties. The *complete preordering polytope* can be described as the convex hull of the characteristic vectors $z^{A'}$ of $A' \subseteq A$ of a given digraph $D = (N, A)$, such that the arcs of A' determines a complete preordering in the vertices of D. The set of complete preorderings of D is given by:

$$\mathcal{CP}(D) = \{A' \subseteq A : A' \text{ is a complete preordering in } D\}.$$

The complete preordering polytope $P_{PO}(D)$ for D is defined as:

$$P_{CP}(D) = conv(\{z^{A'} : z^{A'} \in \mathbf{B}^A,\ A' \in \mathcal{CP}(D)\})$$

This polytope has been studied in [13].

3.1.3 Acyclic Subgraph

The *acyclic subgraph polytope* can be described as the convex hull of the set of characteristic vectors $z^{A'}$ of $A' \subseteq A$ of a given digraph $D = (N, A)$, such that the subgraph induced by A' in D, denoted by $D(A')$, is acyclic. More precisely, given a digraph $D = (N, A)$, let

$$\mathcal{A}(D) = \{A' \subseteq A : D(A') \text{ is acyclic}\}$$

i.e., $\mathcal{A}(D)$ is the family of all subsets of arcs in D whose induced subgraph is acyclic. In other words, $\mathcal{A}(D)$ denotes the set of feasible solutions of the acyclic subgraph problem on D.

Now, the acyclic subgraph polytope $P_{AC}(D)$ for D can be defined as follows:

$$P_{AC}(D) = conv(\{z^{A'} \in \mathbf{B}^A : A' \in \mathcal{A}(D)\})$$

This polytope has been extensively studied in [17, 18, 9].

3.1.4 Partial Ordering

A partial ordering is a binary relation satisfying the antisymmetry and transitivity properties. The *partial ordering polytope* can be described as the convex hull of the characteristic vectors $z^{A'}$ of $A' \subseteq A$ of a given digraph $D = (N, A)$, such that the arcs of A' determines a partial ordering in the vertices of D. Thus, we can define

$$\mathcal{B}(D) = \{A' \subseteq A : A' \text{ is a partial ordering in } D\}$$

to be the set of partial orderings in D. The partial ordering polytope $P_{PO}(D)$ for D is defined as follows.

$$P_{PO}(D) = conv(\{z^{A'} \in \mathbf{B}^A : A' \in \mathcal{B}(D)\}).$$

This polytope has been studied in [13].

3.1.5 Interval Ordering

An interval ordering is a partial ordering $D = (N, A)$ such that there exists a real interval $[\ell_u, r_u)$ for each $u \in N$ with the property that $(u, v) \in A$ if and only if $r_u \leq \ell_v$. Such a set of intervals is called an *interval representation* of D. Figure 3.1 shows an interval ordering together with an interval representation. An interval ordering can be characterized by the absence of a forbidden substructure as in the following Theorem.

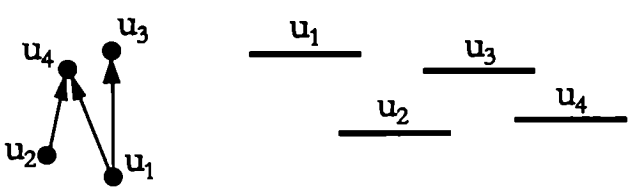

Figure 3.1: Interval order

Theorem 3.1 *[43] A partial ordering $D = (N, A)$ is an interval ordering if and only if D does not contains a digraph of the form $D = (\{v_1, v_2, v_3, v_4\}, \{(v_1, v_2), (v_3, v_4)\})$ as an induced subdigraph.*

We define the *interval ordering polytope* of a given digraph $D = (N, A)$ as the convex hull of the incidence vectors A' such that the subgraph $D' = (N, A')$ determines an interval order. Therefore, this polytope is given by:

$$P_{IO}(D) = conv(\{z^{A'} \in \mathbf{B}^A : D' = (N, A') \text{ is an interval order}\}).$$

The interval ordering polytope has been studied in [43].

3.1.6 Linear Ordering

A linear ordering is a binary relation which satisfies antisymmetry, transitivity and comparability. The *linear ordering polytope* can be described as the convex hull of the characteristic vectors $z^{A'}$ of $A' \subseteq A$ of a given digraph $D = (N, A)$, such that the arcs of A' determines a linear ordering in the vertices of D. Suppose $D = (N, A)$ is a digraph and let

$$\mathcal{L}(D) = \{A' : A' \subseteq A, A' \text{ is a linear ordering in } D\}$$

be the set of linear orderings in D. The linear ordering polytope $P_{LO}(G)$ for D is defined as:

$$P_{LO}(D) = conv(\{z^{A'} : z^{A'} \in \mathbf{B}^A,\ A' \in \mathcal{L}(D).\})$$

This polytope has been extensively studied in [19].

3.1.7 Partition in Linear Orderings

A partition in linear orderings is a binary relation which satisfies antisymmetry, transitivity and weak comparability. The *partition in linear orderings polytope* can be described as the convex hull of the characteristic vectors A' of A of a given digraph $D = (N, A)$, such that the arcs of A' determines a partition of the vertices of D into linear orderings. Suppose $D = (N, A)$ is a digraph and let

$$\mathcal{PL}(D) = \{A' : A' \subseteq A, A' \text{ is a partition of the vertices of } D \text{ into linear orderings}\}$$

be the set of partitions in linear orderings in D. The partition in linear orderings polytope $P_{PLO}(G)$ for D is defined as:

$$P_{PLO}(D) = conv(\{z^{A'} : z^{A'} \in \mathbf{B}^A,\ A' \in \mathcal{PL}(D)\}).$$

We have not found references to this polytope in the literature.

On the other hand, we can establish an useful correspondence between the objects describing an instance or a solution of the PLO problem and those describing an instance or a solution of the scheduling problem from the previous chapter. This correspondence is shown in Table 3.1 and will be used later in the text.

Scheduling problem	*PLO*
Tasks	Vertices
Sequence of tasks processed in a processor	A linear ordering subgraph
Precedences within the processor	Arcs in one linear ordering

Table 3.1: Correspondence between scheduling and PLO problems.

3.1.8 Comparison of Polytopes

In this section we prove several results that establish inclusion relations amongst the different ordering polytopes. Such relations are summarized in Figure 3.8.

Theorem 3.2 $P_{PO}(D) \subset P_{Pre}(D)$.

Proof. Every partial ordering is a binary relation satisfying the antisymmetry and transitivity properties. Therefore it is also a preordering. On the other hand, as depicted in Figure 3.2, the strict inclusion holds since there exist preorderings which are not partial orderings. □

Figure 3.2: Preordering but not partial ordering

Theorem 3.3 $P_{LO}(D) \subset P_{IO}(D) \subset P_{PO}(D) \subset P_{AC}(D)$.

Proof. Let us first prove that $P_{LO}(D) \subset P_{IO}(D)$. Every linear ordering can be represented as an interval ordering by just drawing the intervals $[\ell_u, r_u)$ in a sequence such that $\ell_u \leq r_{u+1}$ for $u = 1, \ldots, n-1$. Thus the inclusion holds and it is strict as can be seen from the example in Figure 3.1.

Now let us consider the case $P_{IO}(D) \subset P_{PO}(D)$. Since an interval ordering is a partial ordering by definition, the inclusion is obvious. Theorem 3.1 provides an immediate case showing that strict inclusion holds.

Finally, let us consider the case $P_{PO}(D) \subset P_{AC}(D)$. The inclusion is obvious since a partial ordering is acyclic. However, though the digraph $(\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3)\})$ is acyclic, it does not define a partial ordering on the vertices since transitivity does not hold. Therefore, the inclusion is strict. □

Theorem 3.4 $P_{PLO}(D) \subset P_{PO}(D)$.

Proof. By definition a partition in linear orderings is a partial ordering satisfying also the weak comparability property. The example in Figure 3.3 shows that the inclusion is strict since the largest component of this digraph is not a linear ordering. □

Theorem 3.5 $P_{LO}(D) \subset P_{PLO}(D)$.

Figure 3.3: Partial ordering but not PLO

Proof. By definition a linear ordering is a partial ordering that satisfies also the comparability property. Since every ordering satisfying comparability property also has the weak comparability property the result holds. See Figure 3.4 for an example showing that the inclusion is strict. □

Figure 3.4: *PLO* but not linear ordering

Theorem 3.6 $P_{IO}(D) \not\subset P_{PLO}(D)$ *nor* $P_{PLO}(D) \not\subset P_{IO}(D)$.

Proof. See Figure 3.5 for an interval ordering that is not a *PLO*.

Figure 3.5: Interval ordering but not PLO

See Figure 3.6 for an example of a *PLO* which is not an interval order.

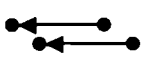

Figure 3.6: *PLO* but neither interval ordering nor complete preordering

□

Theorem 3.7 $P_{CP}(D) \not\subset P_{PLO}(D)$ *nor* $P_{PLO}(D) \not\subset P_{CP}(D)$.

Proof. See Figure 3.7 for a complete preordering that is not a *PLO*. See Figure 3.6 for an example of

Figure 3.7: Complete preordering but not PLO

a *PLO* which is not a complete preordering. □

Theorem 3.8 $P_{Pre}(D) \cap P_{AC}(D) = P_{PO}(D)$.

Proof. By definition a preordering is a binary relation satisfying the transitivity property, and acyclic graphs satisfies the antisymmetry property. So $P_{Pre}(D) \cap P_{AC}(D) \subset P_{PO}(D)$. Conversely by Theorem 3.3 $P_{AC}(D) \supset P_{PO}(D)$ and by definition $P_{Pre}(D) \supset P_{PO}(D)$ so $P_{Pre}(D) \cap P_{AC}(D) \supset P_{PO}(D)$. □

Theorem 3.9 $P_{CP}(D) \cap P_{PLO}(D) = P_{LO}(D)$.

Proof. By definition a complete preordering is a binary relation satisfying the transitivity and comparability properties, and the PLO is a binary relation which satisfies antisymmetry, transitivity and weak comparability properties. So $P_{CP}(D) \cap P_{PLO}(D) \subset P_{LO}(D)$. Conversely by Theorem 3.5 $P_{PLO}(D) \supset P_{LO}(D)$ and by definition $P_{CP}(D) \supset P_{LO}(D)$ so $P_{CP}(D) \cap P_{PLO}(D) \supset P_{LO}(D)$. □

Theorem 3.10 $P_{CP}(D) \cap P_{IO}(D) = P_{LO}(D)$.

Proof. By definition a complete preordering is a binary relation satisfying the transitivity and comparability properties, and the interval ordering implies antisymmetry property. So $P_{CP}(D) \cap P_{IO}(D) \subset P_{LO}(D)$. Conversely by Theorem 3.3 $P_{IO}(D) \supset P_{LO}(D)$ and by definition $P_{CP}(D) \supset P_{LO}(D)$ so $P_{CP}(D) \cap P_{IO}(D) \supset P_{LO}(D)$. □

Figure 3.8 summarizes the results given above. The importance of them for this dissertation can be measured in two ways. First they show that the P_{PLO} does not correspond to any of the previously studied polytopes. This reinforces the originality of our work. Moreover, as mentioned earlier, a good linear description of this polytope would be helpful in modeling a wide range of scheduling problems as integer programs. On the other hand, some of the relations above show that several of the valid inequalities from other well studied polytopes are inherited by the P_{PLO} and are good candidates to be facet defining or to be lifted.

3.2 Characterizations of the PLO Polytope

In this section we describe two integer linear programming formulations for the partition in linear orderings problem, and present some elementary facts about the associated polyhedra. We recall from Table 3.1 that there is an analogy between scheduling and PLO solutions. This analogy is used whenever it simplifies the presentation of the text.

Suppose that $P = (N, A)$ is the precedence graph given as an instance of the scheduling problem with $n = |N|$ tasks. Let G^* and G^T denote the transitive closure and the transpose of a graph G respectively. In order to the define the partition in linear ordering problem corresponding to the scheduling problem, we consider the graph D given by

$$D = (N, A) = D_n \setminus (P^*)^T,$$

where D_n is the complete digraph on n vertices. A partition in linear orderings is a collection of subgraphs forming a partition of the vertices (tasks) of N such that each subgraph is a linear ordering of its vertices (tasks). According to our analogy, the set of vertices induced by a subgraph of the collection corresponds to the set of tasks that are assigned to a processor of the scheduling problem. Moreover, these tasks are scheduled in the processor according to the linear ordering provided by this subgraph.

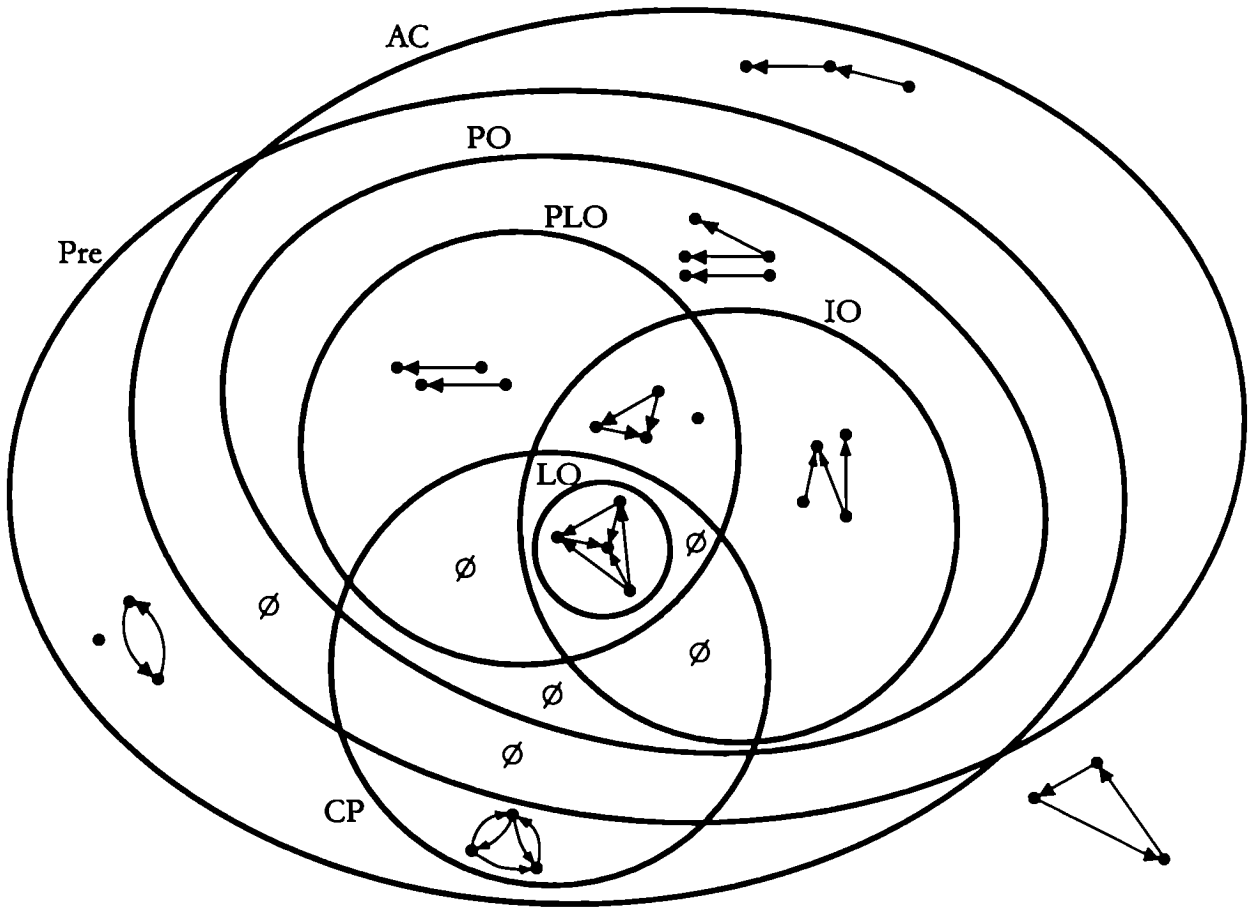

Figure 3.8: Ordering polytopes inclusion diagram

It should be noticed that according to the definition of the graph D, all the precedence constraints described by P are respected by any partition of D in linear orderings. However, in order to simplify our proofs, we will assume in many proofs throughout this text that the arc set of the precedence graph P is empty so that D is complete, i.e., $D = D_n$. In practice, assuming that D is complete is not too restrictive since arbitrary large weights can always be assigned to the forbidden arcs avoiding their presence in any optimal solution.

There is a natural way to associate a polytope with a given instance of the partition in linear orderings problem such that every vertex of the polytope corresponds to a feasible solution and vice versa. To this end a partition in linear orderings can be represented by an incidence vector $z \in \mathbb{B}^{n(n-1)}$ whose coordinates are indexed by all possible ordered pairs (i, j), $i \neq j$ of vertices of D. Thus, $z_{ij} = 1$ means that both nodes i and j belong to the same subgraph of the partition and that node j is preceded by node i in the linear ordering induced by this subgraph.

This representation yields to the definition of $P_{PLO}(D)$ as the convex hull of all incidence vectors z representing a partition in linear orderings of the vertices of D. So, if we let

$$L = \{z \in \mathbf{B}^{n(n-1)}, z \text{ incidence vector of a partition in linear orderings of } D\}$$

we can define,

$$P_{PLO}(D) \equiv conv(L).$$

We can go one step further and define a new set of variables $y = (y_{jk}) \in \mathbf{B}^{np}$. These variables relate the vertices of D with the subsets of the partition of N to which they are assigned. We have now a second representation of the partition in linear orderings polytope, where the incidence vectors of the partitions are defined on both sets of variables y and z.

This representation yields to the definition of $P^{yp}_{PLO}(D)$ as the convex hull of all incidence vectors (y, z) representing a partition of the vertices of D in at most p linear orderings. Now, let

$$L_y = \{(y, z) \in \mathbf{B}^{n(n+p-1)}, (y, z) \text{ incidence vector of a partition in linear orderings of } D\}$$

so that we can define,

$$P^{yp}_{PLO}(D) \equiv conv(L_y).$$

Theorem 3.11 *For a digraph D, a correct formulation for the partition of D into linear orderings is given by the following system:*

$$z_{hi} + z_{ij} - z_{hj} + z_{ih} + z_{ji} - z_{jh} \leq 1, \qquad (h,i,j) \in N^3,\ h \neq i \neq j \neq h, \tag{3.1}$$

$$z_{hi} + z_{ij} + z_{jh} - z_{ih} - z_{ji} - z_{hj} \leq 1, \qquad (h,i,j) \in N^3,\ h \neq i \neq j \neq h, \tag{3.2}$$

with $z \in \mathbf{B}^{n(n-1)}$ when $n \geq 3$.

We call inequalities (3.1) *double triangle inequalities*, *dT* for short. Inequalities (3.2) are called *double cycle inequalities*, *dC* for short. Support graphs of these inequalities are given in Figure 3.9. In this figure and for all the remaining of this chapter we adopt the following convention when drawing the support graph of valid inequalities: full lines represent arcs with positive coefficients and dashed lines represent those with negative coefficients. Moreover, lines with no arrows represent the couple of opposites arcs. Inequalities are always in the "$\leq$" form.

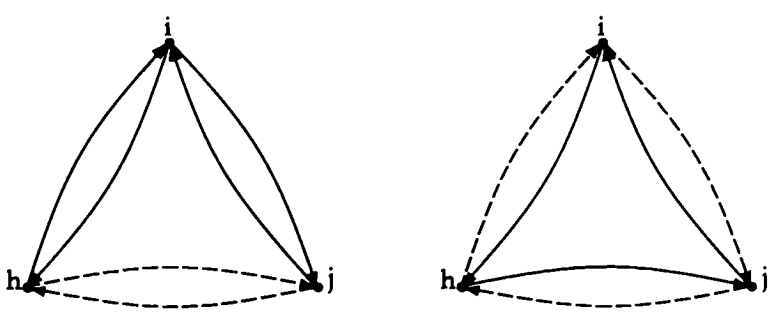

Figure 3.9: Support Graphs of dT and dC Inequalities

Proof. The proof is divided in two parts. In part (a) we show that every incidence vector z of a *PLO* satisfies inequalities (3.1) and (3.2). In part (b) we show that every $0-1$ vector in z-space satisfying inequalities (3.1) and (3.2) is an incidence vector of a PLO.

Proof of Part (a). If z is the incidence vector of a PLO then for every pair of tasks $i, j \in N$ we have that $z_{ij} + z_{ji} \leq 1$ so, if there is an inequality in (3.1) that is violated by z then we must have $z_{hi} + z_{ih} = 1$ and $z_{ij} + z_{ji} = 1$. However, in this case necessarily $z_{hj} + z_{jh} = 1$. The six possible cases are depicted in Figure 3.10.

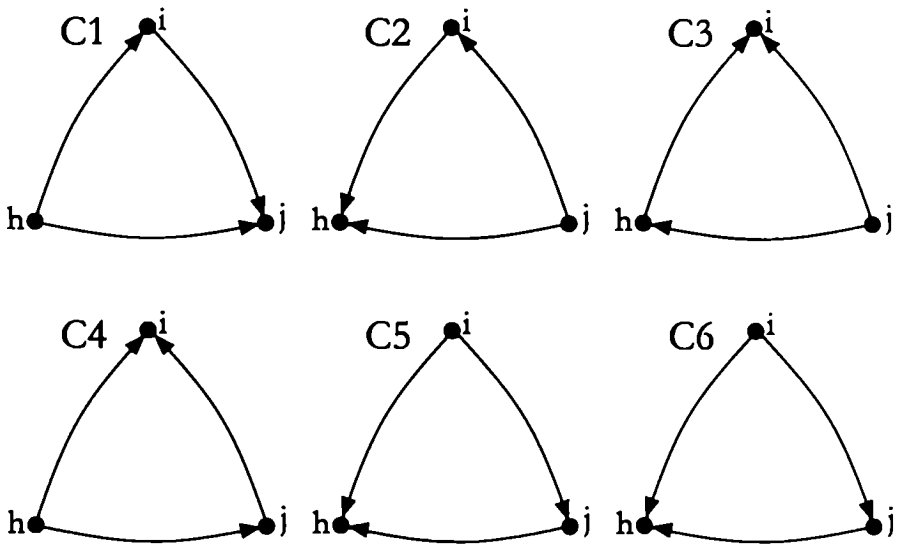

Figure 3.10: Six possible acyclic triangles

Notice that, due to transitivity, the direction of the arc between h and j is mandatory in cases C1 and C2, while cases C3-C6 leave the direction free. However, the existence of an arc between h and j is mandatory in all cases due the weak comparability property of linear ordering partitions.

Now, concerning inequality (3.2), since z is the incidence vector of a PLO, $z_{hi} + z_{ij} + z_{jh} \leq 2$. Otherwise the left hand side would sum 3 and there would be a cycle. But transitivity implies that if two of these arcs are in the solution one of the arcs (i, h), (j, i) or (h, j) should be also in the PLO and the result follows.

