Sede Amministrativa: Università degli Studi di Padova

Dipartimento di Matematica

Corso di Dottorato di Ricerca in: Matematica

Curricolo: Matematica Computazionale

Ciclo XXIX

A branch-and-price approach for Pure Parsimony haplotyping

Coordinatore: Ch.mo Prof. Pierpaolo Soravia

Supervisore: Ch.mo Prof. Luigi De Giovanni

Dottorando: Veronica Dal Sasso

31 Gennaio 2017

Abstract

This thesis comes as the result of a detailed study of decomposition methods for large-scale problems and their application to a particular problem arising in computational biology.

The improvements on computer capabilities and programming techniques in the last decades have widened the set of problems that can be easily solved as Mixed Integer Linear programs. However, several applications still require formulations that involve a non-tractable amount of data necessary to describe the geometry of the solution space. In these cases, decomposition methods are used to reduce the size of the problems to be addressed.

In this thesis we propose the application of some of these methods, as *Dantzig-Wolfe reformulation*, *column generation* and *Lagrangian relaxation*, to a problem related to the study of the human genome. The human DNA is made of two double chains, each of which consists in a sequence of nucleotides. Among these, the ones related to the *Single Nucleotide Polymorphisms (SNPs)* are interesting as they describe the differences between individuals. We define a *haplotype* as a sequence of nucleotides that describes a portion of the SNPs found in a particular chromosome, and a *genotype* as the sequence that aggregates the information on SNPs coming from the double DNA chain of an individual. The problem we address falls into the class defining the *Haplotyping Inference* problem, that consists in recovering the structure of the haplotypes, given the information on the genotypes. In particular, we consider the parsimony criterion, which means that we want to find the minimum number of haplotypes able to explain all the genotypes. This problem is known to be *APX-hard*.

There are several contributions in the literature that can be divided into two main different classes of mixed integer linear formulations. The first one presents a polynomial number of both variables and constraints, thus these formulations are solved using a branch-and-cut approach. The second class consists of formulations that present an exponential number of constraints and variables, solved with a branch-and-cut-and-price approach. The scope of this thesis is to investigate how a new formulation that involves an exponential number of variables and a polynomial number of constraints can be solved by a branch-and-price approach. Its aim is to provide a competitive algorithm with respect to other formulations from the literature, in particular those with a polynomial number of constraints and variables.

We start by providing a review of the state of the art on the Haplotype Inference problem, with particular focus on the Mixed Integer Linear programming approaches for the Haplotype Inference by Pure Parsimony (HIPP) problem. We then consider a new mathematical programming formulation for HIPP that includes a set of quadratic constraints. By applying Dantzig-Wolfe reformulation, we obtained a new integer linear programming formulation, presenting an exponential number of variables and a polynomial number of constraints on the input data. This model is the basis for the development of a branch-and-price approach.

Due to the large number of variables involved, a column-generation approach is needed to solve the linear relaxation at a generic node of the search tree. An initial feasible solution is easily found by means of heuristics and used as starting point to build the Restricted Master Problem (RMP). In order to find variables to be added to the RMP, we solve a dedicated subproblem, the pricing problem, that in our case presents a quadratic objective function. We propose different ways of solving the pricing problem. Among the exact methods, we consider the integer linear model obtained by linearizing the quadratic objective function and a Smart Enumeration approach, that partitions the set of feasible solutions and solves the pricing problem restricted to each subset, exploiting some extra available information to further reduce the size of the subproblems. As heuristic approaches, we at first note that the pricing problem is easily solved for particular haplotypes. Then, for investigating the remaining solutions we propose a local search-based heuristic and an Early-terminated Smart Enumeration, where we stop the Smart Enumeration approach as soon as we find a variable that can be added to the RMP. The oscillatory behaviour of the dual variables involved in the definition of the pricing problem is limited by introducing a stabilization technique adapted to our formulation. In particular, we extended the proof of convergence of this procedure, that consists in using dual values obtained as convex combinations between real dual variables and a chosen stability center, to the cases in which the stabilized dual variables are feasible for the dual problem.

In order to solve the integer model, the solution of the linear relaxation is embedded in a branch-and-price approach. The branching rule we present is inspired to the well-known Ryan-Foster branching rule for set-partitioning problems. The correctness of our approach has been proved. Further observations on the similarity of the formulation's constraints to multiple set-covering ones suggest that we can relax a family of constraints to obtain a new formulation similar to a multiple set-covering. However, we note that the proposed branch-and-price algorithm applied to this formulation does not provide a feasible solution for HIPP, thus we need to integrate the proposed branching rule and recover a feasible optimal solution for HIPP.

This branch-and-price approach has been implemented in C++, with the aid of SCIP libraries and Cplex solver. Results have been obtained from different classes of instances found in literature, coming from real biological data and generated using ad-hoc programs, as well as newly generated ones. The branch-and-price approach proposed for our formulation proves to be competitive with state-of-the-art polynomial-sized formulations. In fact, we can note how the linear relaxation of our formulation is tighter than other linear relaxations and provides an effective starting solution for the branch-and-price algorithm. Results show how our approach is efficient, in particular on the set of instances that contain a larger number of genotypes

We proved therefore that a branch-and-price procedure provides a good solution approach for a formulation with exponential number of variables and polynomial number of constraints. Further work may include enhancements on the implementation details, such as exploring different ways of ordering the genotypes or combining heuristic and exact methods in the stabilized framework to solve the pricing problem. Moreover, it is possible to investigate the generalization of the proposed approach in order to solve set-partitioning problems.

Contents

Abstract **i**

Contents **iv**

Aknowledgements **vii**

1 Introduction **1**
- 1.1 The Haplotyping Inference by Pure Parsimony Problem 2
- 1.2 Notation 4
- 1.3 Structure of the thesis and contributions 5

2 Methodological and implementation tools **7**
- 2.1 Linear programming 7
  - 2.1.1 The Simplex Method 9
  - 2.1.2 How to deal with large-size LP models 11
- 2.2 Mixed Integer Linear Programming 14
  - 2.2.1 Cutting plane approach 15
  - 2.2.2 Branch-and-cut approach 16
  - 2.2.3 Branch-and-cut-and-price approach 19
- 2.3 Lagrangian relaxation 20
- 2.4 Dantzig-Wolfe decomposition 22
- 2.5 Some well-studied optimization problems 22
  - 2.5.1 Selection problem 23
  - 2.5.2 Knapsack problem 25
  - 2.5.3 Set-partitioning and set-covering problems 26
- 2.6 Implementing tools: SCIP and CPLEX 27

3 State of the Art **29**
- 3.1 Haplotype Inference in literature 29
  - 3.1.1 The Maximum Resolution Problem 29
  - 3.1.2 Perfect Phylogeny Haplotyping 30
  - 3.1.3 Other Haplotype Inference principles 32
- 3.2 Haplotype Inference by Pure Parsimony 32
- 3.3 Exponential-size integer formulations 33
  - 3.3.1 An intuitive formulation 33
  - 3.3.2 A set-covering approach. 34
- 3.4 Polynomial-size integer formulations 38

3.4.1 A formulation with duplicate haplotypes 38
3.4.2 A formulation with distinct haplotypes 39
3.4.3 A class representative model 41
3.5 A Heuristic approach 43

4 New formulations for the HIPP problem 47
4.1 Improvements to the class representatives model 47
4.2 A new intuitive polynomial two-index formulation 50
4.3 A new exponential-size formulation 52

5 A column-generation approach for the LP 59
5.1 Heuristics for computing an initial solution of the Restricted Master Problem 59
5.2 Different approaches to solve the Pricing Problem 61
5.2.1 Solve the Mixed Integer Program 63
5.2.2 A Smart Enumeration approach 63
5.2.3 Fixed haplotypes 65
5.2.4 A Local Search-based approach 66
5.2.5 An Early-terminated Smart Enumeration 66
5.2.6 A Maximum flow approach 67
5.3 Adding multiple variables per iteration 72
5.3.1 Adding variables with a larger genotypes' subset 72
5.3.2 Adding a selected set of extra variables 74
5.4 Convergence issues 75
5.4.1 A Lagrangian lower bound 77
5.4.2 A stabilization method to smooth dual variables' values 78
5.4.3 Stabilized column generation and heuristic pricing 82

6 Branch-and price frameworks for the IP 85
6.1 Branching rule for the HIPP problem 85
6.1.1 The branching rule 86
6.1.2 Proof of correctness 88
6.1.3 Branching priorities 92
6.2 Solution method on a set-covering-like relaxation 92
6.2.1 The drawbacks of the set-covering-like relaxation 93
6.2.2 Recovering the optimal integer solution 94

7 Implementation 99
7.1 The LP of the exponential-size formulation 99
7.1.1 Choosing an order for the genotypes 99
7.1.2 Overview of the procedure to solve the linear relaxation 100
7.2 The branch-and-price framework 102
7.2.1 Embedding branching constraints in the pricing problem 102
7.2.2 Embedding branching constraints in the Smart Enumeration approach 106

8 Computational results 109
8.1 Alternative algorithm configurations 109

8.2 Instance benchmarks 110
8.3 Choosing parameters and algorithm configuration 111
8.3.1 Tuning of the parameters 111
8.3.2 Comparing alternative formulations 112
8.4 Computational results on different benchmarks 114
8.5 Computational results on the *manygen* benchmark 115
8.5.1 Performance on the solution of the root node 115
8.5.2 Performance of the branch-and-price algorithm 117

9 Conclusion 119

Bibliography 121

Aknowledgements

I would like to thank my supervisor Luigi De Giovanni, for all the help he gave me during these three years, for being patient and never letting me feel I was too slow in understanding the theory behind what I was supposed to do. He always tried to rise my self-esteem, particularly concerning my coding skills.

Special thanks go also to Prof. Martine Labbé, that supervised me during my stay at the Université Libre de Bruxelles and gave me precious advices on the mathematical aspects related to part of this thesis. The opportunity I had of working in her research group has been greatly rewarding.

I want to thank the SCIP team, which is always ready to help through the mailing list, whether the questions they receive are trivial or challenging. Thank you also to the IRIDIA team for providing a useful tool for tuning algorithms and in particular to Leslie Pérez Cánceres, that helped me with the really basics of it.

Thank you to Prof. Daniele Catanzaro and Prof. Giuseppe Lancia, who kindly spent their time reading my thesis and who provided me with constructive feedbacks.

If the support I received from an academic point of view has been fundamental in carrying out this thesis, the friendly environment that surronded me has not been less important. Whether it was for the lunch time spent together, coffee breaks, frittella breaks, (and I regret not so many) spritz and evenings together, I have to thank my friends at the Math department in Padova, from the ones that greeted me when I arrived in the small 339 office to the ones I left in the bright 712 office, to the ones I never shared an office with. A heartfelt thank you to the collegues and friends of the Computer Science department at the ULB, for the time spent together during both the working and spare time. It has been a real pleasure to share with you ideas, experiences and happy moments.

Thanks to the long-known friends, always present when it comes to have a good dinner together or simply to meet and have a drink after a busy, tiring week. I should also mention both been teased for my naivety in coding and been given advices on how to improve my skills from some of you. This is what friends are for.

The greatest thanks go, finally, to my family, that always supported me. Parents that worried when I was worried, that had relief toasts with me when finally I was able to find a way of overcoming difficulties. And an older brother always ready to give wise advices or to spend with me wonderful vacations.