Proof of Part (b). Let z be an incident vector of a spanning subgraph D' of D that satisfies (3.1) and (3.2). We will show that D' is a PLO of D. Assume that z partitions the underlying graph of D into $p \leq n$ connected components.

First notice that the inequality $z_{ij} + z_{ji} \leq 1$, $\forall\ i \neq j \in N$, is obtained by adding the two (3.1) inequalities below:

$$z_{ij} + z_{ji} + z_{hi} + z_{ih} - z_{jh} - z_{hj} \leq 1,$$
$$z_{ij} + z_{ji} - z_{hi} - z_{ih} + z_{jh} + z_{hj} \leq 1.$$

Secondly the transitivity inequality $z_{hi} + z_{ij} - z_{hj} \leq 1$ can be obtained by adding (3.1) and (3.2) below:

$$z_{hi} + z_{ih} + z_{ij} + z_{ji} - z_{hj} - z_{jh} \leq 1,$$
$$z_{hi} - z_{ih} + z_{ij} - z_{ji} - z_{hj} + z_{jh} \leq 1.$$

We now prove two facts that will complete the proof of part (b).

Fact A: The underlying graph of each component induced by z in D is a clique.

This result is obvious when the component has one or two vertices. If it has three vertices (since it is connected) the inequalities (3.1) ensure that there must be three arcs joining the vertices. A simple induction on the number of vertices in the component can now show that we must have a clique. Suppose that there are vertices i and j, in the same component, such that no arc connects them. As they are part of the same connected component, there is a path between them. Let such path passes through the vertices $(i = u_1, \ldots, u_h = j)$. As u_1 is connected with u_2 and u_2 is connected with u_3, by applying inequality (3.1) we find that u_1 is connected with u_3. By induction, one can see that if u_1 is connected with u_{h-1} and u_{h-1} is connected with u_h then u_1 is connected with u_h. This implies that vertices i and j are connected by an arc. Therefore as any pair of vertices is connected, the component is a clique.

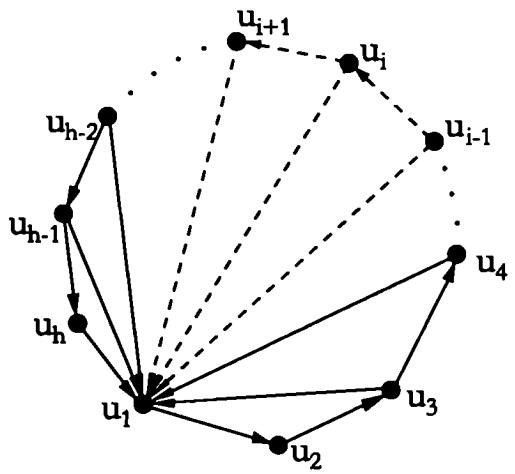

Figure 3.11: Cycle $(u_1, u_2, \ldots, u_{h-1}, u_h)$

Fact B: The subgraph induced by z in D is acyclic.

Suppose there is a cycle C in the subgraph D' represented by z. Let $(u_1, u_2, \ldots, u_{h-1}, u_h)$ be such cycle (see Figure 3.11).

From the transitivity inequality for vertices u_{h-1}, u_h, and u_1, the arc (u_{h-1}, u_1) belongs to D'. If we consider the cycle $(u_1, u_2, \ldots, u_{h-1}, u_1)$ and use the same reasoning again, we get a smaller cycle (of $h-2$ vertices) that must be in D'. Continuing in this way, we end up by concluding that the cycle (u_1, u_2, u_3, u_1) is in C. But this is impossible, since otherwise there would be a dC inequality (3.2) violated by z. This completes the proof of fact B.

The two facts imply that z partitions D into a complete acyclic digraph or, in other words, it partitions D in linear orderings. This completes the proof. □

Theorem 3.12 *For a digraph D, a correct formulation for the partition of D into at most m linear*

orderings is given by the following system:

$$\sum_{k=0}^{m-1} y_{jk} = 1, \qquad j \in N, \tag{3.3}$$

$$z_{ij} + z_{ji} + y_{ik} - y_{jk} \leq 1, \qquad (i,j,k) \in N^2 \times M, i \neq j, \tag{3.4}$$

$$y_{ik} + y_{jk} - z_{ij} - z_{ji} \leq 1, \qquad (i,j,k) \in N^2 \times M, i \neq j, \tag{3.5}$$

$$z_{hi} + z_{ij} + z_{jh} - z_{ih} - z_{ji} - z_{hj} \leq 1, \qquad (h,i,j) \in N^3,\ h \neq i \neq j \neq h. \tag{3.6}$$

with $z \in \mathbf{B}^{n(n-1)}$ *and* $y \in \mathbf{B}^{nm}$ *when* $n \geq 3$.

Proof. Once again the proof, is divided in two parts: (a) every incident vector (y,z) of a PLO satisfies inequalities (3.3) to (3.6), (b) every $0-1$ vector in y,z-space satisfying inequalities (3.3) to (3.6) is an incidence vector of a PLO.

Proof of Part (a). Let (y,z) be an incident vector of a PLO. Equations (3.3) holds, since each task should be assigned to exactly one processor.

For every pair of tasks t_i and t_j, $z_{ij} + z_{ji} \leq 1$. Thus, inequality (3.4) could be violated only if $z_{ij} = 1$ or $z_{ji} = 1$, $y_{ik} = 1$ and $y_{jk} = 0$. But this is clearly impossible, because $z_{ij} = 1$ or $z_{ji} = 1$ would imply that both t_i and t_j belong to the same subset of the partition of N and, therefore, if $y_{ik} = 1$ then $y_{jk} = 1$. Thus, inequality (3.4) holds.

The only way inequality (3.5) could be violated would be if $y_{ik} = y_{jk} = 1$. But, the nature of the problem implies that $z_{ij} = 1$ or $z_{ji} = 1$ and inequality (3.5) holds.

Now, concerning inequality (3.6), a similar argument to that used for inequality (3.2) in the proof of Theorem 3.11 holds and this completes the proof of (a).

Proof of Part (b). First notice that inequalities $z_{ij} + z_{ji} \leq 1$ can be obtained by adding $z_{ij} + z_{ji} + y_{ik} - y_{jk} \leq 1$ to $z_{ij} + z_{ji} + y_{jk} - y_{ik} \leq 1$ and dividing the resulting inequality by 2.

Now, let N_k be the set of vertices for which $y_{ik} = 1$. The subgraph induced by (y,z) in D whose vertex set is N_k is complete, because if y_{ik} and y_{jk} are both set to 1, then inequality (3.5) forces either z_{ij} or z_{ji} to be 1.

Now we show that the graph D' induced by (y,z) in D is acyclic. Since every connected component of D' is complete, cycles can only go through vertices belonging to the same component. Therefore, we restrict ourselves to prove that for any connected component of D', inequalities (3.3) to (3.5) imply inequalities (3.1). Then, by noticing that (3.2) and (3.6) coincide, we apply part (b) of Theorem 3.11 to complete the proof.

Consider three tasks (vertices) i, j and h in N and assume they all belong to the same component N_k of D'. By adding inequalities $z_{ih} + z_{hi} \leq 1$, $z_{hj} + z_{jh} \leq 1$ and (3.5) we obtain

$$z_{hi} + z_{ih} + z_{hj} + z_{jh} - z_{ij} - z_{ji} \leq 3 - y_{jk} - y_{ik}. \tag{3.7}$$

Since tasks i and j are in N_k, we have that $y_{jk} = y_{ik} = 1$. Thus, for processor k, (3.7) coincides with (3.1) and by repeating the steps in the proof of part (b) of Theorem 3.11 we can complete the proof. □

Theorem 3.13 $dim(P_{PLO}(D_n)) = n(n-1)$.

Proof. We will exhibit $n(n-1)+1$ affinely independent points in the polytope $P_{PLO}(D_n)$. Since D_n is a complete digraph, the incidence vector of every single pair of vertices is a *valid point* of $P_{PLO}(D_n)$. Thus, the polytope contains these $n(n-1)$ affinely independent points. Moreover, since the null vector is feasible, we obtain $n(n-1)+1$ affinely independent points in $P_{PLO}(D_n)$. Hence, as $dim(P_{PLO}(D_n)) \leq n(n-1)$ because the polytope is embedded in $\mathrm{R}^{n(n-1)}$, we conclude that $dim(P_{PLO}(D_n)) = n(n-1)$. □

Theorem 3.14 *Let $|M| = p = n$ be the maximum size of a feasible partition in linear orderings of D_n. Then $dim(P^{y_p}_{PLO}(D_n)) = 2n(n-1)$.*

Proof. Let $\gamma y + \pi z = \pi_0$ be a hyperplane containing $P^{y_p}_{PLO}(D_n)$. Then, $\gamma y + \pi z = \pi_0$ is a linear combination of equations (3.3). We will construct successive feasible solutions (y^i, z^i) and by means of simple algebraic operations deduce the general form of $\gamma y + \pi z = \pi_0$.

Let (y^1, z^1) be a feasible solution where $y^1_{ii} = 1$, for all $i \in N$, $y^1_{ij} = 0$, for all $i \neq j$, $i, j \in N$. Clearly $z^1 = 0$. Let (y^2, z^2) be another feasible solution such that $y^2_{hh} = 1$, for all $h \in N \setminus \{i, j\}$, $y^2_{ij} = y^2_{ji} = 1, y^2_{uv} = 0$ otherwise. Once again we have $z^2 = 0$. Subtracting $\gamma y^2 + \pi z^2 = \pi_0$ from $\gamma y^1 + \pi z^1 = \pi_0$, we get

$$\gamma_{ii} + \gamma_{jj} = \gamma_{ij} + \gamma_{ji}. \tag{3.8}$$

Let (y^3, z^3) be such that $y^3_{ii} = y^3_{ji} = z^3_{ij} = 1, y_{hh} = 1$, for all $h \in N \setminus \{i, j\}, y^3_{uv} = z^3_{rs} = 0$ otherwise. Let (y^4, z^4) be such that $y^4_{jj} = y^4_{ij} = z^4_{ij} = 1, y^4_{hh} = 1$, for all $h \in N \setminus \{i, j\}, y^4_{uv} = z^4_{rs} = 0$ otherwise. Subtracting $\gamma y^4 + \pi z^4 = \pi_0$ from $\gamma y^3 + \pi z^3 = \pi_0$, we get

$$\gamma_{ii} + \gamma_{ji} = \gamma_{ij} + \gamma_{jj}. \tag{3.9}$$

Subtracting (3.9) from (3.8), we get

$$\gamma_{ji} = \gamma_{jj}. \tag{3.10}$$

Operating with (y^1, z^1), (y^3, z^3) and (3.10) yields to the following equations:

$$\sum_{h \in N, h \neq j, h \neq i} \gamma_{hh} + \gamma_{ii} + \gamma_{jj} = \pi_0, \tag{3.11}$$

$$\sum_{h \in N, h \neq j, h \neq i} \gamma_{hh} + \gamma_{ii} + \gamma_{ji} + \pi_{ij} = \pi_0. \tag{3.12}$$

Subtracting and applying identity (3.10), we conclude that $\pi_{ij} = 0$, for all $i \neq j \in N$. This shows that all π_{ij} are 0. Moreover, by defining $\gamma_{jk} \equiv \alpha_j$ for all $k \in M$, we obtain from (y^1, z^1) that $\pi_0 = \sum_{j \in N} \alpha_j$. Hence,

$$P^{y_p}_{PLO}(D_n) \subset \{(y, z) : (y, z) \in \mathrm{R}^{2n(n-1)} \text{ such that } \gamma y + \pi z = \pi_0\},$$

with $(\gamma y + \pi z = \pi_0) = \sum_{i=1}^{n} \alpha_i (\sum_{k \in M} y_{ik} = 1)$, which implies that $dim(P^{y_p}_{PLO}(D_n)) = 2n(n-1)$. □

Once we have established the dimension of the PLO polytope, we are now ready to study the strength of some valid inequalities. The forthcoming sections are devoted to prove that some of these inequalities define facets of the polytope. This shows, at least theoretically, that they are useful in tightening our integer formulation of the scheduling problem.

3.3 Valid Inequalities and Facets

In this section we investigate the dimension and the facial structure of the PLO polyhedra. We start by recalling the two well known methods used to prove that an inequality defines a facet of a given polyhedron. Then, we study the inequalities that are part of the integer programming formulation of the PLO in the z-space to determine whether they define facets of $P_{PLO}(D_n)$ or not. Next, we search for new inequalities that are not part of the formulation and that define facets of $P_{PLO}(D_n)$. Finally, we establish a lifting result which allows us to obtain facets for the polytope $P^{y_p}_{PLO}(D_n)$.

The two basic methods used to prove the facet defining property can be described as follows:

Method 3.1 (Direct Construction) [10] Suppose that polyhedron P is full dimensional. A direct proof that the inequality $\pi z \leq \pi_0$ defines a facet of P consists in exhibiting a set of $\dim(P)$ affinely independent points in P, each of which satisfies $\pi z \leq \pi_0$ with equality. This proves that the face induced by $\pi z \leq \pi_0$ is a facet F, since $\dim(F) = \dim(P) - 1$. If P is not full dimensional, we first need to establish that $\pi z \leq \pi_0$ is not an implicit equation, e.g. by exhibiting a point $z' \in P$ with $\pi z' > \pi_0$.

To construct a set of affinely independent points we should take advantage of the combinatorial structure of the inequality.

Method 3.2 (Verifying Maximality) [10] Let $\pi z \leq \pi_0$ be a valid inequality inducing a face F of a full dimensional polyhedron P. In this method for proving that $\pi z \leq \pi_0$ is facet inducing, we have to show that the face F is not contained in any larger proper face of F. As P is the convex hull of a finite set S of points, this method can be interpreted as follows. Let $\mathcal{F} = \{s \in S : \pi z_s = \pi_0\}$. Next, we need to show that if $cz \leq d$ is any valid inequality for P such that $\mathcal{F} \subseteq \{z : cz = d\}$, then, in fact, $cz \leq d$ is a positive scalar multiple of $\pi z \leq \pi_0$. If P is not of full dimension, then we must first verify that $\pi z \leq \pi_0$ is not an implicit equation. Furthermore, we must allow for the possibility that $cz \leq d$ and $\pi z \leq \pi_0$ differ not only by a positive scalar multiple, but also by a linear combination of the implicit equations for P.

The procedure goes step by step, using properties of $\mathcal{F}$ to gather information about the coefficients of c.

3.3.1 Trivial Inequalities

A natural question while studying polyhedra associated to combinatorial optimization problems formulated as $0-1$ integer program is whether or not the trivial inequalities of the form $x \geq 0$ and $x \leq 1$ define facets. This question is investigated below.

Theorem 3.15 *The nonnegativity constraints $z_{ij} \geq 0$ define facets of $P_{PLO}(D_n)$.*

Proof. Let $a = (i,j) \in A$. Then $z_{ij} = 0$ is satisfied by the zero vector and all unit vectors $z^{\{a'\}}, a' \in A, a' \neq a$. These $|A|$ vectors are incidence vectors of PLOs and are affinely independent. □

Theorem 3.16 *The upper bound constraints $z_{ij} \leq 1$ do not define facets of $P_{PLO}(D_n)$.*

Proof. Let $a = (i,j) \in A$ and let (h,i,j) be a triangle. Then, the sum of the following four facet-defining inequalities

$$z_{hi} + z_{ih} + z_{ij} + z_{ji} - z_{hj} - z_{jh} \leq 1,$$
$$-z_{hi} - z_{ih} + z_{ij} + z_{ji} + z_{hj} + z_{jh} \leq 1,$$
$$-z_{ji} \leq 0, \text{ and}$$
$$-z_{ji} \leq 0$$

leads to $z_{ij} \leq 1$ and hence this inequality does not defines a facet of $P_{PLO}(D)$. □

3.3.2 Double Triangle Inequalities

In this section we prove that double triangle inequalities (3.1) are facet defining for $P_{PLO}(D_n)$.

Theorem 3.17 *Inequalities $z_{hi} + z_{ij} - z_{jh} + z_{ih} + z_{ji} - z_{hj} \leq 1$ define facets of $P_{PLO}(D_n)$.*

Proof. We apply method 3.3 in order to find $\dim(P_{PLO}(D_n))$ affinely independent points in $P_{PLO}(D_n)$ satisfying (3.1) at equality.

For the sake of clarity, we divide the coordinates of the z variable in four sets: z^+ is the set of four coordinates that appears in (3.1) with positive coefficients; z^- are the two coordinates that appears in (3.1) with negative coefficients; z^ϕ is the set of coordinates $(h,v),(v,h),(j,v),(v,j)$ with $v \neq i$, or (u,v) with $u \notin \{h,i,j\}$ and $v \notin \{h,i,j\}$ that are not present in (3.1) and z^ψ is the set of coordinates that are not present in (3.1) and represent arcs of the form (i,v) or (v,i) with $v \notin \{h,j\}$.

The block matrix below represents the $n(n-1)$ affinely independent points in $P_{PLO}(D_n)$. The incidence vectors of feasible solutions are given in the columns of matrix A. All blocks of the diagonal are identity matrices of dimension corresponding to the number of z^+, z^-, z^ϕ, and z^ψ variables, in that order:

$$A = \begin{bmatrix} I_4 & B_1 & B_2 & B_3 \\ 0 & I_2 & 0 & 0 \\ 0 & 0 & I_{n(n-3)} & B_4 \\ 0 & 0 & 0 & I_{2(n-3)} \end{bmatrix}$$

Matrix B_1 has two ones in each column in order to balance the presence of the negative element of the diagonal of the identity I_2. Matrix B_2 has one 1 in each column in order to satisfy (3.1) at equality. This can always be achieved because the coefficient of the identity $I_{n(n-3)}$ represents an arc with at most one vertex in $\{h,j\}$. Suppose that $z_{hv} = 1$ with $v \in N \setminus \{h,i,j\}$. Then, we can define $z_{ij} = 1$ and all other coordinates z_{uv} of z to be zero, so that (3.1) is satisfied at equality. Matrices B_3 and B_4 have one 1 in each column. Identity $I_{2(n-3)}$ has ones of the form $z_{iv} = 1$ ($z_{vi} = 1$) with $v \in N \setminus \{h,i,j\}$. To achieve the equality in (3.1) with a PLO, we need to consider an arc $(j,v),((v,j))$ (matrix B_4) and one arc (i,j) (matrix B_3).

Since matrix A is upper triangular, it is not singular. Then we have $n(n-1)$ affinely independent points satisfying (3.1) at equality and the proof is complete. □

3.3.3 Double Cycle Inequalities

In this section we prove that double cycle inequalities (3.6) are facet defining for $P_{PLO}(D_n)$.

Theorem 3.18 *Let i, j and k be three distinct vertices of D_n. Then, inequalities $z_{hi} + z_{ij} + z_{jh} - z_{ih} - z_{ji} - z_{hj} \leq 1$ define facets of $P_{PLO}(D_n)$.*

Proof. We will construct a set of $n(n-1)$ affinely independent points satisfying (3.6) at equality. The incidence vectors of feasible solutions are given in the columns of matrix A below. Without loss of generality, we will consider arcs $(h,i),(i,j)$ and (j,h) to index the first three components of each point. Thus the first three columns are easily built by observing that each of these arcs alone is a PLO. Now given any other arc a, we can always choose one of the arcs $(h,i),(i,j),(j,h)$ so as to construct a PLO satisfying (3.6) at equality. These arcs are represented as elements of matrix B.

$$A = \begin{bmatrix} I_3 & B \\ 0 & I_{n(n-1)-3} \end{bmatrix}$$

Matrix A is upper triangular, hence it is not singular. Thus, there are $n(n-1)$ affinely independent points satisfying (3.6) at equality and the proof is complete. □

3.3.4 Lifting Theorems

In this section we prove two useful lifting theorems. These theorems enable us to extend facet defining results to spaces of higher dimension. The first one shows that every inequality that defines a facet of $P_{PLO}(D_n)$ and satisfies a given condition, also defines a facet of $P_{PLO}(D_h), h > n$. The second one shows that under certain assumptions a facet defining inequality of $P_{PLO}(D_n)$ also defines a facet of $P^{y_p}_{PLO}(D_n)$ for some values of p. This last theorem will allow us to extend the facet results of the previous subsections to the PLO polytope in the (y,z)-space.

Theorem 3.19 *Suppose $\pi z \leq \pi_0$ defines a nontrivial facet of $P_{PLO}(D_h)$. Then, this inequality also defines a facet of $P_{PLO}(D_n), n > h$, provided that the following condition is satisfied:*

(L) there exists a PLO $P = A_h(W_1, \ldots, W_p)$ of D_h and a vertex $v \in N_h$ such that $\pi z^P = \pi_0$ and $\{v\} = W_i$ for some $i \in \{1, \ldots, p\}$, i.e., the i-th subset of the partition is a singleton

where N_h is the set of vertex of D_h.

Proof. We will show that the given inequality defines a facet of $P_{PLO}(D_{h+1})$. The statement of the theorem then follows by induction, since condition (L) remains satisfied in D_{h+1}.

Let $N_h = \{1, \ldots, h\}$, $N_{h+1} = N_h \cup \{h+1\}$, and $\pi = (\pi_{ij})_{(i,j)\in A_h}$, $\overline{\pi} = (\overline{\pi}_{ij})_{(i,j)\in A_{h+1}}$, where $\overline{\pi}_{ij} = \pi_{ij}$ for $(i,j) \in A_h$ and $\overline{\pi}_{ij} = 0$ for $(i,j) \in A_{h+1} \setminus A_h$. The validity of $\pi z \leq \pi_0$ for $P_{PLO}(D_{h+1})$ is obvious.

Since $\pi z \leq \pi_0$ defines a nontrivial facet of $P_{PLO}(D_h)$ then $\pi_0 > 0$. Thus, there are $|A_h|$ PLOs $P_1, \ldots, P_{|A_h|}$ whose incidence vectors are linearly independent and satisfy $\pi z \leq \pi_0$ with equality.

Each P_i is also a linear ordering of D_{h+1} and satisfies $\pi z^{P_i} = \pi_0$. Thus, let M be the nonsingular $|A_h| \times |A_h|$ matrix whose columns are the vectors z^{P_i}. We may assume that the rows and columns of M

are arranged in such way that: *(a)* the last $2(h-1)$ rows of M correspond to the arcs $(i,v),(v,i)$ with $i \in N_h \backslash \{v\}$, where v is the special node in condition (L), and *(b)* the lower right corner $2(h-1)\times 2(h-1)$ submatrix of M, denoted by N, is nonsingular. Notice that such arrangement of the columns of M exists, otherwise M would be singular.