“I think,” said Anne slowly, “that I really have learned to look upon each little hindrance as a jest and each great one as the foreshadowing of victory.”
Lucy M. Montgomery

CHAPTER 1

Introduction

Mixed Integer Linear Programming (MILP) is a powerful tool to solve problems that arise from different contexts, for which we want to identify the most convenient scenario among well-defined choices. As the set of these possible choices can include a large number of different scenarios, evaluating and comparing every single possibility is impractical. Thus, the strength of a MILP approach consists in avoiding the extensive computations and, by means of solving a mathematical model, detecting the best solution. A Mixed Integer Linear Program (or MILP model) consists of a linear function that we want to maximize or minimize within a set of possible choices, represented by linear inequalities and integrality constraints on some variables. With the improvements on mathematical programming techniques and computer's architecture and technology, in particular the extension of memory capacity and processor's speed, the dimension of programs that we consider as tractable has significantly increased in the last decades. Nevertheless, many problems addressed by MILP are formulated in such a way that still challenges the present computational limits. Thus, solving MILP models still presents several challenging issues whenever we face large-scale problems, that involve a huge number of constraints or variables and lead to non-acceptable computational times. Several approaches presented in the literature aim at overcoming these difficulties and provide viable solution processes for large-scale formulations. These approaches consist in reformulating difficult programs into easier ones, for example applying a *Dantzig-Wolfe decomposition*, relaxing some families of constraints as to obtain a *Lagrangian relaxation*, or reducing the size of the program to an acceptable one and then adding those variables or constraints that are needed to detect the best solution, as in *row* and *column generation*.

In this thesis we consider a particular problem arising from computational biology and a possible formulation characterized by an exponential number of variables. Solving the associated MILP model involves the use of decomposition methods, like the ones mentioned above. In the next sections 1.1 and 1.2 we present respectively the problem and the main notation used throughout this thesis. Section 1.3 shows the structure of the thesis and the main contributions.

1.1 The Haplotyping Inference by Pure Parsimony Problem

The sequencing of human genome has been one of the most important achievements of the last years in molecular biology. It has been completed in 2001 and showed how all humans share the 99% of the information contained in the DNA, while all the significant differences among individuals, for example the color of the eyes or of the hair, are contained in the remaining information. Each site of this 1% portion of the human genome is called a *Single Nucleotide Polymorphism (SNP).*

It is well known that humans are diploid organisms, meaning that the DNA is organized in pairs of chromosomes, each copy coming from one of the two parents, unless a recombination occurs. In the case of a recombination event, it is possible that a chromosome copy of the child consists of portions from both chromosome's copies of the same parent. However, recent studies show that human chromosomes have a block structure, meaning that they can be partitioned into blocks in which no (or very few) recombinations have occurred. For this reason, we will not deal with recombinations, as one can always assume we are considering DNA portions involving only one of these blocks. Every single chain in the DNA is made of a sequence of nucleotides, each of which is made of a phosphate group, a five-sided sugar and a nitrogenous base. The nucleotide is fully characterized by the base, that can be chosen among the four: Adenine (A), Thymine (T), Cytosine (C) and Guanine (G), so we can refer to different kind of nucleotides referring to their nitrogenous base.

Thus, every SNP is characterized by a pair of nucleotides, each of them belonging to the same chain in the two different chromosome's copies. Studies show that the human DNA sites are almost always biallelic, that is only two nucleotides out of four can be found at any site. We refer to each of these possible values as to the *alleles* of that site. As a consequence, the nucleotides identifying a SNP can be equal on both chromosome's copies, or they can be different, but in any case they can assume only two values. For example, let us assume only nucleotides A and C appear in a certain position. Then, four scenarios are possible for that particular SNP:

1. both nucleotides are of type A,
2. both nucleotides are of type C,
3. the nucleotide in the first chain is of type A, the second of type C,
4. the nucleotide in the first chain is of type C, the second of type A.

According to these scenarios, we can give the following definitions:

Definition 1.1 (Heterozygous and homozygous SNP). *If the same nucleotide appears in both chromosome's copies, as in cases 1 and 2, then the SNP is* homozygous*, otherwise it is* heterozygous*.*

Definition 1.2 (Haplotype and genotype). *A* haplotype *is a single sequence of SNP values of a chromosome copy. A* genotype *is a sequence of values providing information regarding the union of the two chromosome copies, that in particular tells us if each SNP is homozygous or heterozygous, specifying which allele composes the homozygous sites.*

Example 1.1. *We give here an example on how the information regarding a pair of haplotypes is combined to form a genotype. Note that we denote with the symbol* $-$ *a heterozygous site.*

$$\left.\begin{array}{l} h^1 = A\ C\ A\ C \\ h^2 = A\ C\ C\ A \end{array}\right\} g^1 = A\ C\ \text{-}\ \text{-}$$

Here we give another example:

$$\left.\begin{array}{l} h^3 = A\ C\ A\ A \\ h^4 = A\ C\ C\ C \end{array}\right\} g^2 = A\ C\ \text{-}\ \text{-}$$

Note that the two genotypes g^1 and g^2 in Example 1.1 are identical and may be associated to the same information, but, in fact, they come from different single sequencies of nucleotides. This ambiguity arises from the fact that the definition we gave of a genotype does not discern case 3 from case 4. The reason that justifies this ambiguity is the fact that the information on homozygous and heterozygous sites is obtained *in vitro* and, normally, experiments are not able to provide also information on which chromosome's copy is which nucleotide.
Moreover, we give the following definition:

Definition 1.3 (Genotype resolution)**.** *Two haplotypes* resolve *a certain genotype if the information they provide regarding homozygous and heterozygous sites is consistent with the genotype itself, when paired.*

In Example 1.1, h^1 and h^2 resolve g^1. Notice that g^1 could also be resolved by h^3 and h^4.