From the $2(h-1)$ linear ordering partitions $P_{|A_h|-2h+3}, \cdots, P_{|A_h|}$, whose incidence vectors are the last $2(h-1)$ columns of M, we construct $2(h-1)$ linear ordering partitions of D_{h+1} as follows. For $i \in \{|A_h|-2h+3, \ldots, |A_h|\}$, let $(Y_i, A_h(Y_i))$ be the linear ordering of P_i containing v, i.e., $v \in Y_i$. If $|Y_i| < 2$, then v would be a singleton and, in this case, N would contain a null column which imply that N is singular, a contradiction. Hence $|Y_i| \geq 2$ holds. Now, let us define the sets Q_i as follows:

$$Q_i = P_i \cup \{(j,h+1) : (j,v) \in A_h(Y_i)\} \cup \{(h+1,j) : (v,j) \in A_h(Y_i)\} \cup \{(v,h+1)\}.$$

Then $\pi z^{Q_i} = \pi_0$ holds by construction. Finally, let P be the particular linear ordering described in condition (L) and define the sets:

$$Q_{v,h+1} = P \cup \{(v,h+1)\},$$
$$Q_{h+1,v} = P \cup \{(h+1,v)\}.$$

It is easy to check that $\pi z^{Q_{v,h+1}} = \pi z^{Q_{h+1,v}} = \pi_0$

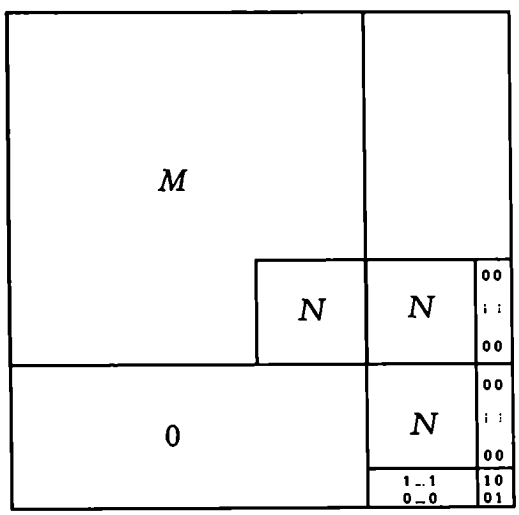

Figure 3.12: Matrix $\overline{M}$ (proof of Theorem 3.19)

Assume that $\overline{M}$ is the $|A_{h+1}| \times |A_{h+1}|$ matrix whose rows are the incidence vectors (in $\mathbb{B}^{A_{h+1}}$) of the linear ordering partitions $P_1, \ldots, P_{|A_h|}, Q_{|A_h|-2h+3}, \cdots, Q_{|A_h|}, Q_{v,h+1}, Q_{h+1,v}$. Then, $\overline{M}$ can be put into the form shown in Figure 3.12, where M and N are nonsingular. Obviously, $\overline{M}$ is nonsingular. Thus, there are $|A_{h+1}|$ PLOs in D_{h+1} whose incidence vectors satisfy $\pi z \leq \pi_0$ and are linearly independent. This implies that $\pi z \leq \pi_0$ defines a facet of $P_{PLO}(D_{h+1})$. □

The next theorem gives some necessary conditions under which the lifting of facet defining inequalities from z-space to facet defining inequalities in the (y,z)-space is possible for the PLO polytopes. The lemma below is needed to prove this result.

Lemma 3.1 *Let F be a face of $P^{y_p}_{PLO}(D_n)$ that is contained in the hyperplane $\gamma y + \pi z = \pi_0$. Then $\gamma_{jk} = \alpha_j$ for all $k \in M$ provided the following condition is satisfied:*

(X) for all $k \in M$ there exists a point (y^*, z^*) in F for which j is the only task assigned to processor k and there exists a processor $\ell \in M \setminus \{k\}$ for which no task is assigned. Moreover, if task j is assigned to processor ℓ instead of processor k another point in F is obtained.

Proof. This result can be easily obtained by applying Method 3.3. □

Theorem 3.20 *Let $\beta z \leq \beta_0$ a facet defining inequality for $P_{PLO}(D_n)$. Suppose that for every vertex (task) j in D_n property (X) holds with respect to the hyperplane $\beta z = \beta_0$. Then $\beta z \leq \beta_0$ is facet defining for $P^{y_n}_{PLO}(D_n)$.*

Proof. Let $F = \{(y,z) \in P^{y_n}_{PLO}(D_n) : \beta z = \beta_0\}$ be a face of $P^{y_n}_{PLO}(D_n)$ and $F' = \{(y,z) \in P^{y_n}_{PLO}(D_n) : \gamma y + \pi z \leq \pi_0\}$ be a generic face of $P^{y_n}_{PLO}(D_n)$ such that $F \subseteq F'$. We apply Method 3.3. If we prove that $\gamma y + \pi z \leq \pi_0$ is a positive scalar multiple of $\beta z \leq \beta_0$, plus a linear combination of equations (3.3), the proof will be completed.

Since property (X) holds, by Lemma 3.1 we have that $\gamma_{jk} = \alpha_j$ for all $k \in M$. This means that the contribution in the left hand side of $\gamma y + \pi z \leq \pi_0$ by assigning task j to any processor is the same, which proves the relations of the coefficients for the y variables. Now, given any two integer points in F, say (y_1, z_1) and (y_2, z_2), since $F \subseteq F'$ we have that $\gamma y_1 + \pi z_1 = \gamma y_2 + \pi z_2$. From previous results on the γ coefficients, this leads to $\pi z_1 = \pi z_2$. Thus by using the same affinely independent points as in the proof that $\beta z \leq \beta_0$ is facet defining for $P_{PLO}(D_n)$ we complete the proof. □

3.3.5 Double Simplex Inequalities

This set of inequalities is a generalization of the dT inequalities in (3.1). Let $S = \{v_0\}$ and $T = \{v_1, v_2, \ldots, v_h\}$ be two disjoint sets of vertices of D_n. We define the double simplex inequality associated with S and T as follows:

$$z(\delta(S,T)) - z(A(T)) \leq 1. \tag{3.13}$$

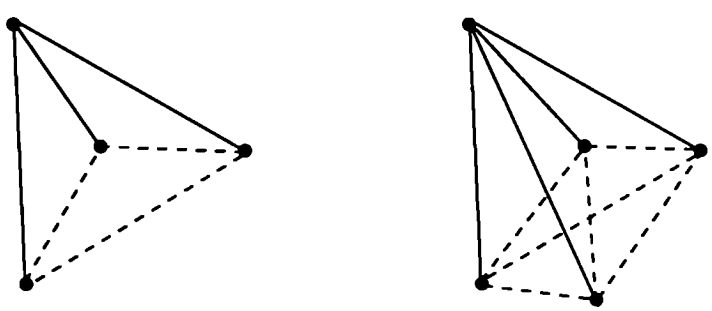

Figure 3.13: Double Simplex Inequalities, dS_3 and dS_4

Theorem 3.21 *The double simplex inequality (3.13) defines a facet of $P_{PLO}(D_n)$.*

Proof. We prove the validity of (3.13) by induction on $|T|$ and applying the Chvàtal-Gomory procedure. For $|T| = 2$ the result is immediate, since in this case we have a dT inequality. So, assume that $|T| = t \geq 3$. By the induction hypothesis, for every $v \in T$ the double simplex $(S, T \setminus \{v\})$-inequality

$$z(\delta(S, T \setminus \{v\})) - z(A(T \setminus \{v\})) \leq 1$$

is valid for $P_{PLO}(D_n)$. Adding up these inequalities for all $v \in T$ we obtain that

$$(t-1)z(\delta(S,T)) - (t-2)(z(A(T)) \leq t.$$

Since $-(z(A(T)) \leq 0$ is also valid for $P_{PLO}(D_n)$, by adding this inequality to the above one we get

$$(t-1)(z(\delta(S,T)) - z(A(T))) \leq t.$$

Hence, since $(t-1) > 1$,

$$z(\delta(S,T)) - z(A(T)) \leq \frac{t}{t-1},$$

which implies that

$$z(\delta(S,T)) - z(A(T)) \leq \lfloor \frac{t}{t-1} \rfloor = 1.$$

Thus inequality (3.13) is valid for $P_{PLO}(D_n)$.

We now apply Method 3.3 in order to find $\dim(P_{PLO}(D_n))$ affinely independent points in $P_{PLO}(D_n)$ satisfying (3.13) with equality. This will proof that this inequality is facet defining.

For the sake of clarity, we separate the coordinates of the z variable in four sets: z^+ the set of $2h$ coordinates that appear in (3.13) with positive coefficients; z^- the set of $h(h-1)$ coordinates that appear in (3.13) with negative coefficients; z^ϕ the set of $(n+h-2)(n-h-1)$ coordinates that are not present in (3.13), and represent arcs with one vertex in T and the other in $N \setminus (S \cup T)$ or both vertices in $N \setminus (S \cup T)$, and finally z^ψ represents the set of $2(n-h-1)$ coordinates that are not present in (3.13), and represent arcs with one vertex in S and the other in $N \setminus (S \cup T)$.

The block matrix A below represents the $n(n-1)$ affinely independent points in $P_{PLO}(D_n)$ and satisfying (3.13) at equality. The incidence vectors of feasible solutions are given as columns of matrix A. All blocks of the diagonal are identity matrices of dimension corresponding to the number of z^+, z^-, z^ϕ, and z^ψ variables in that order.

$$A = \begin{bmatrix} I_{2h} & B_1 & B_2 & B_3 \\ 0 & I_{h(h-1)} & 0 & 0 \\ 0 & 0 & I_{(n+h-2)(n-h-1)} & B_4 \\ 0 & 0 & 0 & I_{2(n-h-1)} \end{bmatrix}$$

Matrix B_1 has two ones in each column in order to balance the presence of the negative element of the diagonal of the identity $I_{h(h-1)}$. Matrix B_2 has a single one in each column in order to satisfy (3.13) with equality. This can always be achieved because a coefficient of the identity $I_{(n+h-2)(n-h-1)}$ represents an arc with exactly one vertex in T. Suppose that the $z_{ij} = 1$ with $i \in T$ and $j \in N \setminus (S \cup T)$. Then it is clear that we can define $z_{i'v} = 1$ with $i' \in T$, $i' \neq i$ and all other coordinates z_{ij} set to zero, so as to obtain $z(\delta(S,T)) - z(A(T)) = 1$.

Matrices B_3 and B_4 have a single one in each column. Identity $I_{2(n-h-1)}$ has ones of the form $z_{iv} = 1$ or $z_{vi} = 1$ with $i \in N \setminus (S \cup T)$. So as to achieve the equality in (3.13) with a PLO, we need to consider an arc (j,v) from T to S (matrix B_3) and another arc (i,j) from T to $N \setminus (S \cup T)$ (matrix B_4).

Matrix A is upper triangular, hence it is not singular. Thus, we have $n(n-1)$ affinely independent points satisfying (3.13) with equality. The proof is complete. □

3.3.6 2-Partition Inequalities

In this section we introduce a class of facet defining inequalities that further generalizes the class of double triangle (3.1) and double simplex (3.13) inequalities.

Let S and T be subsets of N such that $S \cap T = \emptyset$ and $(S \cup T) \subseteq N$. We define the 2-partition inequality induced by S and T, or (S,T)-inequality for short, as:

$$z(\delta(S,T)) - z(A(S)) - z(A(T)) \leq \min\{|S|, |T|\}. \tag{3.14}$$

The support graph of a 2-partition inequality with $S = \{1,2,3\}$ and $T = \{4,5,6,7\}$ is shown in Figure 3.14. Note that, if $|S| = 1$ and $|T| = 2$, the corresponding (S,T)-inequality is a dT inequality. Also, if $|S| = 1$ and $|T| > 2$ the (S,T)-inequality corresponds to a double simplex inequality.

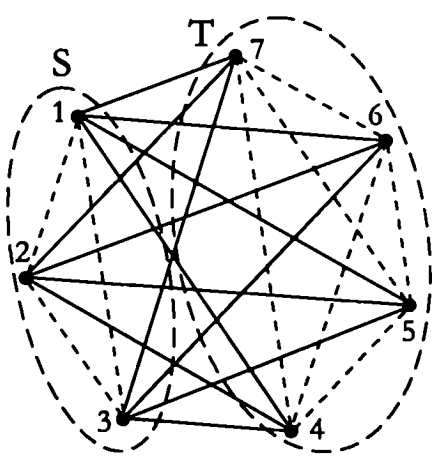

Figure 3.14: 2-partition inequality

Theorem 3.22 *For every $n \geq 3$ and every two nonempty disjoint subsets S,T of N the corresponding 2-partition inequality (3.14) is valid for $P_{PLO}(D_n)$. It defines a facet if and only if $|S| \neq |T|$.*

Proof. Assume without loss of generality that $|S| \leq |T|$. We prove the validity of (3.14) by induction on $|S| + |T|$ applying the Chvàtal–Gomory procedure. Let $|S| = 1$ and $|T| \geq 1$. For $|T| = 2$ the result is immediate since in this case we have a dT inequality. For $|T| > 2$ the result follows, since the (S, T) inequality becomes a double simplex inequality.

Now, let $|S| = s \geq 2, |T| = t \geq 2, |S| + |T| = h$, and suppose that (3.14) is valid for $|S| + |T| \leq h - 1$. For every $v \in S$, consider the $(S \setminus \{v\}, T)$-inequality,

$$z(\delta(S \setminus \{v\}, T)) - z(A(S \setminus \{v\})) - z(A(T)) \leq s - 1, \tag{3.15}$$

and for every $v \in T$ consider the $(S, T \setminus \{v\})$-inequality,

$$z(\delta(S, T \setminus \{v\})) - z(A(S)) - z(A(T \setminus \{v\})) \leq \min\{s, t - 1\}. \tag{3.16}$$

By induction hypothesis, all these inequalities are valid for $P_{PLO}(D_n)$. Adding up the inequalities (3.15) for every $v \in S$ and (3.16) for every $v \in T$ we obtain

$$(s + t - 2)(z(\delta(S, T)) - z(A(S)) - z(A(T))) \leq s(s - 1) + t(\min\{s, t - 1\}). \tag{3.17}$$

If $|S| < |T|$, then (3.17) yields

$$z(\delta(S, T)) - z(A(S)) - z(A(T)) \leq \lfloor \frac{s(s + t - 1)}{s + t - 2} \rfloor = \lfloor s + \frac{s}{s + t - 2} \rfloor = |S|.$$

If $|S| = |T|$, i.e., $s = t$, then (3.17) can be written as

$$(2s - 2)(z(\delta(S, T)) - z(A(S)) - z(A(T))) \leq s(2s - 2),$$

which implies that

$$z(\delta(S, T)) - z(A(S)) - z(A(T)) \leq |S|.$$

This completes the proof that inequality (3.14) is valid for $P_{PLO}(D_n)$. When $|S| = |T|$ the above proof shows that the inequality (3.14) can be obtained by nonnegative linear combinations of other valid inequalities, and therefore it does not define a facet of $P_{PLO}(D_n)$.

Now assume that $s = |S| < |T|$. We first prove that (3.14) defines a facet of $P_{PLO}(D_h)$, where $h = |S| + |T|$.

Let $F = \{z \in P_{PLO}(D_n) : z(\delta(S, T)) - z(A(S)) - z(A(T)) = s\}$ be the face defined by inequality (3.14) in $P_{PLO}(D_h)$, and let $F' = \{z \in P_{PLO}(D_h) : \pi z = \pi_0\}$ be a generic face of $P_{PLO}(D_h)$ such that $F \subseteq F'$.

Notice that: (a) F is a proper face of $P_{PLO}(D_h)$, since $z = 0$ belongs to $P_{PLO}(D_h) \setminus F$; (b) F is nonempty since if we match each vertex in S with a distinct vertex in T and take one of the arcs joining each of those pairings, we obtain a linear ordering partition whose incidence vectors lies in F and (c) inequality (3.14) is valid for $P_{PLO}(D_h)$.

Therefore, if we apply Method 3.3 and prove that $\pi z \leq \pi_0$ is a scalar multiple of $z(\delta(S, T)) - z(A(S)) - z(A(T)) \leq s$, we can conclude that (3.14) defines a facet of $P_{PLO}(D_h)$.

We will use the following notation. Indices $i_1, \ldots, i_s$ represent an arbitrary ordering of the elements of S and indices $j_1, \ldots, j_s, j_{s+1}, \ldots, j_t$ represent an arbitrary ordering of the elements T.

Let z^1 and z^2 be two points of $P_{PLO}(D_h)$ representing the following set of arcs, both of size s:

$$P_1 = \{(i_1, j_1), \ldots, (i_{s-1}, j_{s-1}), (i_s, j_s)\}$$

and

$$P_2 = \{(i_1, j_1), \ldots, (i_{s-1}, j_{s-1}), (j_s, i_s)\}.$$

It is clear that P_1 and P_2 both represent partitions in linear orderings and that $z^1, z^2 \in F \subseteq F'$. Therefore, we conclude that $\pi z^1 = \pi z^2$ and

$$\pi_{i_s j_s} = \pi_{j_s i_s}, \ \forall\, i_s \in S, \ \forall\, j_s \in T. \tag{3.18}$$

Hence, the coefficients of opposite arcs in $\delta(S, T)$ are equal.

Let z^3 be a point of $P_{PLO}(D_h)$ representing the PLO of size s with the following set of arcs:

$$P_3 = \{(i_1, j_1), \ldots, (i_{s-1}, j_{s-1}), (i_s, j_{s+1})\}.$$

Since $z^1, z^3 \in F \subseteq F'$, it follows that $\pi z^1 = \pi z^3$ and we get

$$\pi_{i_s j_s} = \pi_{i_s j_{s+1}}, \ \forall\, i_s \in S, \ \forall\, j_s, j_{s+1} \in T.$$

As indices denote an arbitrary ordering of elements of S and T, we can conclude that given a fixed vertex $i \in S$,

$$\pi_{ij} = \pi_{ij'}, \ \forall\, j, j' \in T. \tag{3.19}$$

Let z^4 be a point of $P_{PLO}(D_h)$ representing the set $P_4 = P_1 \cup \{(i_s, j_{s+1}), (j_s, j_{s+1})\}$. As both $z^1, z^4 \in F \subseteq F'$ we have that $\pi z^1 = \pi z^4$. Thus,

$$\pi_{i_s j_{s+1}} + \pi_{j_s j_{s+1}} = 0, \ \forall\, i_s \in S, \ \forall\, j_s, j_{s+1} \in T. \tag{3.20}$$

Let z^5 be a point of $P_{PLO}(D_h)$ representing the set $P_5 = P_1 \cup \{(i_s, j_{s+1}), (j_{s+1}, j_s)\}$. Clearly $z^1, z^5 \in F \subseteq F'$ are incidence vectors of PLOs. Therefore, $\pi z^1 = \pi z^5$. Thus,

$$\pi_{i_s j_{s+1}} + \pi_{j_{s+1} j_s} = 0, \ \forall\, i_s \in S, \ \forall\, j_s, j_{s+1} \in T, \tag{3.21}$$

and from (3.20) and (3.21) we get

$$\pi_{jj'} = \pi_{j'j}, \ \forall\, j, j' \in T. \tag{3.22}$$

Let z^6 and z^7 be two points of $P_{PLO}(D_h)$ representing the following sets of arcs

$$P_6 = \{(i_1, j_1), \ldots, (i_{s-2}, j_{s-2}), (i_{s-1}, j_s), (i_s, j_{s-1})\}$$

and $P_7 = P_6 \cup \{(i_{s-1}, j_{s+1}), (j_s, j_{s+1})\}$. Since $z^6, z^7 \in F \subseteq F'$, it follows that $\pi z^6 = \pi z^7$ therefore

$$\pi_{i_{s-1}j_{s+1}} + \pi_{j_s j_{s+1}} = 0, \ \forall\, i_{s-1} \in S, \ \forall\, j_s, j_{s+1} \in T. \tag{3.23}$$

Hence, from (3.20) and (3.23) we get $\pi_{i_s j_{s+1}} = \pi_{i_{s-1} j_{s+1}}$. As indices denote an arbitrary ordering of elements of S and T, we can conclude that given a fixed vertex $j \in T$

$$\pi_{ij} = \pi_{i'j}, \ \forall\, i, i' \in S. \tag{3.24}$$

Thus, from (3.19), (3.24) and (3.18), we get

$$\pi_{ij} = \pi_{ji} = \lambda, \ \forall\, i \in S, \ \forall j \in T. \tag{3.25}$$

From (3.21), (3.22) and (3.25), we get

$$\pi_{jj'} = -\lambda, \ \forall\, j, j' \in T. \tag{3.26}$$

Let z^8 be a point of $P_{PLO}(D_n)$ representing the arc set $P_8 = P_1 \cup \{(i_{s-1}, j_s), (i_s, j_{s-1}), (i_{s-1}, i_s), (j_{s-1}, j_s)\}$. Since $z^1, z^8 \in F \subseteq F'$, we have that $\pi z^1 = \pi z^8$. Hence $\pi_{i_{s-1}j_s} + \pi_{i_s j_{s-1}} + \pi_{i_{s-1}i_s} + \pi_{j_{s-1}j_s} = 0$.

Combining the above equation with (3.25) and (3.26), we obtain $\pi_{ii'} = -\lambda, \ \forall\, i, i' \in S$, and therefore (3.14) defines a facet of $P_{PLO}(D_h)$.

For any vertex $v \in T$ there exists a matching $M \subseteq \delta(S,T)$ of size s, not covering v. Since M is a PLO whose incidence vector lies on F and v is a vertex satisfying condition (L), by Theorem 3.19 the (S,T)-inequality defines a facet of $P_{PLO}(D_n)$, for all $n \geq h$. This completes the proof. □

3.3.7 2-Chorded Cycle Inequalities

Given digraph D_n, let C be a directed cycle in D_n and C^2 the set of 2-chords of C with the same direction of C. In other words, if arcs (h,i) and (i,j) belong to C then arc (h,j) belongs to C^2. Then, the inequality

$$z(C) - z(C^2) \leq \lfloor \frac{|C|}{2} \rfloor, \tag{3.27}$$

is the 2-chorded cycle inequality induced by C. Figure 3.15 shows a 7-cycle and its set of 2-chords.

Theorem 3.23 *Let $C \subseteq A$ be a cycle with $|C| \geq 5$ and let C^2 be the set of 2-chords of C. Then, the 2-chorded cycle inequality (3.27) induced by C is valid for $P_{PLO}(D_n)$ and defines a facet of $P_{PLO}(D_n)$ if and only if $|C|$ is odd.*

Proof. We start by showing the validity of the inequality. Suppose without loss of generality that $C = \langle (1,2), (2,3), \ldots, (2h, 2h+1), (2h+1, 1) \rangle$ so $C^2 = \{(i, i+2) \bmod (2h+1) : 1 \leq i \leq 2h+1\}$ is the

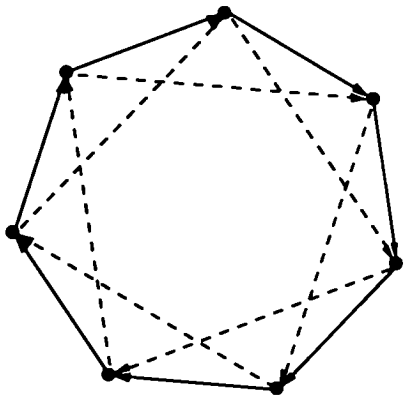

Figure 3.15: 2-chorded cycle inequality

set of 2-chords of C. For each arc (h,j) of C^2, we consider the two arcs (h,i) and (i,j) in C such that they induce the following triangle inequality

$$z_{hi} + z_{ij} - z_{hj} \leq 1. \tag{3.28}$$

Adding $|C|$ inequalities of the form (3.28), one for each arc of C^2, we get

$$2z(C) - z(C^2) \leq |C^2| = |C|. \tag{3.29}$$

Adding inequality $-z(C^2) \leq 0$ that is clearly valid for $P_{PLO}(D_n)$ and dividing the resulting inequality by 2, we get

$$z(C) - z(C^2) \leq \frac{|C|}{2}. \tag{3.30}$$

Since the left hand side of (3.30) is integer in every vertex of $P_{PLO}(D_n)$, by applying the Chvàtal-Gomory procedure we can round the right hand side while keeping validity. On the other hand, it is clear that when $|C|$ is even, (3.30) can be obtained as a linear combination of valid inequalities of the form (3.28) and then it does not define a facet of $P_{PLO}(D_n)$.

Now, we show that the inequality defines a facet of $P_{PLO}(D_{2h+1})$, then by using Theorem 3.19 we extend the result to $n > 2h+1$. Thus, let us suppose that $|C| = 2h+1$, $h \geq 2$. Moreover, assume that all indices are computed modulo $2h+1$.

Let $F = \{z \in P_{PLO}(D_{2h+1}) : z(C) - z(C^2) = h\}$ be the face defined by inequality (3.27) in $P_{PLO}(D_{2h+1})$ and $F' = \{z \in P_{PLO}(D_{2h+1}) : \pi z = \pi_0\}$ be a generic face of $P_{PLO}(D_{2h+1})$, such that $F \subseteq F'$. Notice that: (a) the feasible solution $z = 0$ belongs to $P_{PLO}(D_{2h+1}) \setminus F$, hence F is proper and (b) F is not empty since the arc set $\{(2,3),(4,5),\ldots,(2h,2h+1)\}$ is a PLO and its incidence vector satisfies (3.27) at equality.

We apply Method 3.3 to prove that $\pi z \leq \pi_0$ is a scalar multiple of $z(C) - z(C^2) \leq h$. Then, we conclude that (3.27) defines a facet of $P_{PLO}(D_{2h+1})$.

For each $i \in N$ let P_i be the set of arcs of size h given by $P_i = \{(i+1, i+2), (i+3, i+4), \ldots, (i+2h-1, i+2h)\}$. Notice that P_i defines a partition in linear orderings in which the vertex i is a singleton. Moreover, $z^{P_i} \in F \subseteq F'$ for all $i \in N$. Therefore

$$\pi z^{P_1} = \pi z^{P_2} = \ldots = \pi z^{P_{2h+1}} = \pi_0. \tag{3.31}$$

Since $P_i \triangle P_{i+2} = \{(i, i+1), (i+1, i+2)\}$ and $\pi z^{P_i} = \pi z^{P_{i+2}} = \pi_0$, we conclude that $\pi_{i,i+1} = \pi_{i+1,i+2}$. Thus, for all arcs in C we have the same coefficient, i.e.:

$$\exists\ \lambda \in \mathrm{R}_+ \text{ such that } \pi_e = \lambda\ \forall\ e \in C. \tag{3.32}$$

Let $Q_i = P_i \cup \{(i, i+1), (i, i+2)\}$, for all $i \in N$. Since $z^{Q_i} \in F \subseteq F'$, for all $i \in N$, $\pi z^{P_i} = \pi z^{Q_i}$. Hence,

$$\pi_{i,i+1} + \pi_{i,i+2} = 0. \tag{3.33}$$

Thus, from (3.32) and (3.33), $\pi_{i,i+2} = -\lambda$, and we conclude that

$$\exists\ \lambda \in \mathrm{R}_+ \text{ such that } \pi_f = -\lambda,\ \forall\ f \in C^2. \tag{3.34}$$

From (3.31) and (3.32), we conclude that $\pi_0 = h\lambda$.

Let $R_i = P_i \cup \{(i+1, i), (i+2, i)\}$, for all $i \in N$ (see Figure 3.16 (a)). Since $z^{R_i} \in F \subseteq F'$, for all $i \in N$, $\pi z^{P_i} = \pi z^{R_i}$ and hence

$$\pi_{i+1,i} + \pi_{i+2,i} = 0. \tag{3.35}$$

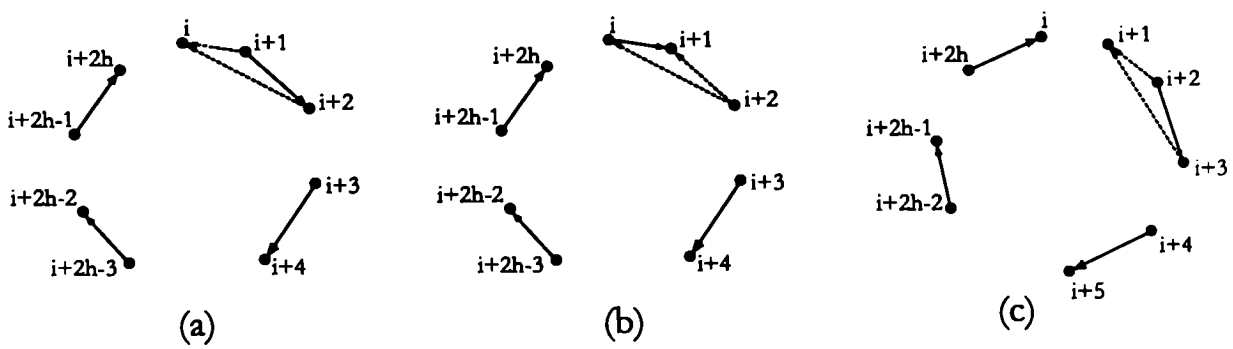

Figure 3.16: R_i, S_i and T_i

Let $S_i = P_{i+2} \cup \{(i+2, i+1), (i+2, i)\}$, for all $i \in N$ (see Figure 3.16 (b)). Since $z^{S_i} \in F \subseteq F'$, for all $i \in N$, $\pi z^{P_i} = \pi z^{S_i}$, we have that

$$\pi_{i+2,i+1} + \pi_{i+2,i} = 0. \tag{3.36}$$

From (3.35) and (3.36), we conclude that $\pi_{i+2,i+1} = \pi_{i+1,i}$. Therefore, we can conclude that

$$\exists\ \alpha \in \mathrm{R}_+ \text{ such that } \pi_{\overline{e}} = \alpha,\ \forall\ \overline{e} \in \overline{C}. \tag{3.37}$$

Let $T_i = P_{i+1} \cup \{(i+2, i+1), (i+3, i+1)\}$, for all $i \in N$ (see Figure 3.16 (c)). But, $z^{T_i} \in F \subseteq F'$ for all $i \in N$, $\pi z^{P_i} = \pi z^{T_i}$. Hence,

$$\pi_{i+2,i+1} + \pi_{i+3,i+1} = 0. \tag{3.38}$$

From (3.36) and (3.38), we conclude that $-\pi_{i+2,i+1} = \pi_{i+2,i} = \pi_{i+3,i+1}$. Therefore,

$$\exists\ \alpha \in \mathrm{R}_+ \text{ such that } \pi_{\overline{f}} = -\alpha,\ \forall\ \overline{f} \in \overline{C}^2. \tag{3.39}$$

At this point, we split the proof in two parts. We first consider the case $h = 2$. Then, we investigate the case $h \geq 3$. For $h = 2$, i.e. $|C| = 5$, consider the partition in linear orderings induced by the arc set $\{(1,2), (3,4), (3,2), (4,2), (3,1), (1,4)\}$.

The incidence vector of this arc set lies in F. Moreover, since the first two arcs are in C, the third arc is in $\overline{C}$ and the three remaining arcs are in $\overline{C}^2$, the previous results imply that $2\lambda + \alpha - 3\alpha = 2\lambda$.

Thus, we conclude that $\alpha = 0$ and the proof for $h = 2$ is complete. So, let us suppose that $h \geq 3$ and assume that $j \in \{3, 5, \ldots, 2h-3\}$.

Let $U_i^1 = P_i \cup \{(i, i+j), (i, i+j+1)\}$, for all $i \in N$. Since $z^{U_i^1} \in F \subseteq F'$ and $\pi z^{P_i} = \pi z^{U_i^1}$, then

$$\pi_{i,i+j} + \pi_{i,i+j+1} = 0. \tag{3.40}$$

Let $U_i^2 = P_i \cup \{(i+j, i), (i, i+j+1)\}$, for all $i \in N$. Since $z^{U_i^2} \in F \subseteq F'$ and $\pi z^{P_i} = \pi z^{U_i^2}$, then

$$\pi_{i+j,i} + \pi_{i,i+j+1} = 0. \tag{3.41}$$

Let $U_i^3 = P_i \cup \{(i+j, i), (i+j+1, i)\}$, for all $i \in N$. Since $z^{U_i^3} \in F \subseteq F'$ and $\pi z^{P_i} = \pi z^{U_i^3}$, then

$$\pi_{i+j,i} + \pi_{i+j+1,i} = 0. \tag{3.42}$$

From (3.40) and (3.41), we conclude that

$$\pi_{i,i+j} = \pi_{i+j,i}. \tag{3.43}$$

From (3.41) and (3.42), we conclude that

$$\pi_{i,i+j+1} = \pi_{i+j+1,i}. \tag{3.44}$$

Then,

$$\pi_{i,i+j} = \pi_{i+j,i} = -\pi_{i,i+j+1} = -\pi_{i+j+1,i}. \tag{3.45}$$

Let $W_i = P_{i-1} \cup \{(i+j+1,i),(i+j+1,i+1),(i+j+2,i),(i+j+2,i+1)\}$, for all $i \in N$ (see Figure 3.17). Since $z^{W_i} \in F \subseteq F'$, $\pi z^{P_i} = \pi z^{W_i}$, hence

$$\pi_{i+j+1,i} + \pi_{i+j+1,i+1} + \pi_{i+j+2,i} + \pi_{i+j+2,i+1} = 0. \tag{3.46}$$

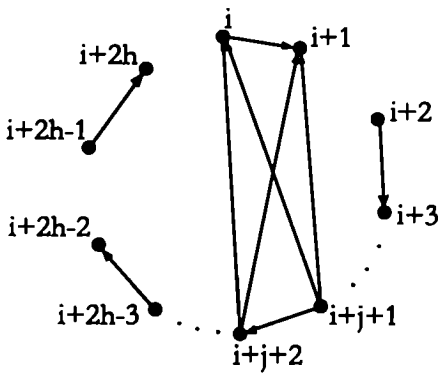

Figure 3.17: W_i

From (3.42) and (3.46), we conclude that

$$\pi_{i+j+1,i} = -\pi_{i+j+2,i}. \tag{3.47}$$

Now, consider the linear ordering partition induced by the arc set $K_i = P_{i-1} \cup \{(i+3,i),(i+3,i+1),(i+2,i+1),(i+2,i)\}$ whose incidence vector is in F. Since $(i+3,i+1)$ and $(i+2,i)$ are both in $\overline{C}^2$ and $(i+2,i+1)$ is in $\overline{C}$, from (3.37) and (3.39) we obtain that $\pi_{i+3,i} = \alpha$. Moreover, from (3.45) and (3.47), we can extend this result to

$$\alpha = \pi_{i+3,i} = \pi_{i,i+3} = -\pi_{i+4,i} = \ldots = \pi_{i+2h-3,i} = \pi_{i,i+2h-3} = -\pi_{i+2h-2,i} = -\pi_{i,i+2h-2}. \tag{3.48}$$

Now, consider the PLO whose incidence vector belongs to F and with arc set given by $P_{i+2h-3} \cup \{(i,i+2h-2),(i,i+2h-1),(i-1,i+2h-2),(i-1,i+2h-1)\}$. Since $(i,i+2h-1)$ and $(i-1,i+2h-2)$ are in $\overline{C}^2$, and $(i-1,i+2h-1)$ is in $\overline{C}$, we obtain $\pi_{i,i+2h-2} - 2\alpha + \alpha = 0 \Rightarrow \pi_{i,i+2h-2} = \alpha$. Comparing the last equation with (3.48), we conclude that $\alpha = 0$.

Finally, to see that the double cycle inequality also defines a facet of $P_{PLO}(D_n)$ for all $n > 2h+1$, observe that every node i together with the PLO P_i satisfies the condition (L) of Theorem 3.19. The result follows. □

3.3.8 Double 2-Chorded Cycle Inequalities

Let C be a cycle in D_n and C^2 be the set of 2-chords of C. The sets formed by the inverse arcs of C and C^2 are denoted as $\overline{C}$ and $\overline{C^2}$, respectively. Figure 3.18 shows a double 7-cycle and its set of 2-chords. The inequality

$$z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq \lfloor \frac{|C|}{2} \rfloor \tag{3.49}$$

is the double 2-chorded cycle inequality induced by C.

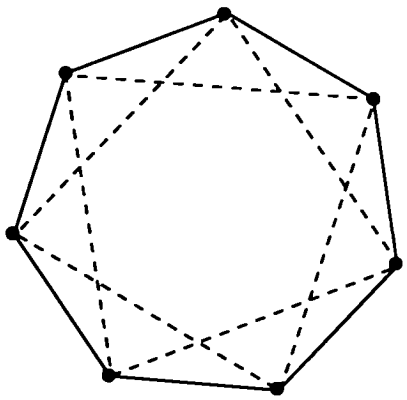

Figure 3.18: Double 2-chorded cycle inequality

Theorem 3.24 *If C is a cycle in D_n with $|C| \geq 5$, then the double 2-chorded cycle inequality (3.49) induced by C is valid for $P_{PLO}(D_n)$. It defines a facet of $P_{PLO}(D_n)$ if and only if $|C|$ is odd.*

Proof. Let $C = \langle (1,2),(2,3),\ldots,(|C|-1,|C|)\rangle$. We first prove the validity of the (3.49). Given three consecutive vertices $i-1, i, i+1$ in C, they induce the following double triangle inequality (3.1)

$$z_{i-1,i} + z_{i,i+1} - z_{i-1,i+1} + z_{i,i-1} + z_{i+1,i} - z_{i+1,i-1} \leq 1.$$

Adding $|C|$ inequalities of this form, one for each triple of consecutive vertices of C we get

$$2z(C) + 2z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq |C|. \tag{3.50}$$

Now, adding $-z(C^2) \leq 0$ and $-z(\overline{C^2}) \leq 0$ that are trivially valid for $P_{PLO}(D_n)$ and dividing the resulting inequality by 2, we get

$$z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq \frac{|C|}{2}. \tag{3.51}$$

The left-hand side of (3.51) is integer for every vertex of $P_{PLO}(D_n)$. Thus, we can apply the Chvàtal–Gomory procedure and round the right hand side of the above inequality. On the other hand,

it is clear that, for $|C|$ even, inequality (3.51) can be obtained as a linear combination of other valid inequalities and therefore it does not define a facet of $P_{PLO}(D_n)$.

Now, suppose that $|C| = 2h+1$ for $h \geq 2$. Moreover, let $C^2 = \{(i, i+2) \bmod (2h+1) : 1 \leq i \leq 2h+1\}$ be the set of 2-chords of C. As in the previous proof, assume that all index computations are done modulo $2h+1$.

Let $F = \{z \in P_{PLO}(D_{2h+1}) : z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) = h\}$ be the face defined by inequality (3.49) in $P_{PLO}(D_{2h+1})$. Let $F' = \{z \in P_{PLO}(D_{2h+1}) : \pi z = \pi_0\}$ be a generic face of $P_{PLO}(D_{2h+1})$ such that $F \subseteq F'$. Notice that: (a) the feasible solution $z = 0$ belongs to $P_{PLO}(D_{2h+1}) \setminus F$, so F is proper, and (b) F is not empty since the arc set $\{(1,2),(3,4),\ldots,(2h,2h+1)\}$ is a linear ordering partition whose incidence vector is in F.

We apply Method 3.3 to prove that $\pi z \leq \pi_0$ is a scalar multiple of $z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq h$ to conclude that (3.49) defines a facet of $P_{PLO}(D_{2h+1})$.

For each $i \in N$ let P_i be the arc set given by $P_i = \{(i+1, i+2), (i+3, i+4), \ldots, (i+2h-1, i+2h)\}$. Notice that $|P_i| = h$ and its incidence vector lies in F. Then,

$$\pi z^{P_1} = \pi z^{P_2} = \ldots = \pi z^{P_{2h+1}} = \pi_0. \tag{3.52}$$

Now, since $P_i \triangle P_{i+2} = \{(i, i+1), (i+1, i+2)\}$ and $z^{P_i} \in F \subseteq F'$ for all $i \in N$, $\pi z^{P_i} = \pi z^{P_{i+2}}$. Therefore, $\pi_{i,i+1} = \pi_{i+1,i+2}$, i.e. all arcs in C have the same coefficient, or

$$\exists\, \lambda \in \mathrm{R}_+ \text{ such that } \pi_e = \lambda,\ \forall\, e \in C. \tag{3.53}$$

For all $i \in N$, let $Q_i = P_i \cup \{(i, i+1), (i, i+2)\}$. Since $z^{Q_i} \in F \subseteq F'$, $\pi z^{P_i} = \pi z^{Q_i}$. Then,

$$\pi_{i,i+1} + \pi_{i,i+2} = 0. \tag{3.54}$$

Thus, from (3.53) and (3.54), $\pi_{i,i+2} = -\lambda$ and we conclude that

$$\exists\, \lambda \in \mathrm{R}_+ \text{ such that } \pi_f = -\lambda,\ \forall\, f \in C^2. \tag{3.55}$$

Let $R_i = P_i \cup \{(i+1, i), (i, i+2)\}$, for all $i \in N$. Since $z^{R_i} \in F \subseteq F'$, $\pi z^{P_i} = \pi z^{R_i}$. Then,

$$\pi_{i+1,i} + \pi_{i,i+2} = 0, \tag{3.56}$$

From (3.55) we conclude that $\pi_{i+1,i} = \lambda$ or, more generally,

$$\pi_{\overline{e}} = \lambda,\ \forall\, \overline{e} \in \overline{C}. \tag{3.57}$$

Let $S_i = P_i \cup \{(i+1, i), (i+2, i)\}$, $\forall\, i \in N$. Since $z^{S_i} \in F \subseteq F'$, $\pi z^{R_i} = \pi z^{S_i}$ and hence $\pi_{i+2,i} = \pi_{i,i+2} = -\lambda$. In general, we can write that

$$\pi_{\overline{f}} = -\lambda,\ \forall\, \overline{f} \in \overline{C^2}. \tag{3.58}$$

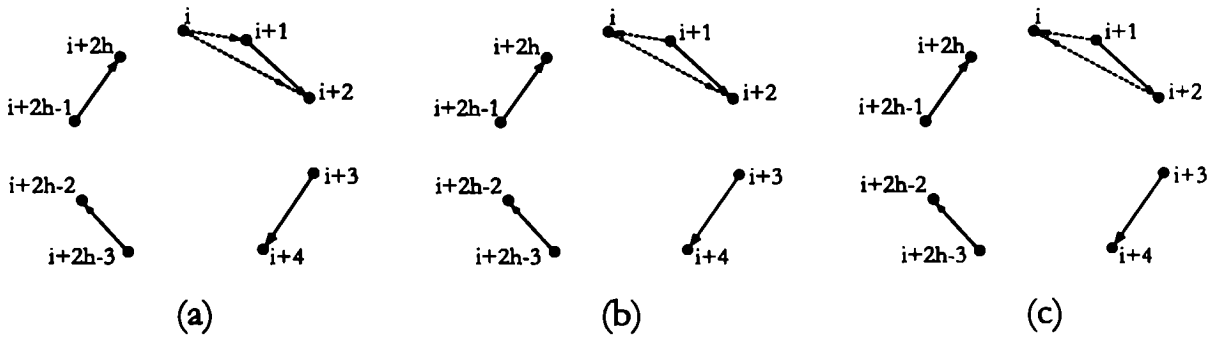

Figure 3.19: Q_i, R_i and S_i

Since for $(C \cup \overline{C} \cup C^2 \cup \overline{C^2}) = D_5$, the results above are enough to prove that (3.49) defines a facet of $P_{PLO}(D_5)$. To prove that it also defines a facet of $P_{PLO}(D_n)$ for all $n > 2h+1$, observe that every node i together with the PLO P_i satisfies the condition (L) of Theorem 3.19. So, from now on, assume that $h \geq 3$.

We will show that $\pi_g = 0$ for all $g \in A \setminus (C \cup \overline{C} \cup C^2 \cup \overline{C^2})$. To this end, assume that $j \in \{3, 5, \ldots, 2h-3\}$.

Let $U_i^1 = P_i \cup \{(i, i+j), (i, i+j+1)\}$, for all $i \in N$. Clearly $z^{U_i^1} \in F \subseteq F'$, therefore $\pi z^{P_i} = \pi z^{U_i^1}$ and

$$\pi_{i,i+j} + \pi_{i,i+j+1} = 0. \tag{3.59}$$

Let $U_i^2 = P_i \cup \{(i+j, i), (i, i+j+1)\}$, for all $i \in N$. Clearly $z^{U_i^2} \in F \subseteq F'$. Therefore $\pi z^{P_i} = \pi z^{U_i^2}$ and

$$\pi_{i+j,i} + \pi_{i,i+j+1} = 0. \tag{3.60}$$

Let $U_i^3 = P_i \cup \{(i+j, i), (i+j+1, i)\}$, for all $i \in N$. Clearly $z^{U_i^3} \in F \subseteq F'$. Thus $\pi z^{P_i} = \pi z^{U_i^3}$ and

$$\pi_{i+j,i} + \pi_{i+j+1,i} = 0. \tag{3.61}$$

From (3.59) and (3.60), we conclude that

$$\pi_{i,i+j} = \pi_{i+j,i}. \tag{3.62}$$

From (3.60) and (3.61), we conclude that

$$\pi_{i,i+j+1} = \pi_{i+j+1,i}. \tag{3.63}$$

Then,

$$\pi_{i,i+j} = \pi_{i+j,i} = -\pi_{i,i+j+1} = -\pi_{i+j+1,i}. \tag{3.64}$$

Let $W_i = P_{i-1} \cup \{(i+j+1,i),(i+j+1,i+1),(i+j+2,i),(i+j+2,i+1)\}$, for all $i \in N$ (see Figure 3.20). Clearly $z^{W_i} \in F \subseteq F'$, therefore $\pi z^{P_i} = \pi z^{W_i}$ and we obtain

$$\pi_{i+j+1,i} + \pi_{i+j+1,i+1} + \pi_{i+j+2,i} + \pi_{i+j+2,i+1} = 0. \tag{3.65}$$

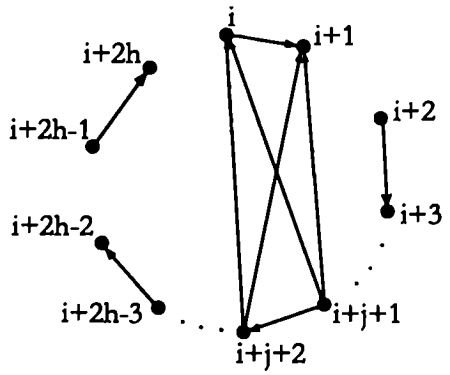

Figure 3.20: W_i

From (3.64) and (3.65), we conclude that

$$\pi_{i+j+1,i} = -\pi_{i+j+2,i}. \tag{3.66}$$

From (3.64) and (3.66), we can write that

$$\alpha_i = \pi_{i+3,i} = \pi_{i,i+3} = -\pi_{i,i+4} = -\pi_{i+4,i} = \ldots = -\pi_{i+2h-2,i} = -\pi_{i,i+2h-2}, \tag{3.67}$$

for some constant α_i and for all $i \in \{1,\ldots,2h+1\}$,. More generally, for $k \in \{3,\ldots,2h-2\}$:

$$\pi_{i+k,i} = \pi_{i,i+k} = (-1)^{k+1}\alpha_i. \tag{3.68}$$

Now, define $i' = i + 2h - 2 - \ell$ and $k = 3 + \ell$ for ℓ ranging from 0 to $2h-5$. If we rewrite equation (3.68) for i' and k, we get

$$\begin{aligned}
\pi_{i'+k,i'} = \pi_{i',i'+k} = (-1)^{k+1}\alpha_{i'} &\Rightarrow \\
\pi_{i+2h-2-\ell+3+\ell,i+2h-2-\ell} = \pi_{i+2h-2-\ell,i} = (-1)^{4+\ell}\alpha_{i'} &\Rightarrow \\
\pi_{i+2h+1,i+2h-2-\ell} = (-1)^{\ell}\alpha_{i+2h-2-\ell} &\Rightarrow \\
\pi_{i,i+2h-2-\ell} = (-1)^{\ell}\alpha_{i+2h-2-\ell}.
\end{aligned}$$

Now, if we let $k' = 2h-2-\ell$, the equation above can be rewritten as

$$\pi_{i+k',i} = \pi_{i,i+k'} = (-1)^{k'}\alpha_{i+k'}, \tag{3.69}$$

with k' ranging from 3 to $2h-2$. Notice here that we have used the fact that ℓ and k' have the same parity, so $(-1)^{\ell} = (-1)^{k'}$.