Haplotypes have an important role in medical and pharmaceutical studies, for example they are fundamental in detecting diseases or in studying the different behaviour of various individuals to the same therapy. Sequencing them *in vitro* is not practical, as it is normally very expensive and time consuming, while it is easier to experimentally obtain the information stored in genotypes. Thus, we want to answer the question whether it is possible to deduce *in silico* (hence more conveniently) the haplotypes that resolve all the given genotypes. Several approaches have been used in order to solve this problem, its difficulty consisting in the fact that, due to the ambigous definition of heterozygous sites, once we have k heterozygous SNPs in the same genotype we have 2^{k-1} possible pairs of haplotypes that can resolve it. We need then some criteria to prefer one pair among the others. As usually we want to deal with a population, we assume to have a set of genotypes, one for each individual, that represent the same portion of the DNA and we look for a criterion that has a biological foundation to choose which haplotypes resolve the given genotypes. Different approaches have been proposed in biological literature, that will be described in Chapter 3, considering for example the evolutionary history of a population, modeled as a rooted tree, or the so called *Pure Parsimony* criterion, based on the observation that in the real cases only a small number of haplotypes that resolve a set of genotypes is observed. Thus, we are inclined to consider as good a solution if it involves the smallest possible number of haplotypes and we state the problem as follows:

Definition 1.4 (Haplotyping Inference by Pure Parsimony Problem - HIPP). *Given a set of genotypes G, determine a set of haplotypes H such that*

- *each genotype is resolved by two haplotypes belonging to H,*
- *H has minimum cardinality.*

1.2 Notation

In this section we introduce some notation that will be useful throughout the thesis. First of all, remembering the biallelic property of each SNP site, we do not need to code the values of each chain site using one of the four nucleotides names, but we just need two symbols to identify the two different nucleotides that can be found in a particular position. For example, knowing that a SNP only admit the nucleotides A and C, we can associate the value 0 to A and the value 1 to C, while for another SNP allowing only the other nucleotides, we can associate the values 0 to T and 1 to G respectively. In this way, a haplotype of n SNPs can be encoded as a string of length n over the alphabet $\{0, 1\}$. By using this approach we can also represent a genotype as a string, taking care of the right encoding for homozygous and heterozygous sites. Thus, genotypes are represented by strings of length n with values within the alphabet $\Omega = \{0, 1, 2\}$, where homozygous sites are represented by their value itself and the value 2 identifies heterozygous sites, as can be clearly seen in the foolowing example:

Example 1.2. *Consider a small portion of the human DNA contained in a specific chromosome, where capital letters represent SNPs:*

```
taggtccCtatttTccaggcgcCgtatacttcgacgggTctata
taggtccCtatttGccaggcgcAgtatacttcgacgggTctata
```

The haplotypes obtained from each chain of nucleotides are the following:

$$CTCT$$
$$CGAT$$

that can be represented as strings in $\{0, 1\}$ with associated genotype g as string in $\{0, 1, 2\}$:

$$\left.\begin{array}{l} h^1 = 0101 \\ h^2 = 0011 \end{array}\right\} \quad g = 0221$$

Using this formulation, we can formalize the notion of haplotypes that resolve a genotype.

Definition 1.5 (Componentwise sum of haplotypes). *Let h^1 and h^2 be two haplotypes representing the same set of n SNPs, we define their componentwise sum as*

$$(h^1 \oplus h^2)_p = \begin{cases} 0 \textit{ if } h^1_p = h^2_p = 0 \\ 1 \textit{ if } h^1_p = h^2_p = 1 \\ 2 \textit{ if } h^1_p \neq h^2_p \end{cases} \qquad \forall p = 1, \dots, n$$

Definition 1.6 (Genotype resolution via haplotype sum). *Let h^1 and h^2 be two haplotypes of length n and g a genotype of length n. We say that two haplotypes h^1 and h^2 resolve a genotype g if $h^1 \oplus h^2 = g$.*

Note that genotypes may exist with no heterozygous sites. In this case, it is trivial to choose the haplotypes that resolve it, as they are simply two copies of the genotype itself. Moreover, these haplotypes are forced to be present in the solution of the HIPP problem.

Definition 1.7 (Fixed haplotypes and fixed genotypes). *A genotype is fixed if it has no heterozygous site. The haplotype that resolves it is called a fixed haplotype.*

Other useful definitions involve the notion of compatibility between haplotypes and genotypes.

Definition 1.8 (Compatibility between haplotypes and genotypes). *Let h^1, h^2 be haplotypes and g^1, g^2 be genotypes representing the same set of SNPs. We say that:*

1. *h^1 and g^1 are compatible if for each position p such that $g^1_p \neq 2$ we have $g^1_p = h^1_p$,*
2. *g^1 and g^2 are compatible if $g^1_p = g^2_p$ whenever they are both different from 2 or, equivalently, they have at least a compatible common haplotype.*

Definition 1.9 (Complementary haplotype). *A haplotype h^1 is the complementary of h^2 (and viceversa) with respect to a genotype g^1 if h^1 and h^2 can be used together to resolve g^1.*

From now on, we consider a set of m genotypes $G = \{g^1, \ldots, g^m\}$ made of n SNPs. The set of all haplotypes that are compatible with at least one genotype of G is called H_G. The set of positions associated to SNPs is denoted by $P = \{1, \ldots, n\}$. Given a haplotype h, we denote with $G(h) \subseteq G$ the subset of genotypes compatible with h.

1.3 Structure of the thesis and contributions

The thesis is divided into the following Chapters.

Chapter 2 briefly presents the basic methodological tools that will be used and adapted in the modelling and solution approach proposed in this thesis, together with the tools used for its implementation. In particular, Mixed Integer Linear Programming, Decomposition methods and some well-studied optimization problems of interest for this thesis will be introduced. We will also mention the software libraries SCIP and Cplex, used to implement in the C++ computer language the algorithms developed in the thesis.

Chapter 3 reviews the state of the art and makes an excursus on the different MILP approaches investigated in literature to solve haplotyping problems. After presenting approaches for alternative haplotyping criteria, we focus on pure parsimony haplotyping and review the formulations of the HIPP problem that present a polynomial or exponential number of variables.

In Chapter 4 we give in detail three different formulations for HIPP. The first one has a polynomial number of both constraints and variables and represents a slightly improved version of a polynomial-size MILP model from the literature. The second one presents a quadratic set of constraints. It has been reformulated using a Dantzig-Wolfe decomposition to obtain the third formulation, an integer linear programming model which, to the best of our knowledge, is the first one including an exponential number of variables and a polynomial number of constraints. In this thesis we show how we can obtain a solution approach whose performance is competitive with state-of-the-art polynomial-sized formulations by applying decomposition methods.