Theorem 3.26 *Let $C \subseteq A$ be a cycle of length at least 5. Let C^2 be the set of 2-chords of C, and $\overline{C}$ and $\overline{C^2}$ the inverse sets of C and C^2. Then, the double 2-chorded cycle inequality $z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq \lfloor \frac{|C|}{2} \rfloor$ induced by C is valid for $P_{PLO}^{y_p}(D_n)$. Moreover, it defines a facet of $P_{PLO}^{y_p}(D_n)$ if $|C|$ is odd and $p \geq \frac{|C|-1}{2} + 3$.*

Clearly, it is interesting to have more facet defining inequalities in the (y, z)-space, since they tend to define stronger valid inequalities for the scheduling problem then those from the PLO polytope on the z-space. The search of further inequalities of this property is left for a future work.

In next chapter most of the inequalities studied in this chapter are used as cuts in a branch-and-cut algorithm designed to solve the problem of scheduling unrelated processors under precedence constraints.

Chapter 4

Algorithms and Computational Results

In this chapter we describe a branch-and-cut algorithm we have developed for the scheduling problem presented in Section 2.3 and analyze the results obtained in computational experiments. In Section 4.1 we describe several families of cutting planes for the problem of scheduling unrelated processors under precedence constraints and the corresponding separation algorithms. These cutting planes are used in a branch-and-cut algorithm. In Section 4.2 we discuss the implementation of this algorithm. In Section 4.3 we show that this algorithm substantially reduces the size of the enumeration tree. As a direct consequence, we have been able to solve to optimality instances of sizes that earlier formulations were not able to solve and to obtain better solutions for instances not solved to optimality. Sets P_j, Q_j and R_j defined in Section 2.3 are used in the definition of valid inequalities below.

4.1 Valid Inequalities and Separation Algorithms

In this section we show several families of valid inequalities for the problem of scheduling unrelated processors under precedence constraints. These inequalities are based upon our new formulation given by equations (2.13) to (2.20). Separation algorithms for each family of inequalities are proposed.

4.1.1 Least Starting Time Inequalities

Let δ_j be the least time at which task t_j can start without increasing the makespan. The exact calculation of these values is as difficult as finding a solution of the original problem, but we can estimate a good lower bound $\widehat{\delta_j}$ of δ_j to obtain a family of strong valid inequalities. Such inequalities can be expressed as follows:

$$\widehat{\delta_j} \leq e_j, \ \forall j \in N. \tag{4.1}$$

This means that e_j, the starting time of task t_j, cannot be smaller than an underestimate $\widehat{\delta_j}$ of δ_j. The better the estimation $\widehat{\delta_j}$ of δ_j, the stronger inequality (4.1) is.

Mixed Integer Formulation to Compute $\widehat{\delta_j}$

A mixed integer linear formulation is given for obtaining a better value $\widehat{\delta_j}$ for the bound δ_j. This formulation corresponds to the scheduling problem disregarding precedence relations. Solving this

problem can be as hard as solving the original one. However, in practice, this problem has shown to be much easier. In case this simplified problem remains intractable, it is still possible to solve the linear relaxation in order to augment the current value of $\widehat{\delta_j}$ for the tasks that satisfy $\widehat{\delta_j} \leq e_j$ at equality. To build the integer programming model, consider the 0-1 variable x_i^k defined as:

$$x_i^k = \begin{cases} 1 & \text{if task } t_i \text{ is scheduled to processor } p_k, \\ 0 & \text{otherwise.} \end{cases}$$

The formulation forces all tasks preceeding task t_j i.e., those in P_j, to be assigned to a machine, which reads:

$$\min \xi = \widehat{\delta_j},$$

$$\sum_{k=0}^{m-1} x_i^k = 1, \qquad \forall i \in P_j, \tag{4.2}$$

$$\sum_{i \in P_j} d_{ik} x_i^k - \xi \leq 0, \qquad \forall k \in M, \tag{4.3}$$

$$x_i^k \in \{0,1\} \qquad \forall i \in P_j, \forall k \in M.$$

All instances we are using for our tests (see Section 4.2.1) have tasks of the same length (unit lengths). There exists a single faster processor p_0, while the other $m-1$ processors are identical. The *Processor Power Ratio* ($PPR = d_{01}/d_{00}$) is the ratio of the speeds of p_0 and p_1.

With the above considerations, the time d_{ik} taken by processor p_k to process task t_i is defined as $d_{i0} = 1/PPR$ and $d_{ik} = 1$ for $k \neq 0$. The problem can be interpreted as an assignment problem, in which $|P_j|$ tasks have to be assigned to m processors. Processor p_0 is PPR times faster than the other $m-1$ processors, so that it can process PPR tasks in the same time the other processors process one task.

An optimal greedy algorithm for this problem can be designed. This algorithm attempts to balance the number of tasks assigned to each processor, in such a way that all processors are most of the time busy. The first PPR tasks of P_j are assigned to p_0, because every other assignment will give a higher processing time. After p_0 has processed PPR tasks, the next $m-1$ tasks are assigned to processors $p_1, \ldots, p_{m-1}$, with no change in the total processing time. This scheme is repeated until all task of P_j are scheduled. The complexity of this algorithm is $O(|P_j|)$, because each task is assigned to a processor in $O(1)$ operations. A more formal definition of the algorithm is given in Figure 4.1.

Variable c_0 counts the number of tasks not yet assigned. Step 2 controls a loop in which the current task is assigned to processor p_0. This is the case because in this way the *total processing time* ξ for the assigned tasks is not greater than the total processing time if such task is assigned to another processor, otherwise the loop is broken and the algorithm go to Step 3. Constant c_1 controls this loop until PPR tasks are assigned to p_0 or c_0 becomes 0. In Step 3 the current task is assigned to a processor different from p_0. In this case ξ is not increased, because there are idle processors ready to process tasks before time ξ. Constant c_2 controls this loop until $m-1$ tasks are assigned to all processors $p_1, \ldots, p_{m-1}$ or c_0 becomes 0.

Separation Algorithm

Due the simple form of these inequalities, the separation of each of them can be performed with a single comparison in $O(1)$ time provided that the $\widehat{\delta_j}$ values have been computed a priori.

Greedy algorithm for the estimation of δ

1. **Initialization:** Input PPR, m, $|P_j|$, assign $c_0 = |P_j|$, $c_1 = 0$, $c_2 = 0$, $\xi = 0$
2. **Increment ξ:** While($c_0 > 0$ and $c_1 < PPR$)
   3. $c_1 \leftarrow c_1 + 1$
   4. $\xi \leftarrow \xi + 1/PPR$
   5. $c_0 \leftarrow c_0 - 1$
   6. If($c_0 = 0$) output ξ and stop.
3. **Do not change ξ:** While($c_0 > 0$ and $c_2 < m - 1$)
   8. $c_2 \leftarrow c_2 + 1$
   9. $c_0 \leftarrow c_0 - 1$
   10. If($c_0 = 0$) output ξ and stop,
   11. else $c_1 = 0$, $c_2 = 0$ and go to step 2.

Figure 4.1: Algorithm for the estimation of δ

4.1.2 Previously Processed Tasks Inequalities

The aim of these inequalities is to take into account the total processing times of the tasks scheduled in processor p_k before t_j. Clearly, the starting time of t_j is at least as large as the sum of the processing times of these tasks in processor p_k. One such inequality is given by:

$$\sum_{i \in P_j} d_{ik}(y_{jk} + y_{ik} - 1) + \sum_{i \in R_j} d_{ik}(y_{jk} + y_{ik} - z_{ji} - 1) \leq e_j, \ (j,k) \in N \times M \tag{4.4}$$

Inequality (4.4) can be interpreted as follows. If $i \in P_j$, and both tasks t_i and t_j are processed in p_k, then the first summation of (4.4) will be d_{ik}, in any other case this sum should not be greater than zero. For the second summation, in case $i \in R_j$, if task t_j is not executed in processor p_k, then $y_{jk} = 0$ and $y_{jk} + y_{ik} - z_{ji} - 1 \leq 0$ and, therefore, the inequality is redundant. On the other hand if task t_j is executed in processor p_k, then $y_{jk} = 1$ and we distinguish three cases:

1. If t_i is executed in processor p_k and t_i is executed before t_j, then
   $$y_{jk} + y_{ik} - z_{ji} - 1 = 1 + 1 - 0 - 1 = 1$$
   and the contribution of task t_i is d_{ik}
2. If t_i is executed in processor p_k and t_i is executed after t_j,then
   $$y_{jk} + y_{ik} - z_{ji} - 1 = 1 + 1 - 1 - 1 = 0$$
   and the contribution of task t_i is 0.
3. If t_i is not executed in processor p_k, then
   $$y_{jk} + y_{ik} - z_{ji} - 1 = 1 + 0 - 0 - 1 = 0$$
   and the contribution of task t_i is 0.

Strengthening at task t_0

With no loss of generality, we can suppose that all DAGs describing the precedence among the tasks have a source and a sink node. In case this is not true we can create one or two dummy nodes with these characteristics. By doing so, one can strengthen inequalities (4.4) by adding the term $\sum_{h=0,h\neq k}^{m-1} d_{0h}y_{0h}$ to the left-hand side. This summation represents the fact that the initial task t_0 is a bottleneck of the DAG and no task will begin to be processed until t_0 is completely processed. Therefore, (4.4) can be rewritten as:

$$\sum_{h=0,h\neq k}^{m-1} d_{0h}y_{0h} + \sum_{i\in P_j} d_{ik}(y_{jk} + y_{ik} - 1) + \sum_{i\in R_j} d_{ik}(y_{jk} + y_{ik} - z_{ji} - 1) \leq e_j,\ (j,k) \in N \times M. \quad (4.5)$$

Separation Algorithm

Separation for these inequalities is done by simple enumeration of all possible cases in $O(mn(m+n))$ time.

4.1.3 Successors Inequalities

These inequalities consider all the successors of task t_j that will be processed in the same processor, say p_k. Clearly, the sum of their processing times plus the completion time of t_j cannot be greater than the makespan. This fact leads to the following family of inequalities:

$$e_j + \sum_{k=0}^{m-1} d_{jk}y_{jk} + \sum_{i\in Q_j} d_{ik}y_{ik} \leq C_{max},\ (j,k) \in N \times M. \quad (4.6)$$

Strengthening at task t_{n-1}

As we saw in the previous section we can suppose with no loss of generality that task t_{n-1} is a sink of the DAG. The summation $\sum_{h=0,h\neq k}^{m-1} d_{n-1,h}y_{n-1,h}$ represents, the fact that the last task t_{n-1} is a bottleneck of the DAG and it will not begin to be processed until all its precedent tasks are completely processed. Thus, this term can be added to inequalities (4.6) in order to get a stronger version of them, which reads:

$$e_j + \sum_{k=0}^{m-1} d_{jk}y_{jk} + \sum_{i\in Q_j} d_{ik}y_{ik} + \sum_{h=0,h\neq k}^{m-1} d_{n-1,h}y_{n-1,h} \leq C_{max},\ (j,k) \in N \times M. \quad (4.7)$$

Separation Algorithm

Separation for these inequalities is done by simple enumeration of all possible cases in $O(mn(n+m))$ time.

4.1.4 Strengthening of Inequalities (2.14)

The next inequalities were developed as a strengthening of inequalities (2.14). Using $\sum_{k=0}^{m-1} y_{jk} = 1$ we can rewrite inequality $z_{ij} + z_{ji} - y_{jk} \leq 1 - y_{ik}$ as:

$$z_{ij} + z_{ji} - y_{jk} \leq \sum_{h \neq k} y_{ih}.$$

By considering a general partition of the processors in two subsets, we can generalize the partition of M in subsets $\{k\}$ and $M \setminus \{k\}$ into a partition of subsets $M_1 \subset M$ and $M \setminus M_1$. Thus,

$$z_{ij} + z_{ji} - \sum_{h \in M_1} y_{jh} \leq \sum_{h \in M \setminus M_1} y_{ih}.$$

Using $\sum_{k=0}^{m-1} y_{jk} = 1$, we can rewrite the above inequalities as:

$$z_{ij} + z_{ji} + \sum_{k \in M_1} (y_{ik} - y_{jk}) \leq 1, \ (i,j,k) \in N \times N \times M, j \in R_i, M_1 \subset M. \tag{4.8}$$

Thus, if task t_i is processed in any processor of the set M_1 and t_j is processed in any processor of $M \setminus M_1$, then $z_{ij} + z_{ji} \leq 0$ and both z_{ij} and z_{ji} are forced to be null. This includes the original case in which $M_1 = \{k\}$.

Separation Algorithm

Although there is an exponential number of these inequalities, their separation can be performed by simple enumeration of some cases in polynomial time as follows.

Given a fractional solution, if we fix tasks t_i and t_j we have a fixed value of $z^*_{ij} + z^*_{ji}$. Then, we search for a processor k such $y^*_{ik} - y^*_{jk} > 0$ and include those indices k in M_1 exclude those for which $y^*_{ik} - y^*_{jk} \leq 0$. This can be done in $O(n^2m^2)$ time.

4.1.5 Processor Symmetry

The aim of this section is to show how symmetric solutions can be avoided. We exploit the fact that all except one of the processors in our test sets are identical (see Section 4.2.1). There are $(m-1)!$ symmetric optimal solutions that can be obtained from a single optimal solution just by permuting the $m-1$ slowest processors. By permuting, we mean that if a subset of tasks T_1 is assigned to processor p_1 in a given optimal solution and a subset of tasks T_2 is assigned to processor p_2, then a permutation is obtained by assigning the tasks of T_1 to p_2 and the tasks of T_2 to p_1. We notice that p_0 is not involved in these inequalities, because it has a different processing speed with respect to the other processors.

The following inequalities avoid considering all $(m-1)!$ symmetric optimal solutions that arise from all the permutations of the $m-1$ slowest processors. Each of these inequalities enforces the sum of the labels of the tasks assigned to processor p_k to be greater or equal than the sum of the labels of the tasks assigned to processor p_{k+1}. By transitivity, the sum of the labels of the tasks assigned to

processor p_k is greater or equal than the sum of the labels of tasks assigned to processor p_{k+j}, for any j greater than zero. The inequalities are given by:

$$\sum_{i=0}^{n-1} i \cdot y_{i,k+1} - \sum_{i=0}^{n-1} i \cdot y_{ik} \leq 0, \text{ for } k = 1, \ldots, m-2. \tag{4.9}$$

A relaxed version of this inequalities can be obtained by changing the sum of the number of the tasks by the number of tasks assigned to each processor:

$$\sum_{i=0}^{n-1} y_{i,k+1} - \sum_{i=0}^{n-1} y_{ik} \leq 0, \text{ for } k = 1, \ldots, m-2. \tag{4.10}$$

The next set of symmetry breaking inequalities is due to Mendez Díaz and Zabala [35]. It breaks ties that are accepted by inequalities (4.9) and (4.10). The tasks are labeled from 0 to $n-1$, then $j \in \{0, \ldots, n-1\}$. The processors are labeled from 1 to $m-1$.

$$y_{jk} = 0, \qquad (j,k) \in, N \times M,\ j+1 < k, \tag{4.11}$$

$$y_{jk} - \sum_{i=k-1}^{j} y_{i-1,k-1} \leq 0, \qquad (j,k) \in, N \times M,\ j+1 \geq k \geq 2. \tag{4.12}$$

The idea underlying (4.11) is that no task will be scheduled to a processor with label greater than its own label. This is true because there always exists a processor of smaller label available for processing such task. If the processors are sorted by the minimum label of the tasks assigned to each of them, then inequalities (4.11) assert that task t_k should be assigned to processor p_k or to another one with smaller label.

Inequalities (4.12) imply that a task would not be assigned to a processor p_k unless at least one of the tasks with smaller label is assigned to the processor p_{k-1}.

These inequalities were tested by including them in the basic formulation from scratch, so no separation algorithm was devised for them.

4.1.6 Separation Heuristics for some Inequalities from PLO Polytope

In this section we describe the separation algorithms for 2-partition inequalities (3.14) and double 2-chorded cycle inequalities (3.49). Both families have an exponential number of inequalities and we have not been able to devise a polynomial time algorithm to separate them. Hence, heuristics algorithms have been designed to accomplish this task.

2-Partition Separation Heuristic

Given the 2-partition inequality

$$z(\delta(S,T)) - z(A(S)) - z(A(T)) \leq \min\{|S|, |T|\},$$

the idea of this algorithm is to create a random 2-partition S_1, S_2 and then find a pair of tasks $i \in S_1$ and $j \in S_2$ such that swapping i with j the new partition increases the value of the left hand side

of the inequality (i.e. $z(\delta(S_1, S_2)) - z(A(S_1)) - z(A(S_2))$). This process is repeated until no more improvement in the value of the left hand side is made or a maximum number of iterations is reached. If the partition generated represents a violated inequality, the latter is used as a cut and the whole process is repeated until a certain number of cuts is obtained or a maximum number of iterations is reached.

The external while loop (Step 2) is repeated $O(n^2)$ times, because both values `maxStarts` and `maxTP` are $O(n^2)$. The creation of the random partition (Step 3) takes $O(n)$ time. The internal while loop (Step 4) is repeated at most $5n$ times. Generating a new pair i, j that increases the left hand side (Step 5) takes $O(n^2)$ time. Since the inner while loop takes $O(n^3)$ time, the whole algorithm takes $O(n^5)$. However, in practice, this heuristics behaves far better, because Step 5 takes $O(1)$ time in the average. The initial values of `maxTP`, `maxStarts`, and `maxIter` were determined after some computational experiments have been carried out.

Scheme of the 2-Partition Separation Heuristic

1. Initialize `numberOfCuts`=0, `maxTP`= $((n^2 - n)/28) + 5$, `starts`= 0, `maxStarts`= 2 `maxTP`, `iter`= 0, `maxIter`=$5n$.
2. **While**((`starts` < `maxStarts`) and (`numberOfCuts` < `maxTP`))
   3. Create a random partition of the task set in two sets S_1 and S_2;
   4. **While** ((there exists tasks $i \in S_1$ and $j \in S_2$ such that swapping i and j increases the value of $z(\delta(S_1, S_2)) - z(A(S_1)) - z(A(S_2))$) and (`iter` < `maxIter`))
      5. Generate a new pair $i \in S_1$ and $j \in S_2$;
      6. Increment `iter`;
   7. **If** ($z(\delta(S_1, S_2)) - z(A(S_1)) - z(A(S_2)) \geq \min\{|S_1|, |S_2|\}$); i.e. the inequality is violated, **then:**
      8. Use this inequality as cut;
      9. Increment `numberOfCuts`;
   10. Increment `starts`;

Figure 4.2: 2-Partition separation heuristics

Double 2-Chorded Cycle Separation Heuristic

Given the double 2-chorded cycle inequality

$$z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \leq \lfloor \frac{|C|}{2} \rfloor$$

the scheme of this algorithm is very similar to the previous one. The idea of this algorithm is to create a cycle C and then find a pair of tasks $i, j \in C$ such that by applying the *2-exchange operation* [30] to arcs $(i, i+1)$ and $(j, j+1)$, the newly created cycle increases the value of the left hand side (i.e. $z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2})$) of the inequality. The 2-exchange operation changes a cycle into another one by removing arcs $(i, i+1)$ and $(j, j+1)$ and placing new arcs (i, j) and $(i+1, j+1)$ such that a

new cycle is generated. As we have an oriented cycle, all arcs $(i+1, i+2), (i+2, i+3), \ldots, (j-1, j)$ must have their orientations changed during the operation. This process is repeated until no more improvement in the value of the left hand side is made or a maximum number of iterations is reached. If the cycle generated represents a violated inequality, the latter is used as a cut and the whole process is repeated until a certain number of cuts is obtained or a maximum number of iterations is reached.

The external while (Step 2) is repeated $O(n^2)$ times, because both values `maxStarts` and `maxTC` are $O(n^2)$. The creation of the random cycle (Step 3) takes $O(n)$ time. The internal while loop (Step 4) is repeated at most $5n$ times. Generating a new pair i, j that increases the left hand side (Step 5) takes $O(n^2)$ time. Since the inner while takes in fact $O(n^3)$ time, the whole algorithm is $O(n^5)$. However, in practice, this heuristics behaves far better, because Step 5 takes $O(1)$ time in average.

The initial values of `maxTC`, `maxStarts`, and `maxIter` were determined after some computational experiments.