Chapters 5 and 6 discuss how a branch-and-price approach can be devised to solve the third MILP formulation.

Specifically, in Chapter 5 we investigate how to solve the linear relaxation of the new HIPP formulation with a column-generation algorithm. We provide a heuristic procedure to find a good initial solution, needed to build the initial model corresponding to the Restricted Master Problem, and we propose several methods for solving the pricing subproblem that yields the column to be added. To fasten the convergence of the column-generation procedure, we derive a lower bound obtained from a Lagrangian relaxation of our formulation and we apply a stabilization technique to smooth the values of the dual variables used to solve the pricing subproblem. Our lower bound definition is based on a different relaxation with respect to the one usually applied in stabilized column generation, and we need to extend the proof of convergence of our stabilization approach so that dual feasible stabilized variables are allowed.

Chapter 6 discusses the branching procedure that allows us to recover variable integrality and, thus, solve HIPP. The branching strategy, which stems from observing that the structure of formulation's constraint can be related to set-partitioning problems, can be seen as a generalization of the Ryan-Foster branching rule. The same observation leads to an alternative way of solving the HIPP problem, where we solve at first a relaxation corresponding to a set-covering formulation. This has the advantage of reducing the computational time, but the proposed branching stategy has to be integrated with further branching rules, leading to the overall branch-and-price algorithm described in the second part of the chapter.

In Chapter 7 we describe the implementation details that define the overall branch-and-price algorithm proposed for HIPP. Particular attention has been devoted to how the branching constraints are embedded in the different approaches for the pricing subproblem, in such a way that their structure and their performance is preserved in all the nodes of the branching tree.

Chapter 8 presents the computational results and discusses the performance of the proposed branch-and-price algorithm on different benchmarks from literature and on a new class of large-size instances with increased number of genotypes. Results show that the approach proposed in this thesis outperforms polynomial-size formulations on the new benchmark.

Finally, Chapter 9 concludes the thesis by providing some final remarks and discussing future lines of research.

CHAPTER 2

Methodological and implementation tools

Different Mixed Integer Linear Programs provide formulations for the HIPP problem studied in this thesis. In particular, one amongst these formulations is suitable to be solved by a branch-and-price approach, using column generation to solve linear relaxations. In this chapter we give a brief introduction on what Linear and Integer programming is, together with the most common procedures used to solve problems modeled in such ways. Exhaustive presentation of these tools can be found in [11, 16, 30] among others. Moreover, an overview on column generation and the main procedures used to improve the standard algorithm are presented, for example, in [34, 35].

We report in particular the row- and column-generation procedures, applied to solve large size linear programs, and approaches to solve mixed integer linear programs such as cutting plane method, branch-and-cut and branch-and-price. In the last section we summarize some well studied problems that can occur as subproblems in the approach that we will device for the HIPP problem.

2.1 Linear programming

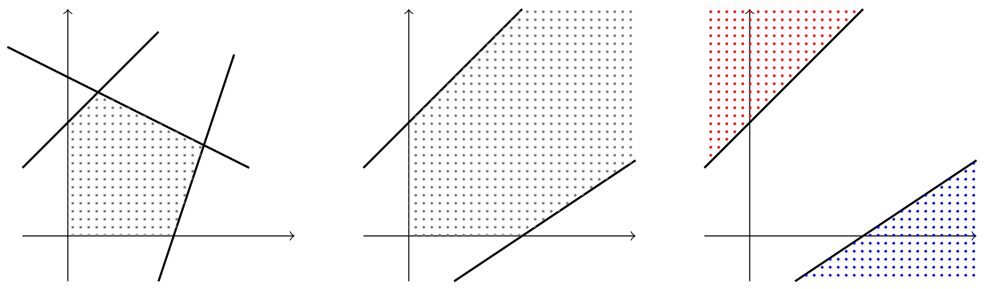
A linear program (LP) is the description of an optimization problem that involves the maximization or the minimizing of a linear function of continuous variables, subjected to some constraints. As for the *objective function*, also the constraints are linear:

$$\min \quad \sum_i c_i x_i \tag{2.1}$$

$$\text{s.t.} \quad \sum_i a_{ij} x_i \geq b_j \qquad \forall j \tag{2.2}$$

$$x_i \geq 0 \qquad \forall i \tag{2.3}$$

An LP can be described in compact form by using matrix algebra. We represent the coefficients a_{ij} of the constraints with a matrix A, the known terms b_j with a vector b



(a) bounded feasible region (b) unbounded feasible region (c) empty feasible region

Figure 2.1: Feasible regions.

and the coefficients of the objective function c_i with a vector c:

$$\min \quad c^T x \tag{2.4}$$

$$\text{s.t.} \quad Ax \geq b \tag{2.5}$$

$$x \geq 0 \tag{2.6}$$

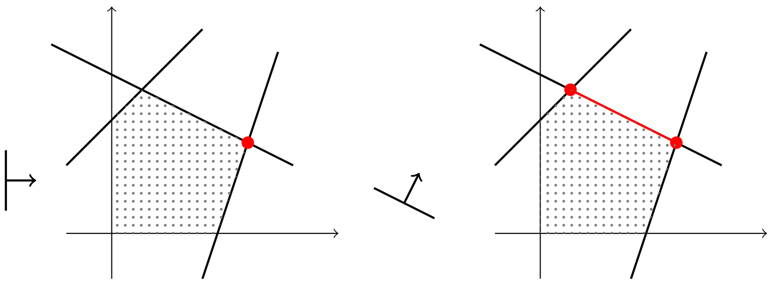
All other linear programs can be transformed to take the shape (2.4)-(2.6).

If a vector $x \geq 0$ satisfies the inequalities $Ax \geq b$ then it is called a *feasible solution*, and if no other better solution can be found, x is also an *optimal solution* to the LP. If the optimal solution does not exist, there are two cases: the problem is *infeasible*, i.e. no feasible x exists, or *unbounded*, i.e. for every $\alpha < 0$ there exists a feasible x such that $c^T x < \alpha$.

The set of all feasible solutions is called the *feasible region*, and it can be seen as the intersection of a finite number of closed half-spaces, each one defined by a constraint of the linear program. If this region is bounded, we call it a polytope, otherwise it is called a polyhedron.

Example 2.1. *We give a graphical interpretation of these concepts in Figure 2.1: in subfigure 2.1a we show a bounded feasible region, whose boundaries are defined by the inequalities* $x + 2y \leq 4.2$, $-x + y \leq 1.5$, $x - 3y \leq -12.6$, $x \geq 0$ *and* $y \geq 0$, *in subfigure 2.1b the feasible region is unbounded and described by* $-x + y \leq 1.5$, $2x - 3y \leq 3$, $x \geq 0$ *and* $y \geq 0$, *while in subfigure 2.1c the half-spaces* $-x + y \geq 1.5$ *and* $2x - 3y \geq 3$ *do not intersect, so that the feasible region is empty.*

Given a polyhedron P, a *vertex* is a point $x \in P$ that is not a convex combination of any other two distinct points in P, that is, we cannot find $x_1, x_2 \in P$ and $\lambda \in]0, 1[$ such that $x_1 \neq x_2$ and $x = \lambda x_1 + (1 - \lambda)x_2$. According to the direction of the objective function, a linear program that allows optimal solutions can either have a unique optimal solution, obtained at a vertex of the polyhedron, or multiple optimal solution, where at least one is achieved at a vertex, as seen in Figure 2.2: on the left of each feasible region we show the direction in which the objective function improves and in red the corresponding optimal solution(s).



(a) unique optimal solution (b) multiple optimal solutions

Figure 2.2: Unique and multiple optimal solutions.

2.1.1 The Simplex Method

There are different algorithms that are used to solve a linear program. Among these, the *Simplex Method* is widely used. It is an algorithm that proceeds by moving from a vertex to another one of the feasible region, improving the value of the current solution, until it reaches optimality. This method is largely used because of its features. It is able to solve any linear program, it can be used to detect whether the problem is infeasible or unbounded, it recognizes redundant constraints in the formulation and it is able to handle problems with more than one optimal solutions [11].

The simplex method starts from a linear program in standard form, that is given by:

$$\begin{aligned} \min \quad & c^T x && (2.7)\\ \text{s.t.} \quad & Ax = b && (2.8)\\ & x \geq 0 && (2.9) \end{aligned}$$

Note that every linear program can be transformed into standard form applying straightforward operations on the rows of the linear system and on the objective function. This program is then re-written into the so-called *tableau form*. Consider an $m \times n$ matrix A such that $rank(A) = m$ $(m \leq n)$. A *basis* for A is a subset of indices $B \in \{1, \dots, n\}$ such that $|B| = m$ and the square submatrix A_B, obtained considering only the columns whose indices are in B, is non-singular. Note that B allows us to select a subset of m variables and the related columns that form matrix A_B. Let $N = \{1, \dots, n\} \backslash B$. A *basic solution* associated with the basis B is the only solution of $Ax = b$ such that $x_N = 0$ and $x_B = A_B^{-1} b$. Suppose, for ease of notation, that such a basis is given by $B = \{1, \dots, m\}$, so that $A = (A_B, A_N)$ and $x^T = (x_B^T, x_N^T)$. This means that x is feasible if and only if

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N.$$

Thus, considering the value of the objective function z, we have $z = c^T x$, or equivalently $z - c_B^T x_B - c_N^T x_N = 0$. We obtain that

$$z - (c_N^T - c_B^T A_B^{-1} A_N) x_N = c_B^T A_B^{-1} b.$$

Then, defining

$$\begin{aligned}
&\bar{A}_N = A_B^{-1} A_N,\\
&\bar{b} = A_B^{-1} b,\\
&\bar{c}_N = c_N - A_N^T {A_B^{-1}}^T c_B = c_N - \bar{A}_N^T c_B\\
&\bar{z} = c_B^T A_B^{-1} b
\end{aligned}$$

we get the tableau form:

$$\begin{array}{ll} \min & z \\ \text{s.t.} & z - c^T x = 0 \\ & Ax = b \\ & x \geq 0 \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \min & z \\ \text{s.t.} & z - \bar{c}_N x_N = \bar{z} \\ & x_B + \bar{A}_N x_N = \bar{b} \\ & x \geq 0 \end{array}$$

The steps performed by the algorithm, once it is given a matrix A, vectors b and c and a feasible basis $B = \{B[1], \ldots, B[m]\}$, are shown in Algorithm 2.1.

Algorithm 2.1 Simplex Method

```
1:  procedure SIMPLEX(A, b, c, B)
2:      compute the tableau associated to the basis $B$ and the basic solution $\bar{x}$,
3:      if $\bar{c} \geq 0$ then
4:          $\bar{x}$ is optimal. STOP.
5:      else
6:          choose $k$ such that $\bar{c}_k < 0$,
7:          if $\bar{a}_{ik} \leq 0$ for all $i \in \{1, \ldots, m\}$ then
8:              the problem is unbounded. STOP.
9:          else
10:             choose $h \in \{1, \ldots, m\}$ such that $\bar{a}_{hk} > 0$ and $h = \text{argmin} \frac{\bar{b}_h}{\bar{a}_{hk}}$,
11:             set $B[i] = k$ for $i$ such that it was $B[i] = h$,
12:             go back to Step 2,
13:         end if
14:     end if
15: end procedure
```

Thus, starting from a feasible solution, identified by the initial basis B, the algorithm looks for a variable that can potentially improve the value of the objective function, based on the variable that reaches the minimum value for $\frac{\bar{b}_h}{\bar{a}_{hk}}$. This new variable x_k replaces the chosen one x_h and the basis B is updated so that $B[i] \leftarrow B[i]$ for each index i s.t. $B[i] \neq h$ and $B[i] \leftarrow k$ if we had $B[i] = h$. Variable x_k is set to a value such that we recover a feasible solution with equal or improved objective value. It may happen that $\min_h \frac{\bar{b}_h}{\bar{a}_{hk}}$ is not unique. In this case, selecting arbitrarily one of the candidate variables we face a degenerate solution: when we compute the related tableau, not only x_h but also other variables still in the basis are set to value 0. At the next iteration of the simplex method, this can lead to a change of basis that does not change the objective function's value, thus affecting the convergence of the algorithm to the optimal solution. This issue, that often occurs in practical problems, can be overcome easily, for example by applying the following rule.