Scheme of the Double 2-Chorded Cycle Separation Heuristic

1. Initialize `numberOfCuts`=0, `maxTC`= $((n^2-n)/28)+5$, `starts`= 0, `maxStarts`= 2 `maxTC`, `iter`= 0, `maxIter`=$5n$.
2. **While**((`starts` < `maxStarts`) and (`numberOfCuts` < `maxTC`))
   3. Create a random cycle C through the tasks of T.
   4. **While**((there exists tasks $i, i+1, j, j+1$ such that a 2-exchange operation increases the value of $z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2})$) and (`iter` < `maxIter`))
      5. Generate a new pair i, j;
      6. Increment `iter`;
   7. **If**($z(C) + z(\overline{C}) - z(C^2) - z(\overline{C^2}) \geq \lfloor \frac{|C|}{2} \rfloor$), i.e. the inequality is violated, then:
      8. Use this inequality as a cut;
      9. Increment `numberOfCuts`;
   10. Increment `starts`;

Figure 4.3: Double 2-Chorded Cycle Separation Heuristic

4.1.7 Estimation of μ_{ij}

Constants μ_{ij} in inequalities (2.20) could be overestimated as $\widehat{\mu_{ij}} = \overline{f_i} - \underline{e_j}$, where $\overline{f_i}$ is an overestimate of the latest time task t_i could finish to be processed in any optimal solution, and $\underline{e_j}$ is an underestimate of the least time at which t_j could be processed.

In this work we estimate $\overline{f_i}$ as the sum of all tasks that could be processed before t_i starts to be processed plus the processing time of t_i. This includes all tasks from P_i and Q_i. In the worst case we can suppose that in all tasks will be processed, one after the other with no delay in between, by processor p_0, whose speed allows to process PPR tasks in a unit of time, then $\overline{f_i} = (|P_i| + |R_i| + 1)/PPR$. The estimate $\underline{e_j}$ is obtained with the method described in Section 4.1.1.

4.2 Computational Experiments

The branch-and-cut code was implemented in C++ using ABACUS [44] and CPLEX [25]. ABACUS is a framework designed for the implementation of linear programming based branch-and-bound algorithms.

The preliminary experiments were performed on a SUN Sparc ULTRA1 140 MHz workstation with 288 MB of RAM, running ABACUS 2.2 and CPLEX 6.0. The final tests were performed on a 450MHz Pentium III processor with 384 MB of RAM, running ABACUS 2.3 and CPLEX 6.5. All running times reported here are given in seconds.

4.2.1 Set of Instances

Although our formulation is suitable for unrelated processors, as mentioned earlier, all computational experiments concern instances involving uniform processors with a single fastest processor p_0 and $m-1$ slower identical processors. The difference between p_0 and the other processors is given by the *Processor Power Ratio* relating their speeds: $PPR = d_{01}/d_{00}$. In our test cases PPR took values 2, 5, and 10.

The instances were selected from the *ANDES* data set [27, 28]. The instances are given by a directed acyclic graph (DAG) whose vertices represent an iterative or serial set of calculation tasks. When these calculations are completed, the generated information is transmitted to those vertices where that information is needed. Thus, arcs of the DAG represent precedence relations on the order in which the set of tasks have to be processed. Different categories of instances were considered:

1. **Diamond** (d) instances representing a systolic calculation [24].
2. **Fast Fourier transform** (f) instances representing a unidimensional fast Fourier transform [28].
3. **Gaussian elimination** (g) instances representing the Gaussian elimination algorithm used for solving systems of linear equations [3].
4. **Iterative algorithm** (i) instances representing a generic iterative algorithm [28]. After an iteration is computed, each vertex sends information to some of the vertices of the next iteration.
5. **Divide-and-conquer** (v) instances representing divide and conquer type algorithms [33].
6. **Random** (r) instances representing random acyclic digraphs.

Two sizes of instances are considered for each type. Small instances have between 14 to 25 vertices, while big instances have between 34 to 50 vertices. Some of the digraphs corresponding to these instances are depicted in Figure 4.4, where all arc orientations are top-down.

Each instance is tested with three different number of processors (2, 4, and 8) and with three different PPR values (2, 5, and 10). This gives us a total of 108 different instances for testing the branch-and-cut algorithm. Each instance is identified with the following code: $nameN.M.PPR$ where N is the number of tasks and M is the number of processors. For example, r48.4.5 denotes the random instance with 48 tasks, 4 processors, and $PPR = 5$.

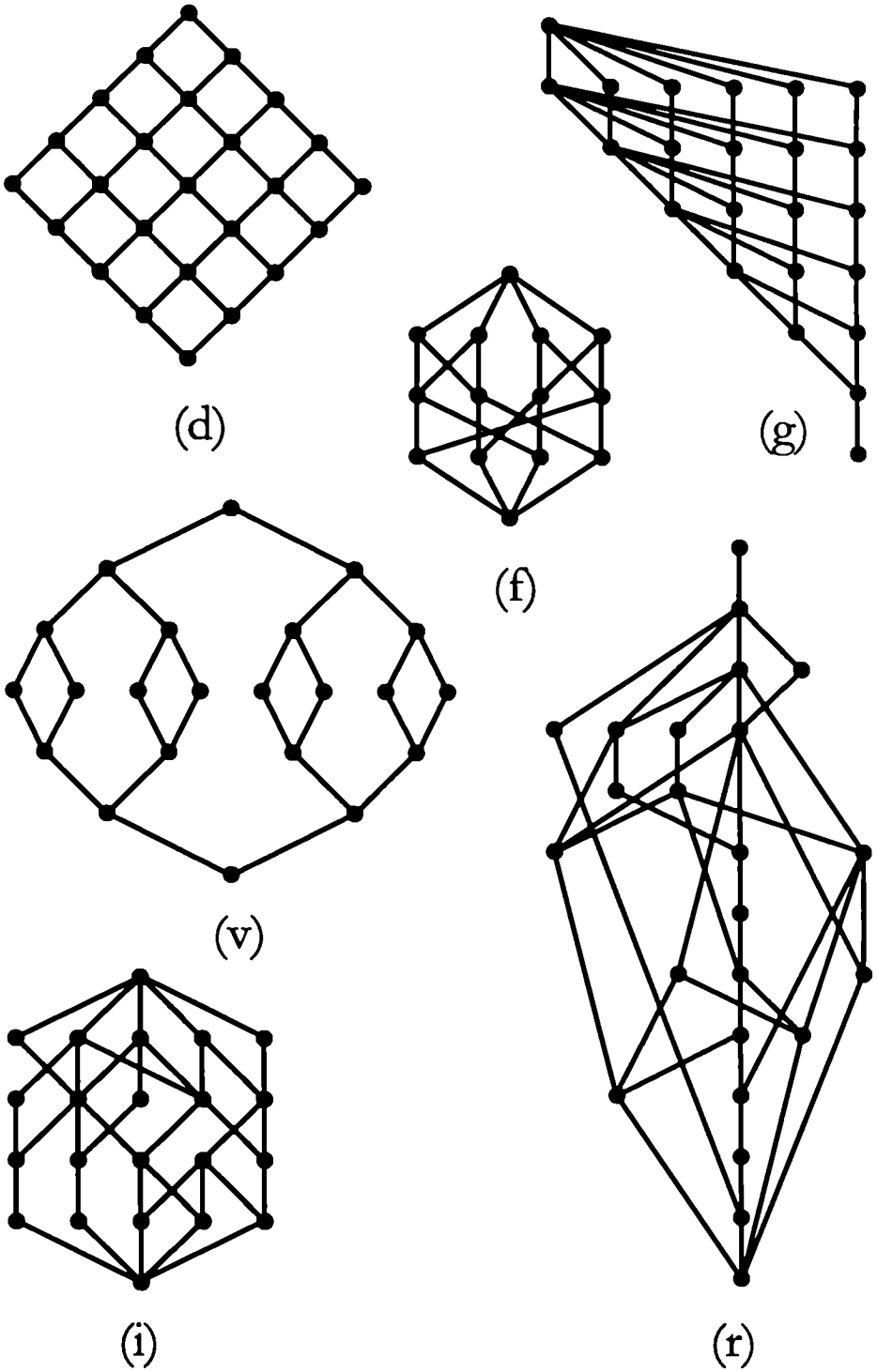

Figure 4.4: ANDES instances

4.2.2 Branch-and-Cut Parameter Setting

In this section we consider a set of ABACUS parameters which characterizes the implementation of the branch-and-cut algorithm, defining the node selection criteria (step 2), branching/cutting decision

(step 4), cut generation (step 5), and branching (step 6).

1. **Node selection (step 2)**
   The purpose is twofold when a node is selected: to find a good integer feasible solution or to prove that no solution better than the current one exists. Therefore, the quality of the current solution value of the LP relaxation is an important factor in determining which node to select for evaluation.

   Given the list of active subproblems, ABACUS parameter `EnumerationStrategy` selects which node should be examined next. We choose the ABACUS default `BestFirst`, that consists in choosing the next subproblem to explore as the one with the largest value of the objective of the LP relaxation.

   This search method will tend to minimize the number of nodes evaluated. At any point during the search it attempts to improve the global lower bound on the problem, i.e., a lower bound that is valid for all the nodes of the branch-and-cut tree. This implies that this method concentrates on proving that no solution better than the current one exists.

2. **Branching/cutting decision (step 4)**
   ABACUS provides a great variety of parameters to specify the branching/cutting decision. We have run a series of tests in order to decide the maximum level in which separation procedures will be run. After these tests (see Table 4.8) we decided to set the value of *maximal separation level* to 4.

3. **Cut generation (step 5)**

   (a) **Maximal number of added constraints**
   Parameter `MaxConAdd` determines the maximum number of constraints added to the linear relaxation per iteration in the cutting plane phase. This parameter was set to $(n^2 - n)/2$, because we observed that this is the number of inequalities from (2.14) to (2.17) that need to be added as violated cuts in the first iteration of the algorithm, see Section 4.2.3 for a discussion.

   (b) **Maximal number of iterations per cutting plane phase**
   Parameter `MaxIterations` limits the number of iterations of the cutting plane phase of a single subproblem. We choose not to limit the number of iterations by means of this parameter, using instead the *tailing off* parameter to control this number.

   (c) **Tailing off parameters**

      i. **Minimal improvement**
      When two successive linear programming relaxations in the subproblem optimization change in percent a quantity that is smaller than parameter `TailOffPercent`, then a branching step is enforced. We set parameter `TailOffPercent` to 10^{-8}.

      ii. **Number of LPs**
      The parameter `TailOffNLps` manages the relaxation of the tailing off condition imposed by the `TailOffPercent` parameter. It indicates the number of successive linear programming relaxations in the subproblem optimization differing in percent a quantity that is smaller than parameter `TailOffPercent` before a branching step is enforced, see Section 4.2.5 for an analysis of the value chosen for this parameter.

4. **Branching (step 6)**
   When we take the branching decision, we are concerned about maximizing the change of the value

of the objective function between a node of the branch-and-cut tree and its children. Several branching strategies are supported by ABACUS parameter `BranchingStrategy`. We choose to set this parameter to `CloseHalfExpensive`, selecting for branching the 0–1 variable with the closest value to 1/2.

ABACUS also supports the facility of programming our own branching strategies. We compared the `CloseHalfExpensive` strategy with another one that first applies the CloseHalfExpensive strategy to the y_{jk} variables. In case all these have integer values, this strategy is applied to the z_{ij} variables, see Section 4.2.7 for a comparative analysis of these strategies.

The initial parameter settings used to begin the computational experiments are summarized in Table 4.1.

Parameter	Value
Enumeration strategy	`BestFirst`
Branching strategy	`CloseHalfExpensive`
Max separation level	100
Max branching level	100
`TailOffNLps`	No tailing off control
`TailOffPercent`	0.00000001
`MaxConAdd`	100
Primal heuristics	tabu search [40]
Rounding heuristics	no
Breaking symmetry	no
Maximum CPU Time	900

Table 4.1: Branch-and-cut parameters

Even the best parameter tuning was not sufficient to get an efficient branch-and-cut code, because the LP that had to be solved in each node of the branch-and-bound tree was very heavy.

4.2.3 Formulation Management

Inequalities (2.14) to (2.17) are the only ones that are $O(n^2m)$ in number. They have a negative impact in the performance of the branch-and-cut algorithm, because they make solving the LP at each node of the branch-and-cut tree very heavy in terms of computational effort.

We decided to take them away from the formulation, so as that the basic formulation to be solved in every node has only $O(n^2)$ inequalities. This forced us to separate inequalities (2.14) to (2.17) at each node of the tree since otherwise, no guarantee is given that an integer solution is feasible. These inequalities were put in a cut pool provided by ABACUS and separated automatically by the ABACUS function *constraintPoolSeparation*. For each instance, we observed that $(n^2-n)/2$ of these inequalities are added at the first time the separation algorithm is performed. The ABACUS parameter `MaxConAdd` sets the maximum number of constraints added per iteration. For this reason, we decided to set this parameter to the value $(n^2-n)/2$, so as that all cutting inequalities of the families (2.14) to (2.17) can be added in the first separation round.

After we decided to work in this way, the next step was to decide which other inequalities will be used as cuts and how these cuts will be included in the cutting plane phase.

4.2.4 Cut Management

We choose a subset of nine instances of the complete set of 108 in order to represent the greatest possible number of different features (type of instance, processor power ratio, number of processors, number of tasks) and different combinations of them, to obtain a good parameter tuning. Table 4.2 displays, the nine selected instances for the preliminary tests. They are arranged in an array where instances in the same columns have the same number of processor and instances in the same row have the same value of PPR.

$PPR \backslash m$	2	4	8
2	i22.2.2	v46.4.2	d49.8.2
5	d25.2.5	r48.4.5	i50.8.5
10	g47.2.10	f14.4.10	r24.8.10

Table 4.2: Instances for branch-and-cut parameter tuning

Number	Name of the family	Ref.	Separation procedure
1	Strengthening of inequalities (2.14)	(4.8)	Optimal $O(m^2n^2)$
2	Double triangle inequalities	(3.1)	Exhaustive, $O(n^3)$
3	Double cycle inequalities	(3.2)	Exhaustive, $O(n^3)$
4	Double 2-partition inequalities	(3.14)	Heuristic, $O(n^5)$
5	Double 2-chorded cycle inequalities	(3.27)	Heuristic, $O(n^5)$
6	Successor inequalities	(4.7)	Exhaustive, $O(mn(m+n))$
7	Previously processed task inequalities	(4.5)	Exhaustive, $O(mn(m+n))$

Table 4.3: Families of cuts

The seven families of cutting planes in Table 4.3 were tested to obtain a qualitative ranking between them. The following criteria were used to rank the families of cuts in decreasing order of importance. If the first criterion is not able to establish a clear ranking between a pair of families, then the second criterion is used. If the latter is not able to make the ranking more clear, then the third criterion is applied and so on. Each test was ran for at most 900 seconds of CPU for each instance. The most powerful family is that leading to the best performance of the branch-and-cut. Each family was tested independently within the branch-and-cut scheme.

1. **Branch-and-cut algorithm.**
   A branch-and-cut scheme with the tested inequalities as cutting planes is considered. We compute the gap between the primal bound (initialized with the solution obtained by tabu search) and the global lower bound.

2. **Pure cutting plane algorithm.**
   This test was performed using a pure cutting scheme. Cuts are added at the first node of the branch-and-cut tree and no branching is performed. We compute the gap between the primal bound (initialized with the solution obtained by tabu search) and the global lower bound.

3. **Measure of the strength of the cuts.**
   Two strength measures of the cuts, each of them giving a scalar value as output, are used to rank the families of cuts. A index is defined as a weighted mean of both measures. Five values of this

index corresponding to different values of the weights are considered to build the rank between the families of cuts. The strength measures are the `AverageCutsDistance`, an ABACUS parameter that gives the average Euclidean distance between the optimum value of the LP relaxation and the added cutting planes, and the *normalized strength* a scalar given by the violation of the cutting planes divided by the norm of the left hand side of the cut.

4. **Effective cutting plane rate.**
   The ratio of inequalities that really cut the LP solution over the total number of generated inequalities is also used to rank the seven families of inequalities.

Branch-and-cut results with a single family of inequalities

This section shows the results of branch-and-cut algorithm with each of the seven families of cuts in the set of nine tested instances. The results are summarized in Figure 4.5. The heights of the bars represent the relative gap between primal bound (initialized by the tabu search result) and the global lower bound, i.e (primal bound - global lower bound) / global lower bound. Families 1 to 5 do not appear in this figure, because their effects in improving the bound obtained by the LP+δ relaxation are null. The LP+δ relaxation is the relaxation of the original formulation plus the least starting time inequalities (Section 4.1.1).

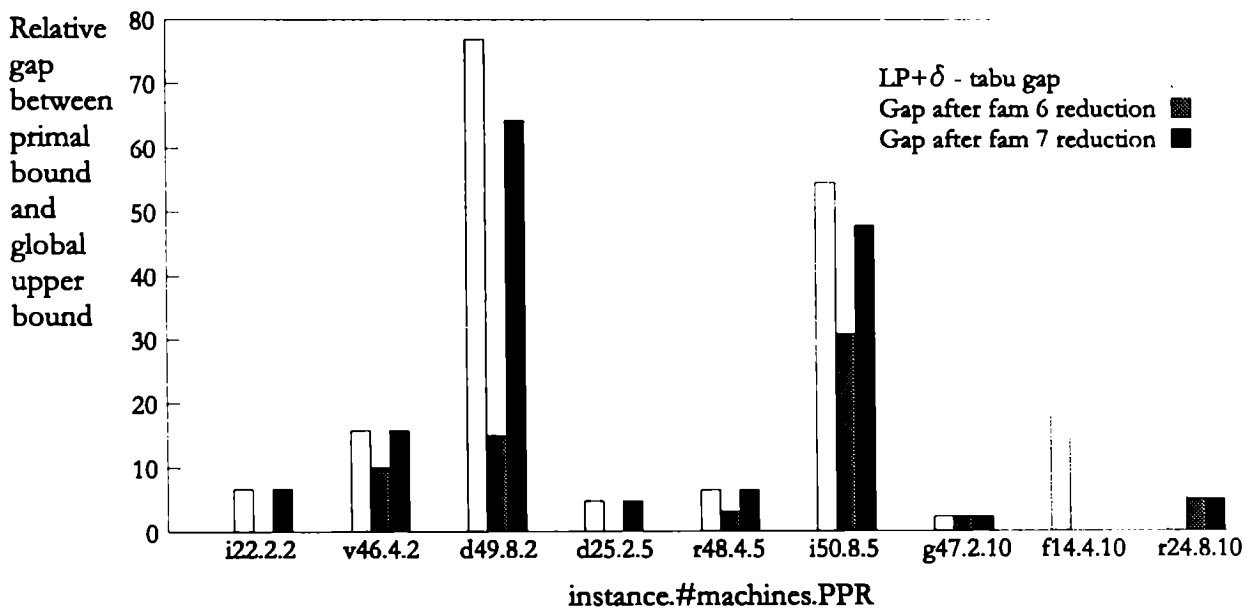

Figure 4.5: Gap reduction with branch-and-cut

These results clearly show that Family 6 is the one with the greatest influence in the final performance of the branch-and-cut algorithm. Family 7 also has a determinant role in the performance of the branch-and-cut algorithm. There is no evidence that the other five families have a significant impact in pruning the branch-and-bound tree. The single exception in which Families 1 to 5 were able to improve the bound given by the relaxation of $LP + \delta$ inequalities was instance d49.8.2, in which they contributed to rise the bound from 13 to 14.

Cutting plane results with a single family of inequalities

This section shows the results of the pure cutting plane algorithm with each of the seven families of cuts in the set of nine tested instances. The results are summarized in Figure 4.6 and also clearly show that Family 6 is the one with the greatest influence in the final performance of the pure cutting plane algorithm. Family 7 also has a determinant role in the performance of the branch-and-cut algorithm. The heights of the bars represent the relative gap between primal bound (initialized by the tabu search result) and the global lower bound, i.e (primal bound - global lower bound) / global lower bound. Families 1 to 5 do not appear in this figure, because they did not contribute to improve the bound obtained by the LP+δ relaxation.

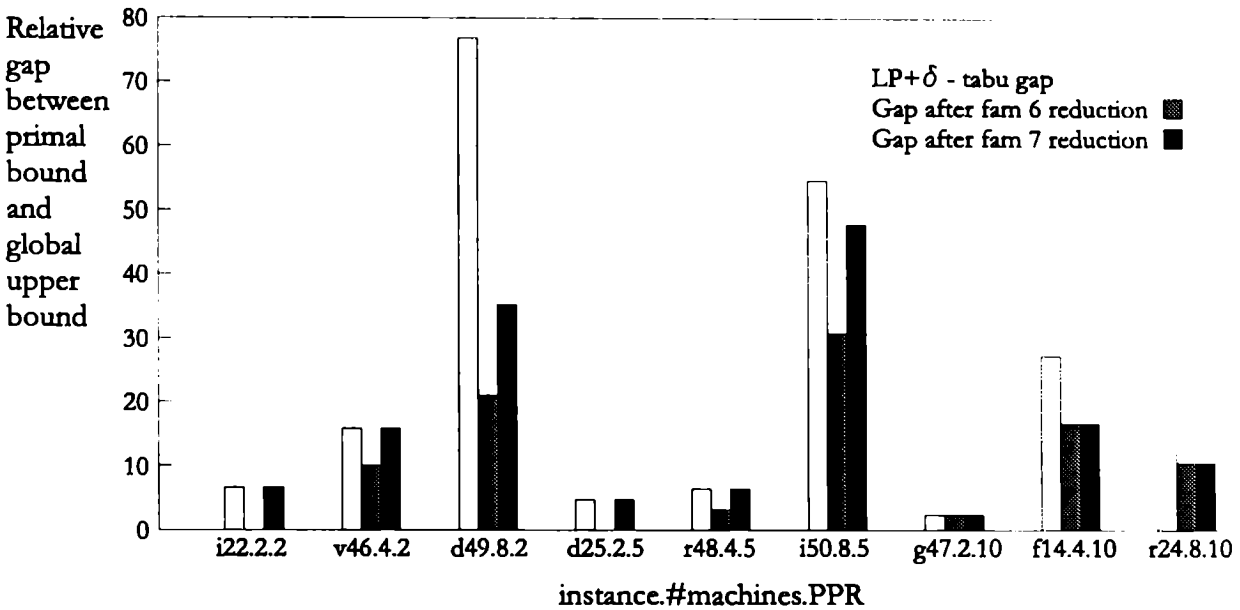

Figure 4.6: Gap reduction with pure cutting plane

No conclusive result other than the good performance of families 6 and 7 was detected.

As initially no tailing off control was used, the branch-and-cut algorithm enforced a branching step as soon as no improvement in the LP was obtained by adding new cuts. However, cuts were generated in the pure cutting plane algorithm after each LP resolution even if no improvement on its value was obtained (i.e. after the solution of a determined number of LPs with no improvement the separation algorithm can generate a cut that allows the cutting plane algorithm to obtain a better gap than that obtained by the branch-and-cut algorithm).

Average distance of cuts and normalized strength results

One of the objectives of this thesis was the design of an efficient algorithm capable of solving as big as possible instances of the problem of scheduling unrelated processors under precedence constraints. We made an attempt to rank the seven families of cuts using the results obtained with their application in the branch-and-cut and pure cutting-plane algorithms as the most important criterion. We used the measures of the strength of the cuts as a third criterion for ranking the cuts, knowing that these measures often do not reflect the algorithmic importance of the cuts.

We compare the cuts using a convex linear combination of two measures of strength.

The first is the value of the *average distance of cuts* (ACD), a functional value given by ABACUS which gives the average Euclidean distance between the optimum value of the LP relaxation and the added cutting planes. Thus the higher the values of ACD, the deeper the inequalities should cut the relaxed polytope.

The other measure is the *normalized strength* (NS). Given the inequality $\alpha x \leq \beta$ and the optimum x^* of the LP relaxation, we define $NS = (\alpha x^* - \beta)/\|\alpha\|_1$. NS is a normalized measure of the magnitude of the violation of the inequality. As for the ACD, inequalities with higher NS values tend to correspond to deeper cuts.

Table 4.4 lists the values of $a \cdot NS + (1 - a) \cdot ACD$, where a takes the values 0, 0.2, 0.5, 0.7, and 1. Column ACD corresponds to $a = 0$ and column NS corresponds to $a = 1$. Each column with a value of a is accompanied by a column with the rank of each family for this parameter value. From the discussion above, a large value of the parameter corresponds to a deep cut.

Cuts	ACD	Rank	a=0.2	Rank	a=0.5	Rank	a=0.7	Rank	NS	Rank
1	0.157	2	0.143	2	0.123	2	0.109	2	0.088	1
2	0.115	6	0.106	4	0.092	4	0.083	4	0.069	3
3	0.124	3	0.112	3	0.096	3	0.085	3	0.068	4
4	0.777	1	0.636	1	0.425	1	0.284	1	0.072	2
5	0.120	4	0.099	6	0.067	7	0.045	7	0.013	7
6	0.070	7	0.070	7	0.069	6	0.069	5	0.068	4
7	0.118	5	0.102	5	0.078	5	0.061	6	0.037	6

Table 4.4: Strength measures for cuts

This test shows that families 4 and 1 clearly prevail over the others. Family 3 is the third in the general ranking, while other families are not clearly discriminated by this test.

Effective cutting plane rate

In this section, we calculate the *effective cutting plane* rate (ECP) which is defined as the quotient between the number of the generated inequalities that effectively cut the LP relaxation solution over the total number of generated inequalities. For example, Table 4.5 shows that more than a half of the generated inequalities of Family 4 are effective cutting planes in the sense that they chop off the LP relaxation solution. On the other extreme, only one out of every 4000 inequalities are cutting planes in the case of Family 1. The last column of the table gives the rank of each family within this indicator.

Final Cut Ranking

After considering these criteria we decided to generate the families of inequalities in the following order: 6, 7, 4, 1, 3, 2, and 5. Rank between families 6 and 7 was decided with the first criterion. Rank between families 4, 1, 3, and 2 was decided with the third criterion. For all family of inequalities, we decided to generate at most cuts, where C = 1/14 `MaxConAdd` + 5 and constants 1/14 and 5 were selected for fixing scaling problems in small instances. The ABACUS parameter `MaxConAdd` sets the maximum number of constraints added per iteration (see Section 4.2.3)and was set to `MaxConAdd`=(n^2 –

Cuts	*ECP*	Rank
1	0.00025	7
2	0.00045	6
3	0.00167	5
4	0.50945	1
5	0.00832	3
6	0.02702	2
7	0.00814	4

Table 4.5: Effective cutting plane rate

$n)/2$. This implies that $C = (n^2 - n)/28 + 5$, which is a big enough value since frequently, separation algorithms do not find such a number of cuts. As families 6 and 7 have clear superior performance over the other families of inequalities, we decided to test three cut generation schemes in order to establish if it is worth the cost of generating cuts of families 1 to 5.

Cuts Generation Scheme

In this section, we test three alternatives of cutting plane generation schemes: generating cuts only from Family 6, generating cuts from families 6 and 7, and generating cuts from all the seven families. Table 4.6 gives the computation times in seconds needed to achieve the minimum gap for each of the three above mentioned schemes. The three schemes achieved the same gap within a time limit of 900 seconds of processing time.

	Cuts generation schemes		
Instance	Family 6	Families 6 and 7	All families
i22.2.2	1	1	1
d25.2.5	1	1	2
g47.2.10	32	34	74
v46.4.2	106	102	205
r48.4.5	136	117	343
f14.4.10	20	144	54
d49.8.2	114	193	260
i50.8.5	114	117	166
r24.8.10	239	121	77

Table 4.6: Computation times (secs.) to minimum gap for different cut schemes

None of the two first schemes clearly outperforms the other, while the third scheme proved to be inferior to them. Therefore, we choose the scheme using families 6 and 7 because we wanted to preserve Family 7, since it showed very good results in previous tests with branch-and-cut and pure cutting plane algorithms.

4.2.5 TailOffNLPs Analysis

ABACUS parameter `TailOffPercent` indicates the minimal change in percent of the objective function value between the solution of `TailOffNLps` sucessive linear programming relaxations in the subproblem optimization which is required such that we do not try to stop the cutting plane algorithm and to enforce a branching step.

A branching step is enforced whenever two successive linear programming relaxations in the subproblem optimization differ by a quantity that is smaller than the parameter `TailOffPercent`.

This condition can be relaxed by setting parameter `TailOffNLps` to a value greater than one. It indicates the number of successive linear programming relaxations in the subproblem optimization differing by a quantity that is smaller than parameter `TailOffPercent` before a branching step is enforced. As an example, if `TailOffNLps`=10 and LP values change less than `TailOffPercent`, then the algorithm will solve 10 LPs in a node of the branch-and-cut tree before going into the branching step.

We analyze the effect of the parameter `TailingOffNLPs` in the performance of the branch-and-cut algorithm. Five values of parameter `TailingOffNLPs` were tested: 2, 4, 8, 16, and unlimited (U). Only two instances of the unlimited case failed to achieve the minimum gap found for that instance with other values of the parameter. The results of this case are doubtless worse than those obtained with the other values of the parameter.

Table 4.7 shows the computation times, in seconds, needed to achieve the minimum gap for each test instance with the five different values of the parameter. All choices of the `TailingOffNLPs` parameter achieved the same gap within a time limit of 900 seconds of processing time except those that appears in the tables with ***. The value of the parameter `TailOffPercent` was fixed in 10^{-8}.

	Number of tailing off LPs				
Instance	2	4	8	16	U
i22.2.2	5	1	1	1	1
d25.2.5	7	2	3	2	2
g47.2.10	22	71	160	281	948
v46.4.2	617	254	228	437	902
r48.4.5	644	341	358	608	922
f14.4.10	95	300	146	189	***
d49.8.2	724	1013	1014	1017	1004
i50.8.5	231	290	536	1005	998
r24.8.10	11	15	24	44	***

Table 4.7: Computation times (secs.) to minimum gap for different `TailingOffNLPs` values

Although there is not a single value of the parameter for which the algorithm is faster for every instance in the table, we used `TailingOffNLPs` = 4.

4.2.6 Separation Level

We define the *separation level* as the maximum level at which separation is performed. In this section, we test five different values for the separation level. In Table 4.8 we show the computation

times in seconds needed to achieve the minimum gap for each test instance with five different values of this parameter. All choices of the separation level achieved the same gap within a time limit of 900 seconds of processing time except those that appears in the tables with ***.

	Separation level values				
Instance	2	4	8	16	99
i22.2.2	1	1	1	1	1
d25.2.5	1	1	1	1	1
g47.2.10	36	35	35	35	35
v46.4.2	102	105	102	102	102
r48.4.5	121	120	121	120	121
f14.4.10	34	85	54	68	77
d49.8.2	200	200	199	199	199
i50.8.5	121	120	126	120	121
r24.8.10	155	135	***	650	816

Table 4.8: Computation times (secs.) to minimum gap for different separation level values

Although there is not a single value of the parameter for which the algorithm is faster for every instance in the table, we set the maximum separation level to four.

4.2.7 Branching Strategies

ABACUS provides functionalities for the implementation of different branching strategies. We tested ABACUS `closeHalf` (CH) branching strategy that selects the variable with fractional value closest to 0.5 versus our own branching strategy *Y Close Half* (YCH). The latter consists of selecting among the set of y variables with fractional value, the one with value closest to 0.5. If all of them have integer values, then we choose the z variable with value closest to 0.5. We privileged branching over the y variables because we understand that they have a clear meaning in the formulation as assignment variables, whereas we regarded the z variables as auxiliar and thus less important ones. Table 4.9 displays the results of the tests for the nine instances. The computation times in seconds needed to achieve the minimum gap for each instance and the number of subproblems of the branch-and-cut tree are displayed. The branch-and-cut code with our YCH branching strategy never took longer than the same algorithm with ABACUS `CloseHalf` strategy, even in the cases in which the former solved slightly more subproblems than the latter. These results leads to the choice of the YCH strategy instead of the standard `closeHalf` strategy.

4.2.8 Symmetry Breaking

The following tests compare the behavior of the branch-and-cut using the original model (ORIG) against its extension (SYMM) using the symmetry breaking inequalities (4.11) and (4.12). Table 4.10 shows that there is no clear difference regarding the results obtained by both approaches. For each instance and for each approach, we give the computation times in seconds needed to achieve the minimum gap for each instance, the number of subproblems in the branch-and-cut tree, and the total number of cuts added during the execution of the algorithm.

The resulting polytope obtained after adding the symmetry breaking inequalities is indeed smaller than the original polytope. For any feasible solution, $(m-1)!$ new solutions are obtained by permuting

Instance	Times (secs.) CH	YCH	Subproblems CH	YCH
i22.2.2	1	1	1	1
d25.2.5	1	1	1	1
g47.2.10	35	35	269	271
v46.4.2	100	100	189	193
r48.4.5	120	114	555	556
f14.4.10	84	3	731	21
d49.8.2	197	195	85	89
i50.8.5	122	116	45	65
r24.8.10	135	67	161	33

Table 4.9: Computation times (secs.) to minimum gap for different branching strategies

Instance	Times (secs.) ORIG	SYMM	Subproblems ORIG	SYMM	Cuts ORIG	SYMM
v46.4.2	101	125	193	201	1764	1597
r48.4.5	119	145	183	181	1616	1911
f14.4.10	84	67	731	525	208	215
d49.8.2	197	277	85	69	2759	2210
i50.8.5	119	116	45	37	2364	2453
r24.8.10	133	38	161	13	420	382

Table 4.10: Computation times (secs.) to minimum gap for original model and using symmetry breaking

the tasks assigned to the $m-1$ slowest processors (see Section 4.1.5). All these solutions are also feasible solutions of the original polytope. However, only one of the $(m-1)!$ symmetric permutations is kept in the new polytope with the addition of the symmetry breaking inequalities. We believed that this fact would result in a reduction of the processing times. However, since the structure of the new polytope is extremely more complex than that of the original one, this property did not lead to smaller computation times. Since the effectiveness of the symmetry breaking inequalities was not clear, we decided not to include them in the algorithm.

4.2.9 Primal Heuristics

A LP-based heuristics for the problem was also developed. Preliminary experiments confirmed that the branch-and-bound algorithm with the new formulation and even the branch-and-cut algorithm have already a great difficulty to find feasible solutions. The goal of this heuristics is to make good use of the information hidden in the LP relaxation of the problem to build a feasible solution. It was ran in each node of the branch-and-cut algorithm, after each LP relaxation was solved. The basic idea is to infer an ordering of the tasks from the solution of the LP relaxation. The algorithm is described in Figure 4.7. Three different criteria (a), (b) and (c) for processor selection (step 3) were tested separately, resulting in three different rounding heuristics.

The main information of the LP used by the heuristics are the starting times of all tasks. The tasks are ordered by the values of $e_j \cdot n \cdot PPR + j$. This value is defined so as that no ties are possible.

Scheme of the rounding heuristics

1. **Initialization:** Initialize a heap with the tasks that are ready to be processed.
2. **Task selection:** If the heap is empty stop, otherwise select the task *taskS* in the root of the heap.
3. **Processor selection:** Given the task selected in step 2, select the processor *processorS* according to one of the three criteria (a), (b) or (c).
4. **Update the heap:** Calculate the *start time* of task *taskS* in processor *processorS*. Update the heap with the new tasks that can be released because all its predecessors have been already processed. Use as index to order the heap the value $e_j \cdot n \cdot PPR + j$. Go to step 2.

Processor selection criteria

(a) Such that the release time of *taskS* is minimal.

(b) With the greatest value of variable $y(taskS, processorS)$.

(c) At random.

Figure 4.7: Rounding heuristics

Table 4.11 displays a comparison between these three heuristics and the tabu search algorithm of Porto and Ribeiro [40]. For each instance and each heuristics ((a), (b), (c) and tabu search), we give the best solution value and the time in seconds needed to obtain the best solution. Results for instance f14.4.10 are not displayed because the branch-and-cut algorithm found the optimal integer solutions in the root node. The first LP solved in the root node of the branch-and-bound tree for the instance r24.8.10 was integer feasible and heuristics (a) was not able to further improve the corresponding value.

Instance	Heur	Value	Time (secs.)
i22.2.2	(a)	16	0.8
	(b)	16	25.8
	(c)	16	14.1
	tabu	16	2.1
d25.2.5	(a)	22	182.7
	(b)	22	230.7
	(c)	22	233.7
	tabu	22	3.1
g47.2.10	(a)	44	995.3
	(b)	44	1906.6
	(c)	45	78.2
	tabu	44	14.8
v46.4.2	(a)	22	35.2
	(b)	23	1866.7
	(c)	23	653.7
	tabu	22	33.9

Instance	Heur	Value	Time (secs.)
r48.4.5	(a)	39	988.0
	(b)	36	5681.3
	(c)	35	1446.5
	tabu	34	52.7
d49.8.2	(a)	24	9.0
	(b)	32	2382.6
	(c)	30	2283.5
	tabu	23	85.5
i50.8.5	(a)	39	1870.3
	(b)	36	450.0
	(c)	37	9157.0
	tabu	34	207.3
r24.8.10	(a)	24	-
	(b)	20	1219.0
	(c)	21	58.8
	tabu	21	34.4

Table 4.11: Results obtained by the rounding heuristics

In four out of the eight tested instances (i22.2.2, d25.2.5, g47.2.10, v46.4.2), at least one of the three rounding heuristics found a feasible solution with the same value of that obtained by the tabu search heuristics of Porto and Ribeiro [40]. In three out of the other four instances (r48.4.5, d49.8.2, i50.8.5), at least one of the rounding heuristics found a solution exceeding by one or two units the tabu searh solution. Finally, for the last instance (r24.8.10), heuristics (b) improved the tabu search by one unit. However, the computation times of the rounding heuristics are much larger than those of tabu search. In the next section, we discuss further instances for which the rounding heuristics found better solutions than tabu search.

4.2.10 Branch-and-Cut versus Tabu Search

The branch-and-cut algorithm found better solutions than those obtained by tabu search for six instances in our dataset. Table 4.12 summarizes these results.

	Solution values	
Instance	Tabu	B&C
f34.8.5	22	21
r24.8.10	21	20
v22.8.5	16	15
d25.4.5	21	20
d25.8.5	21	20
v46.8.2	18	16

Table 4.12: Instances for which branch-and-cut found a better solution than tabu search

In all but one case, the solutions that improved those obtained by tabu search resulted from the rounding heuristics. In only one case (r24.8.10), the branch-and-cut algorithm found an integer feasible solution better than that obtained by tabu search without making use of the rounding heuristics. The fact that even small instances were not solved to optimality by tabu search confirms the hardness of the problem of scheduling unrelated processors under precedence contraints.

4.3 Final Results

4.3.1 Branch-and-Cut

We report in this section the final results for all the 108 instances. The parameter values and implementation criteria used in our branch-and-cut algorithm are displayed in Table 4.13.

The results are summarized in Tables 4.14 to 4.16. Column A displays the value of the linear programming relaxation of the new formulation. Column B displays the value of the relaxation of the original formulation with the least starting time inequalities. Column C displays the best lower bound obtained by the branch-and-cut algorithm. Column D displays the value of the best feasible solution obtained. The tabu search algorithm obtained 102 of the best solutions out of the 108 instances (94.44%), while the best solutions for the remaining 6 instances (5.56%) were obtained by the branch-and-cut algorithm, as reported in the previous section. Columns $G1$, $G2$ and $G3$ display the relative gaps in percent between the best feasible solution (column D) and the lower bounds given by columns A, B, and C. They are computed as follows: $G1 = (D - A) \cdot 100/A$, $G2 = (D - B) \cdot 100/B$, and $G3 =$

Parameter	Value
Enumeration strategy	best first
Branching strategy	close half first on y variables
Maximum separation level	4
Separation scheme	families 6 and 7 of cuts
Maximum branching level	100
`TailOffNLps`	4
`TailOffPercent`	0.00000001
`MaxConAdd`	$(n^2 - n)/2$
Primal heuristics	tabu search [40]
Rounding heuristics	(a) minimal release time of tasks
Symmetry breaking	no
Maximum computation time	3600 seconds

Table 4.13: Branch-and-cut parameters

$(D-C)\cdot 100/C$. Column $H1 = (B-A)\cdot 100/A$ in Tables 4.14 to 4.16 shows the impressive improvement on the LP relaxation value given by the simple introduction of least starting time inequalities (4.1) in the formulation. The introduction of these inequalities in the LP relaxation allowed to prove the optimality of the best feasible solutions for 25 out of the 108 instances.

The best behavior of these inequalities is observed with a small number of processors (all the 25 instances whose gaps were closed to zero had two or four processors) and high PPR values (all but three of the 22 instances whose gaps were closed to zero had $PPR = 5$ or $PPR = 10$).

On the other hand, only six instances did not experience improvements in their LP relaxation values with the introduction of these inequalities. All of them were instances with eight processors and $PPR = 2$. These results show that the performance of inequalities (4.1) become impoverished with the increase in the number of processors and the decrease of PPR.

Also important is the improvement obtained by the branch-and-cut algorithm for the bound given by the value of the LP relaxation with inequalities (4.1). This is ilustrated by the value of the last column $H2 = (C - B) \cdot 100/B$ of Tables 4.14 to 4.16. We notice that non null gaps are observed for 83 out of the 108 instances after running the LP relaxation with inequalities (4.1).

The branch-and-cut algorithm improved 57 of the global lower bounds, including 22 instances in which the gap was reduced to zero. Only 10 instances remain with the gap G3 greater than 20%. The bound given by the LP relaxation was improved for all of the 108 instances.

Feasible solutions for 47 instances were proved to be optimal. Tighter bounds were obtained for the other instances.

4.3.2 Why to use branch-and-cut?

The goal of this section is to answer the following practical question. A practitioner has to solve the problem of scheduling unrelated machines under precedence constraints and he has at his disposal the following facilities: one of the best integer linear programming solvers in the market (e.g. CPLEX 7.0) and a fast machine where this solver can be run (e.g. a 675 MHz Alpha/DEC workstation with 4GB of RAM under UNIX Thru64). Why should he use the machinery exposed in this thesis rather

Instance	A	B	C	D	$G1$ (%)	$G2$ (%)	$G3$ (%)	$H1$ (%)	$H2$ (%)
v46.8.2	9	11	13	18	100.00	63.64	38.46	22.22	18.18
f34.8.5	6	16	16	22	266.67	37.50	37.50	166.67	0.00
r48.8.5	16	22	25	33	106.25	50.00	32.00	37.50	13.64
i50.8.5	10	22	26	34	240.00	54.55	30.77	120.00	18.18
f34.8.10	6	21	21	27	350.00	28.57	28.57	250.00	0.00
v46.8.5	9	21	21	27	200.00	28.57	28.57	133.33	0.00
d49.8.5	13	22	29	36	176.92	63.64	24.14	69.23	31.82
v22.8.5	7	11	13	16	128.57	45.45	23.08	57.14	18.18
i50.8.10	10	31	35	43	330.00	38.71	22.86	210.00	12.90
d49.8.2	13	13	19	23	76.92	76.92	21.05	0.00	46.15
v22.4.2	7	10	10	12	71.43	20.00	20.00	42.86	0.00
d49.8.10	13	31	36	43	230.77	38.71	19.44	138.46	16.13
g47.8.5	11	22	26	31	181.82	40.91	19.23	100.00	18.18
d49.4.2	13	21	21	25	92.31	19.05	19.05	61.54	0.00
f14.8.5	5	7	11	13	160.00	85.71	18.18	40.00	57.14
f14.4.5	5	11	11	13	160.00	18.18	18.18	120.00	0.00
d25.8.5	9	12	18	21	133.33	75.00	16.67	33.33	50.00
d25.4.5	9	16	18	21	133.33	31.25	16.67	77.78	12.50
r48.8.10	16	31	31	36	125.00	16.13	16.13	93.75	0.00
d25.8.2	9	9	13	15	66.67	66.67	15.38	0.00	44.44
d25.4.2	9	11	13	15	66.67	36.36	15.38	22.22	18.18
g47.8.10	11	32	33	38	245.45	18.75	15.15	190.91	3.13
g47.4.2	11	20	20	23	109.09	15.00	15.00	81.82	0.00
f14.8.2	5	5	7	8	60.00	60.00	14.29	0.00	40.00
i22.8.5	6	11	14	16	166.67	45.45	14.29	83.33	27.27
i50.8.2	10	13	16	18	80.00	38.46	12.50	30.00	23.08
d49.4.5	13	31	32	36	176.92	16.13	12.50	138.46	3.23
g47.8.2	11	12	17	19	72.73	58.33	11.76	9.09	41.67
v22.8.10	7	15	17	19	171.43	26.67	11.76	114.29	13.33
i22.8.2	6	7	9	10	66.67	42.86	11.11	16.67	28.57
r48.8.2	16	16	19	21	31.25	31.25	10.53	0.00	18.75
v22.8.2	7	7	10	11	57.14	57.14	10.00	0.00	42.86
v46.4.2	9	19	20	22	144.44	15.79	10.00	111.11	5.26
d25.8.10	9	18	22	24	166.67	33.33	9.09	100.00	22.22
d25.4.10	9	21	22	24	166.67	14.29	9.09	133.33	4.76
g23.8.2	8	8	12	13	62.50	62.50	8.33	0.00	50.00

Table 4.14: Final results (1/3)

than write a good model and run it with CPLEX 7.0 in the 675 MHz Alpha/DEC workstation?