Definition 2.1. Bland's rule [9]. *Consider a predefined order on the variables indices and, when choosing the variables for the change of the basis, always choose the ones with minimum index.*

Thus, convergence of the simplex algorithm is ensured by the facts that there are at most $\binom{n}{m}$ different bases, the method never increases the value of the current solution and Bland's rule ensures that, even if the algorithm visits degenerate basis, the same basis will never be visited twice. It is possible then to prove that the Simplex Method, when using Bland's rule, converges in at most $\binom{n}{m}$ iterations. In practice, however, a much smaller number of iterations suffices to find the optimal solution [11].

2.1.2 How to deal with large-size LP models

While for small-size programs the Simplex Method is highly efficient in practice, when the size of the models increases it could become impractical, due to the huge amount of data that needs to be stored, possibly exponential on the size of the input data. There are two ways in which the size of a program can increase and become impractical: we can have a large amount of constraints, that means that matrix A has a large number of rows, or a large number of variables, so that matrix A has too many columns. The literature describes different strategies in order to deal with both these issues, allowing to consider only a subset of important constraints or variables.

Row generation

Consider the case in which matrix A has a non-tractable number of constraints (as an example, exponential with respect to the size of the problem's data). The idea underlying the row generation method is that the same optimal solution can be reached considering different feasible regions for the same problem, as shown in Figure 2.3. Given the direction in which the objective function improves, we can easily see in Figure 2.3b that we do not affect the final solution of the problem by leaving out from the description of the feasible region two half-spaces. It suffices, then, to provide a tool that discerns which constraints are sufficient to identify the optimum. As this cannot be said a priori, in practice we need an oracle that, given a subset of the program's constraints, tells us when no other constraints are missing or that provides a new constraint to be added to the current set.

In order to reduce the number of constraints considered in the solution process, we define a subset of constraints described by $A'x = b'$, where A' is a matrix containing a subset of rows of A and b' is a vector made of the entries of b corresponding to the rows in A'. Notice that we can even start without any constraint. We give a sketch on how row generation works in Algorithm 2.2. Every time we get an optimal solution $\tilde{x}$ of this reduced model, we need to check whether $\tilde{x}$ is feasible for the original formulation or if there is a constraint that is violated. In the first case, we can say that $\tilde{x}$ is optimal and stop the algorithm, otherwise we add the found constraint and solve again the resulting linear program.

The core point of this method is the detection of a procedure suitable for the role of oracle, that strongly depends on the structure of the program and on the properties of the particular constraints involved.

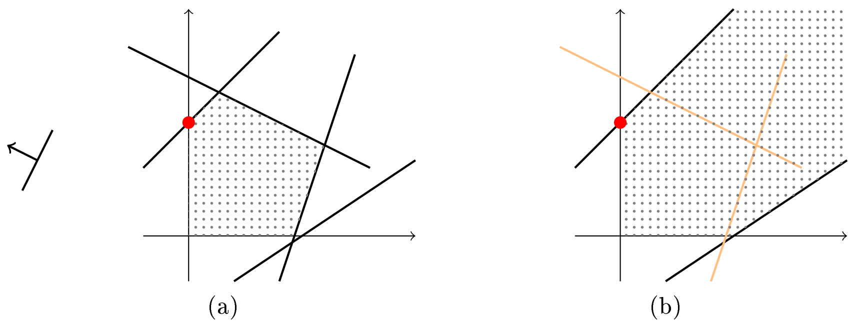


Figure 2.3: Different feasible regions for same optimal solution.

Algorithm 2.2 Row Generation

```
1: procedure Row generation
2:     build the linear program including only constraints $A'x = b'$,
3:     solve the linear program and get the solution $\tilde{x}$ ,
4:     if $\tilde{x}$ is feasible for the complete program then
5:         set $x^{OPT} = \tilde{x}$. STOP.
6:     else
7:         add a constraint contained in $Ax = b$ that is violated by $\bar{x}$,
8:         go back to Step 3,
9:     end if
10: end procedure
```

Column generation

Column generation, on the other hand, is applied when matrix A has a non-tractable number of columns, so that the problem involves a great number of variables. In this case, however, only few of these variables will assume a non-zero value in a basic optimal solution and so it seems advisable to find a way of considering only the relevant ones. As for the row generation method, the main idea of column generation consists in considering at the beginning a reduced number of variables to which other necessary variables are iteratively added. Also in this case, a procedure is needed that could identify a new variable or say that no other variable is necessary, since we reached the optimum.

In order to describe this approach we need to introduce some notions on duality. In fact, for each linear program, that we call primal program, we can build another linear program, called its dual, whose variables are associated to the primal constraints and whose constraints are associated to the primal variables. Thus, looking for an extra variable to be added to the primal program is equivalent to looking for a constraint to be added in the dual.

Given a linear program with the notation of (2.4)-(2.6), its dual linear program can be formulated following some simple rules: a maximization problem becomes a minimization one and viceversa, while the sense of the constraints determines the sign of the variables and viceversa, as in Table 2.1.