Preliminary tests with both machines applying the CPLEX solver to the new formulation for the same set of instances showed that the Alpha / DEC machine behaved approximately 3.06 times faster than the 450 MHz Pentium III with 384 MB of RAM under Linux using CPLEX 6.5, where the branch-and-cut algorithm was run. Every instance was run for at most 1800 seconds of processing time for each algorithm–processor configuration. Branch-and-cut was run with no primal bound information, but with the rounding heuristics activated. In order to speed up the branch-and-cut algorithm, inequalities

Instance	A	B	C	D	$G1$ (%)	$G2$ (%)	$G3$ (%)	$H1$ (%)	$H2$ (%)
g23.4.2	8	11	12	13	62.50	18.18	8.33	37.50	9.09
f14.2.5	5	12	12	13	160.00	8.33	8.33	140.00	0.00
d49.4.10	13	40	40	43	230.77	7.50	7.50	207.69	0.00
f34.4.2	6	15	15	16	166.67	6.67	6.67	150.00	0.00
v46.8.10	9	31	31	33	266.67	6.45	6.45	244.44	0.00
i50.4.5	10	32	32	34	240.00	6.25	6.25	220.00	0.00
g23.2.2	8	16	16	17	112.50	6.25	6.25	100.00	0.00
r24.8.5	13	15	17	18	38.46	20.00	5.88	15.38	13.33
r24.4.5	13	15	17	18	38.46	20.00	5.88	15.38	13.33
g23.4.5	8	17	17	18	125.00	5.88	5.88	112.50	0.00
i22.4.10	6	19	19	20	233.33	5.26	5.26	216.67	0.00
r24.8.10	13	15	20	21	61.54	40.00	5.00	15.38	33.33
g23.8.10	8	16	20	21	162.50	31.25	5.00	100.00	25.00
g23.4.10	8	20	20	21	162.50	5.00	5.00	150.00	0.00
i50.4.10	10	41	41	43	330.00	4.88	4.88	310.00	0.00
r48.4.2	16	21	22	23	43.75	9.52	4.55	31.25	4.76
d25.2.10	9	23	23	24	166.67	4.35	4.35	155.56	0.00
r48.4.5	16	31	32	33	106.25	6.45	3.13	93.75	3.23
g47.2.2	11	32	32	33	200.00	3.13	3.13	190.91	0.00
g47.4.5	11	32	32	33	200.00	3.13	3.13	190.91	0.00
d49.2.2	13	33	33	34	161.54	3.03	3.03	153.85	0.00
g47.4.10	11	38	38	39	254.55	2.63	2.63	245.45	0.00
d49.2.5	13	41	41	42	223.08	2.44	2.44	215.38	0.00
g47.2.10	11	43	43	44	300.00	2.33	2.33	290.91	0.00
r48.2.10	16	44	44	45	181.25	2.27	2.27	175.00	0.00
i22.8.10	6	15	20	20	233.33	33.33	0.00	150.00	33.33
g23.8.5	8	12	16	16	100.00	33.33	0.00	50.00	33.33
f14.4.10	5	11	14	14	180.00	27.27	0.00	120.00	27.27
f14.8.10	5	11	14	14	180.00	27.27	0.00	120.00	27.27
r24.2.10	13	20	23	23	76.92	15.00	0.00	53.85	15.00
f14.4.2	5	7	8	8	60.00	14.29	0.00	40.00	14.29
r24.2.2	13	15	17	17	30.77	13.33	0.00	15.38	13.33
f34.8.2	6	9	10	10	66.67	11.11	0.00	50.00	11.11
r24.2.5	13	19	21	21	61.54	10.53	0.00	46.15	10.53
r24.4.10	13	19	21	21	61.54	10.53	0.00	46.15	10.53
f14.2.10	5	13	14	14	180.00	7.69	0.00	160.00	7.69

Table 4.15: Final results (2/3)

(2.14) to (2.17) were not placed in the initial formulation and were used instead as cuts.

The first column in of Table 4.17 gives the gap between the best feasible solution and the global lower bound obtained by the branch-and-cut algorithm (using the primal bound information given by the tabu search algorithm, column G3 of Tables 4.14 to 4.16). The next column gives the best known feasible solution, obtained in general by the tabu search heuristics. For both the branch-and-cut and the branch-and-bound algorithms, the next columns display the time needed to obtain the best gap, the number of nodes in the enumeration tree, the gap (UB-LB)/UB, the global lower bound and the

Instance	A	B	C	D	$G1$ (%)	$G2$ (%)	$G3$ (%)	$H1$ (%)	$H2$ (%)
r24.4.2	13	13	14	14	7.69	7.69	0.00	0.00	7.69
r24.8.2	13	13	14	14	7.69	7.69	0.00	0.00	7.69
i22.2.2	6	15	16	16	166.67	6.67	0.00	150.00	6.67
v22.2.2	7	15	16	16	128.57	6.67	0.00	114.29	6.67
d25.2.2	9	17	18	18	100.00	5.88	0.00	88.89	5.88
d25.2.5	9	21	22	22	144.44	4.76	0.00	133.33	4.76
i50.4.2	10	21	22	22	120.00	4.76	0.00	110.00	4.76
f34.2.2	6	23	24	24	300.00	4.35	0.00	283.33	4.35
f34.2.10	6	31	32	32	433.33	3.23	0.00	416.67	3.23
v46.2.2	9	31	32	32	255.56	3.23	0.00	244.44	3.23
r48.2.2	16	33	34	34	112.50	3.03	0.00	106.25	3.03
f34.2.5	6	29	29	29	383.33	0.00	0.00	383.33	0.00
f34.4.10	6	28	28	28	366.67	0.00	0.00	366.67	0.00
v46.2.10	9	42	42	42	366.67	0.00	0.00	366.67	0.00
i50.2.10	10	46	46	46	360.00	0.00	0.00	360.00	0.00
v46.2.5	9	39	39	39	333.33	0.00	0.00	333.33	0.00
i50.2.5	10	42	42	42	320.00	0.00	0.00	320.00	0.00
v46.4.10	9	37	37	37	311.11	0.00	0.00	311.11	0.00
f34.4.5	6	22	22	22	266.67	0.00	0.00	266.67	0.00
g47.2.5	11	40	40	40	263.64	0.00	0.00	263.64	0.00
i22.2.10	6	21	21	21	250.00	0.00	0.00	250.00	0.00
d49.2.10	13	45	45	45	246.15	0.00	0.00	246.15	0.00
v46.4.5	9	31	31	31	244.44	0.00	0.00	244.44	0.00
i50.2.2	10	34	34	34	240.00	0.00	0.00	240.00	0.00
i22.2.5	6	19	19	19	216.67	0.00	0.00	216.67	0.00
v22.2.10	7	21	21	21	200.00	0.00	0.00	200.00	0.00
g23.2.10	8	22	22	22	175.00	0.00	0.00	175.00	0.00
v22.2.5	7	19	19	19	171.43	0.00	0.00	171.43	0.00
v22.4.10	7	19	19	19	171.43	0.00	0.00	171.43	0.00
i22.4.5	6	16	16	16	166.67	0.00	0.00	166.67	0.00
r48.2.5	16	41	41	41	156.25	0.00	0.00	156.25	0.00
g23.2.5	8	20	20	20	150.00	0.00	0.00	150.00	0.00
r48.4.10	16	39	39	39	143.75	0.00	0.00	143.75	0.00
v22.4.5	7	16	16	16	128.57	0.00	0.00	128.57	0.00
f14.2.2	5	10	10	10	100.00	0.00	0.00	100.00	0.00
i22.4.2	6	10	10	10	66.67	0.00	0.00	66.67	0.00

Table 4.16: Final results (3/3)

upper bound (i.e. the best feasible solution) obtained by the algorithms.

The difference between the number of nodes opened in the branch-and-bound tree and the number of nodes opened in the branch-and-cut tree is remarkable. Instances i22.2.2 and d25.2.5 are particularly noticeable and illustrate the strength of the cuts. Instance g47.2.10 is a good example of the difficulties found by branch-and-bound to obtain an integer feasible solution. No integer feasible solution was found after going through 12874 nodes. Even when an integer feasible solution was found, it was in general worse than the one found by the rounding heuristics of the branch-and-cut algorithm.

Instance	Best gap	Best feasible	Branch-and-bound Time	Nodes	Gap (%)	LB	UB	Branch-and-cut Time	Nodes	Gap (%)	LB	UB
f14.4.10	0.00	14	73.83	38103	0.00	14	14	18.27	599	0.00	14	14
i22.2.2	0.00	16	408.80	665292	6.25	15	16	0.60	1	0.00	16	16
r24.8.10	0.00	20	20.73	3724	22.73	17	22	1020.62	7259	185.00	20	57
d25.2.5	0.00	22	19.35	669493	4.55	21	22	5.65	29	0.00	22	22
i50.8.5	30.77	34	1800.00	15	∞	22	?	1694.00	193	50.00	26	39
v46.4.2	10.00	22	277.59	1031	54.76	19	42	43.00	1109	10.00	20	22
g47.2.10	2.33	44	1800.00	12874	∞	43	?	75.00	1233	2.33	43	44
r48.4.5	3.13	33	1671.10	391	35.42	31	48	526.00	1027	28.13	32	41
d49.8.2	21.05	23	1800.00	0	∞	14.03	?	103.00	153	26.32	19	24

Table 4.17: Branch-and-cut versus branch-and-bound (times in seconds)

The results of these experiments illustrate of the effectiveness of the branch-and-cut algorithm for the problem of scheduling unrelated processors under precedence constraints. In the next chapter, conclusions and future lines of research are discussed.

Chapter 5

Conclusions and Future Research

In this thesis we proposed a new polyhedral formulation for the problem of scheduling unrelated processors under precedence constraints. This formulation lead to new theoretical and computational results.

The size of the new formulation had a direct impact on the size of the problems that could be solved. First, because the reduction in the number of 0-1 variables clearly reduces the size of the enumeration tree. Second, because the reduction in the number of constraints and nonzero elements in the constraints accelerates the solution of the LP relaxation and therefore the processing time at each node of the branch-and-cut tree.

A subset of the inequalities in this model defines a linear ordering for the set of tasks that is assigned to each processor. This subset of inequalities defines the problem of partitioning a directed graph in linear orderings (PLO). The polytope of partition in linear orderings has shown to have a very similar structure to the clique partitioning polytope. We believe that the effect of the PLO cuts in our branch-and-cut algorithm was poor due to the extremely complex structure of the problem. However, we are convinced that the PLO polytope should play an important role in the formulation of multiple machine scheduling problems and in their solution.

Scheduling problems with many different families of constraints lead to very complex polyhedra, where standard heuristics or simple enumeration procedures often do not succeed in finding a feasible solution. Therefore, primal heuristics play a very important role in successful problem solving. The results obtained by our rounding heuristics ilustrate this conclusion.

Great improvements were achieved in the reduction of the gap between the best feasible solution and the global lower bound. In all tests the best feasible solution was initialized with the result of the tabu search heuristics [40]. The lower bounds were improved for all instances. In 5.6% of the cases, this improvement was also a consequence of improved feasible solutions found by the branch-and-cut algorithm.

Proved optimal solutions were found for the first time for many instances of this problem. These proofs were attained by closing the gap between the best feasible solution and the global lower bound. The LP relaxation of the formulation with inequalities (4.1) lead to the proof of optimality for the solutions obtained for 25 out of the 108 instances used in the final experiments. The branch-and-cut algorithm using the families 6 and 7 of cuts was able to raise the lower bound and gave the guarantee of optimality for other 22 instances.

The gap between the best feasible solution after running the LP relaxation and the global lower bound was larger than 30% for 106 out of the 108 instances (larger than 100% for 78 of them). After branch-and-cut, only ten of the 108 instances remained with gaps bigger than 20%.

Even though it is clear from our computational experiments that we are still at a significant distance from solving to optimality large real world instances of the problem of scheduling unrelated processors under precedence constraints with our branch-and-cut code, a great progress in establishing good bounds was made. These bounds have a great importance to prove the optimality or to assess the quality of the feasible solutions found by tabu search heuristics or by the branch-and-cut algorithm.

Several results in this thesis may lead to further developments:

Relation between PLO and other polytopes The clique partitioning problem can be interpreted as a non-oriented version of the PLO problem, because the clique partitioning problem seeks for partitions in cliques, that can be regarded as non-oriented versions of the linear orderings. Thus both polytopes have many similarities, i.e. given a correct orientation, some facets of the clique partitioning polytope are also facets of the PLO polytope. This fact make us think that a much stronger relation could exist between both polytopes. Other related polytopes could also have a more general underlying common structure, with consequences in the understanding of multiple machine scheduling polyhedra.

LP-based heuristics There are several examples of machine scheduling problems for which effective LP-based heuristics have been developed. The basic idea on which they are based consists on infering an ordering of the tasks from the solution of the LP relaxation of the problem. Recently, there has been theoretical as well as empirical evidence that LP-based heuristics are among the best available approximation algorithms for machine scheduling [26]. The good results obtained by our naïve rounding heuristics stress our belief that the formulation takes into account important information on the structure of the problem and that more intelligent rounding heuristics could be developed.

New valid inequalities and facets of PLO Computational experiments were run with ZeroOne [31], a vertex enumeration software for 0-1 polytopes, and PORTA [8] (POlyhedron Representation Transformation Algorithm). The number of vertices and facets for low order PLO polytopes are summarized in Table 5.1.

PLO	Vertices	Facets
$P_{PLO}(D_3)$	13	11
$P_{PLO}(D_4)$	73	96
$P_{PLO}(D_5)$	501	> 25000
$P_{PLO}(D_6)$	4051	?
$P_{PLO}(D_7)$	37633	?

Table 5.1: Vertices and facets of PLO

This table shows that the PLO polytope is far more complex in structure than the linear ordering polytope [9]. The inequality:

$$6z_{1,2} + 6z_{1,3} - 4z_{2,3} - 5z_{1,4} + 4z_{2,4} + 4z_{3,4} - 5z_{1,5} + 4z_{2,5} + 4z_{3,5} - 3z_{4,5}$$
$$-8z_{2,1} - 8z_{3,1} - 4z_{3,2} + 7z_{4,1} - 6z_{4,2} - 6z_{4,3} + 7z_{5,1} - 6z_{5,2} - 6z_{5,3} - 3z_{5,4} \leq 11$$

was proved to be a facet of $P_{PLO}(D_5)$ using PORTA [8]. Thus, by Theorem 3.19 it is also a facet of $P_{PLO}(D_n)$ for all $n > 5$.

The following inequalities for $P^{y_m}_{PLO}(D_n)$ were found with the aid of ZerOne and PORTA. Given the formulation, we used ZerOne to obtain all the 0-1 vertices of the polytope defined by the convex hull of the integer points included in such formulation. We next used PORTA with all vertices of the polytope to find all the facets.

$$\sum_{i \neq j} z_{ij} \geq h^2, \ \forall (i,j) \in N^2, i \neq j \tag{5.1}$$

$$\sum_{i \neq j} z_{ij} \geq (h-1)^2, \ \forall (i,j) \in N^2, i \neq j \tag{5.2}$$

$$\sum_{i \neq j} z_{ij} + (n-3) \sum_{i=1}^{n} \sum_{h \in M \setminus \{k\}} y_{ih} \geq \frac{1}{2} n(n-1) - 2, \ \forall (i,j,k) \in N^2 \times M, i \neq j \tag{5.3}$$

$$\sum_{i \neq j} z_{ij} + (n-2) \sum_{i=1}^{n} \sum_{h \in M \setminus \{k\}} y_{ih} \geq \frac{1}{2} n(n-1) - 1, \ \forall (i,j,k) \in N^2 \times M, i \neq j \tag{5.4}$$

$$z_{ij} + z_{ji} + \sum_{h \in M \setminus \{k\}} (y_{ih} + y_{jh}) \geq 1, \ \forall (i,j,k) \in N^2 \times M, i \neq j. \tag{5.5}$$

Inequalities (5.1) and (5.2) give a lower bound for the summation of the z_{ij} variables, since only two processors are available for processing all the tasks and the DAG has $2h + 1$ or $2h$ tasks. Inequalities (5.3) and (5.4) are a generalization of inequalities (5.1) and (5.2), in case n tasks and two or three processors are considered. Inequalities (5.5) establish that either tasks t_i and t_j are processed in the same processor and then $z_{ij} + z_{ji} \geq 1$, or they are processed in different processors and then $\sum_{h \in M \setminus \{k\}} (y_{ih} + y_{jh}) \geq 1$.

We conjecture that they define facets of $P^{y_2}_{PLO}(D_{2h+1})$, $P^{y_2}_{PLO}(D_{2h})$, $P^{y_2}_{PLO}(D_n)$, $P^{y_3}_{PLO}(D_n)$, and $P^{y_p}_{PLO}(D_n)$, respectively.

Problem specific cuts Problem specific cuts are those that showed the best behaviour in the branch-and-cut algorithm (see Section 4.2.4). There are other cuts that still have to be tested or developed.

Least starting time inequalities (Section 4.1.1) have proven to be very strong even though they ignore the information about precedence constraints. We believe these inequalities could be strengthened if we were able to add some critical information about precedence constraints. Identifying bottlenecks in the DAG and finding an efficient way of using this information will possibly be of major importance for the solution of this problem.

Combinatorial structure of the instances There is much more to be understood about the combinatorial structure of the instances. Why some instances are more difficult to be solved than others of even greater size? Why are some gaps closed with very little effort of the branch-and-cut algorithm, while others remain open after hours of computation? Are there combinatorial invariants related to the structure of the DAG other than P_i, R_i and Q_i, that are cheap to calculate and give important information about the instance structure? These and other similar questions may have a combinatorial answer that could clarify some hidden aspects of this problem.

Separation algorithms There are not known polynomial separation algorithms for most families of inequalities of the PLO polytope. In certain cases, in which the number of inequalities is polynomial in the size of the problem, a complete enumeration algorithm could be applied. However, even in these cases some fast heuristics will be sometimes preferred. In this thesis, we proposed heuristics algorithms for two families of inequalities: 2-partition inequalities (3.14) and double 2-chorded cycle inequalities (3.49). The results in [7] lead us to conjecture that this second family of inequalities may be separated by an exact algorithm in polynomial time.

Bibliography

[1] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

[2] C. Berge. *Graphs*. North Holland, Amsterdam, 1985.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation - Numerical Methods*. Prentice Hall, Englewood Cliffs, 1989.

[4] J. Blazewicz, W. Cellary, R. Slowinski, and W. J. *Scheduling Under Resource Constraints - Deterministic Models*. J. C. Baltzer, Basel, 1986.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillian, London, 1976.

[6] L. Brunetta, M. Conforti, and M. Fischetti. A polyhedral approach to an integer multicommodity flow problem. *Discrete Applied Mathematics*, 101:13–36, 2000.

[7] A. Caprara and M. Fischetti. $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts. *Mathematical Programming*, 74:221–235, 1996.

[8] T. Christof, A. Löbel, and M. Stoer. *PORTA - a POlyhedron Representation Transformation Algorithm. Version 1.3.2*. Konrad-Zuse-Zentrum für Infortionstechnik (ZIB), Berlin, 1997.

[9] T. Christof and G. Reinelt. Low-dimensional linear ordering polytopes. Technical report, Universität Heidelberg, 1997.

[10] W. J. Cook, W. H. Cunnigham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, New York, 1998.

[11] H. Crowder, E. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[12] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.

[13] M. A. M. de Campos Gurgel. *Poliedros de Grafos Transitivos*. Tese de doutorado, Departamento de Ciência da Computação, Instituto de Matemática e Estadística, Universidade de Sao Paulo, 1992.

[14] M. R. Garey and D. S. Johnson. Strong NP-completeness results: Motivations, examples and implications. *Journal of the ACM*, 25:499–508, 1978.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP- Completeness*. W.H.Freeman, San Francisco, 1979.

[16] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy-Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[17] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[18] M. Grötschel, M. Jünger, and G. Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33:43–60, 1985.

[19] M. Grötschel, M. Jünger, and G. Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33:28–42, 1985.

[20] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming Series B*, 45:59–96, 1989.

[21] M. Grötschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47:367–387, 1990.

[22] F. Harary. *Graph Theory*. Addison-Wesley, Reading, 1969.

[23] K. Hoffman and M. Padberg. Solving airline crew-scheduling problems by branch-and-cut. *Management Science*, 39:667–682, 1993.

[24] O. H. Ibarra and S. M. Sohn. On mapping systolic algorithms onto the hypercube. *IEEE Transactions on Parallel and Distributed Systems*, 1:48–63, 1990.

[25] ILOG, Inc. *Using the CPLEX Callable Library, Version 6.0*, 1998.

[26] E. L. Johnson, G. L. Nemhauser, and M. W. P. Savelsberg. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal of Computing*, 12:2–23, 2000.

[27] J. P. Kitajima, B. Plateau, P. Bouvry, and D. Trystram. A method and a tool for performance evaluation. a case study: Evaluating mapping strategies. Technical report, Institut National Polytechnique de Grenoble, Grenoble, 1996.

[28] J. P. F. W. Kitajima. *Modèles Quantitatifs d'Algorithmes Parallèles*. Doctorate thesis, Institut National Polytechnique de Grenoble, 1992.

[29] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[30] S. Lin. Computer solutions to the traveling salesman problem. *BSTJ*, 44:2245–2269, 1965.

[31] M. Lübbecke. Algorithmen zur Enumeration aller Ecken und Facetten konvexer Polyeder. Master's thesis, Technical University of Braunschweig, Department of Mathematical Optimization, Braunschweig, 1996.

[32] N. Maculan, S. C. S. Porto, C. C. C. Ribeiro, and C. C. de Souza. A new formulation for scheduling unrelated processors under precedence constraints. *Revue d'Automatique, Informatique et Recherche Operationelle*, 33:87–90, 1999.

[33] S. Madal and J. B. Sinclair. Performance of synchronous parallel algorithms with regular structures. *IEEE Transactions on Parallel and Distributed Systems*, 2:105–116, 1991.

[34] D. A. Menascé and S. C. S. Porto. Processor assignment in heterogeneous parallel architectures. In *Proceedings of the IEEE International Parallel Processing Symposium*, pages 186–191. Beverly Hills, 1992.

[35] I. Méndez-Díaz and P. Zabala. A polyhedral approach for graph coloring. *Electronic Notes on Discrete Mathematics*, 7, 2001.

[36] P. Miliotis. Integer programming approaches to the traveling salesman problem. *Mathematical Programming*, 10:367–378, 1976.

[37] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, New York, 1988.

[38] M. Padberg and G. Rinaldi. Optimization of a 532 city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.

[39] S. C. S. Porto and D. A. Menascé. Processor assignment in heterogeneous message passing parallel architectures. In *Proceedings of the Hawaii International Conference on System Science.* Kauai, 1993.

[40] S. C. S. Porto and C. C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High Speed Computing*, 7:45–71, 1995.

[41] S. C. S. Porto and C. C. Ribeiro. A case study on parallel synchronous implementation of tabu search based on neighborhood decomposition. *Investigación Operativa*, 5:233–259, 1996.

[42] F. Rossi and S. Smriglio. A branch-and-cut algorithm for the cardinality stable set problem. *Operations Research Letters*, 28:63–74, 2001.

[43] A. S. Schulz. *Polytopes and Scheduling.* Doctorate thesis, Technischen Universität Berlin, Berlin, 1996.

[44] S. Thienel. *ABACUS - A Branch-And-CUt System.* Doctorate thesis, Universitat zu Köln, Köln, 1995.

[45] E. Uchoa and M. P. de Aragão. Vertex-disjoint packing of two steiner trees: polyhedra and branch-and-cut. *Mathematical Programming Series A*, 90:537–557, 2001.

[46] L. A. Wolsey. *Integer Programming.* Wiley, New York, 1998.