The definition of this dual program is such that the following theorems hold:

Primal	Dual
min	max
$A_i x \leq b_i$	$\pi_i \leq 0$
$A_i x = b_i$	$\pi_i \in \mathbb{R}$
$A_i x \geq b_i$	$\pi_i \geq 0$
$x_j \geq 0$	${A_j}^T \pi \leq c_j$
$x_j \in \mathbb{R}$	${A_j}^T \pi = c_j$
$x_j \leq 0$	${a_j}^T \pi \geq c_j$

Table 2.1

Theorem 2.1 (Weak duality [11]). *Given a primal program* $\min\{c^T x : Ax \geq b, x \geq 0\}$ *and its dual* $\max\{b^T \pi : A^T \pi \leq b, \pi \geq 0\}$*, if the respective feasible regions are not empty, then for every feasible x and π we have that*

$$c^T x \geq b^T \pi.$$

Theorem 2.2 (Strong duality [11]). *Given a primal program* $\min\{c^T x : Ax \geq b, x \geq 0\}$ *and its dual* $\max\{b^T \pi : A^T \pi \leq b, \pi \geq 0\}$*, if the respective feasible regions are not empty and neither of the problems is unbounded, then there exist optimal $\bar{x}$ and $\bar{\pi}$ such that*

$$c^T \bar{x} = b^T \bar{\pi}.$$

We describe now how these notions about duality are used in the column-generation process. We define the *Restricted Master Problem RMP* as the primal problem (2.4)-(2.6) in which only a subset of variables has been considered. Suppose we solve the RMP, and let $\tilde{\pi}$ be the values of the dual variables associated, that can be derived from the last tableau obtained when using the simplex algorithm. Note that these values are such that $b^T \tilde{\pi}$ is equal to the value of the primal objective function. If $\tilde{\pi}$ is a vector feasible for the dual, that has the form

$$\begin{aligned} (\overline{\text{D}}) \quad \min \quad & b^T \pi && (2.10)\\ \text{s.t.} \quad & A^T \pi \leq c && (2.11)\\ & \pi \geq 0, && (2.12) \end{aligned}$$

then it is also optimal, by the weak and strong duality properties. If it is not feasible, instead, there is at least one constraint of the set (2.11) that is violated and can be found solving the following problem, called the *pricing problem*:

$$(\overline{\text{PP}}) \quad \min\{c_i - \tilde{\pi} a_i : i \in \{1, \dots, n\}\} \tag{2.13}$$

where a_i is the i-th column of matrix A. It is possible to look for such a constraint by iterating over all variables the computation of the reduced costs $c_i - \tilde{\pi} a_i$. An alternative way consists in formulating the pricing problem as an optimization problem, for example as a MILP model, that exploits the structure of the primal variables and gives as solution the minimum reduced cost.

In the case in which $\tilde{\pi}$ is dual infeasible, the minimum achieved is negative, thus we found a violated dual constraint. This constraint is associated by construction to a primal variable x_k, that does not appear yet in the RMP, so we can add it in order to obtain a better solution. It can be easily seen that the column-generation procedure for the primal program has been translated in a row generation approach for the dual.

We outline the column-generation procedure in the Algorithm 2.3. Note that this

Algorithm 2.3 Column Generation

```
1: procedure Column Generation
2:     build the RMP from a subset of variables,
3:     solve the RMP and get the solution $\tilde{z}$ and the associated dual variables $\tilde{\pi}$,
4:     if $\tilde{\pi}$ is dual feasible then
5:         set $x^{OPT} = \tilde{z}$. STOP.
6:     else
7:         solve ($\bar{\mathrm{P}}$P) and find a variable $x_k$ to be added to the RMP,
8:         go to Step 3,
9:     end if
10: end procedure
```

procedure needs an initial feasible solution to start with. Usually, it is trivial to provide such an initial solution that is arbitrarily far from the optimal one, but it is also possible to apply more sophisticated heuristics to get a better starting point that may help the convergence of the column-generation approach.

2.2 Mixed Integer Linear Programming

In the previous section we dealt with problems in which the variables are continuous, but it is quite common to require integrality of some variables, as they can represent undivisible units or alternative decisions. A *Mixed Integer Linear Program* is given by a model of this type:

$$\begin{aligned} \min \quad & c^T x + h^T y && (2.14)\\ \text{s.t.} \quad & Ax + Dy \geq b && (2.15)\\ & x, y \geq 0 && (2.16)\\ & x \in \mathbb{Z} && (2.17) \end{aligned}$$

When all the variables are required to be integer, we have a pure integer linear program.

In the presence of integer variables, the simplex method can still be applied to the linear relaxation of the problem, that consists simply in ignoring the integrality constraints and considering as feasible region the set $P = \{(x, y) : Ax+Dy \geq b, x, y \geq 0\}$ instead of $Q = \{(x, y) : Ax + Dy \geq b,\ x, y \geq 0,\ x \in \mathbb{Z}\}$. However, the solution we get can be infeasible for the original problem, as there is no guarantee that variables x are integer. This is due to the fact that the simplex method gives as solution the optimal vertex of P, which is not guaranteed to be in Q. As we can see in Figure 2.4a, the optimal solution returned from the linear relaxation is not feasible for the integer problem. In this case, the simplex method can give only a lower bound on the real optimal value of our problem.

In particular cases the optimal solution to a linear problem can be integral: if the feasible region described by the constraints has integer vertices, we are sure that

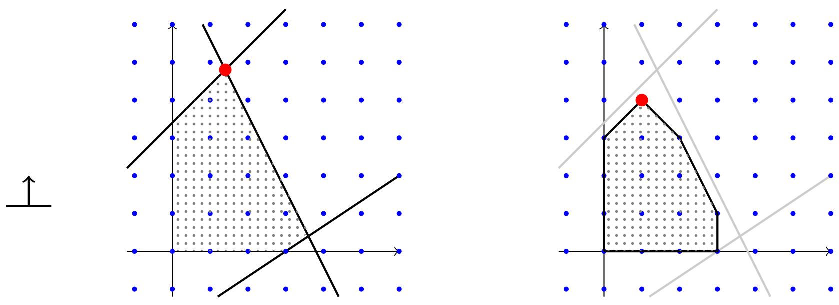


Figure 2.4: Linear relaxation, convex hull and feasible solutions.

solution approaches such as the simplex method return an optimal solution feasible for the integer case.

For example, a kind of matrices that guarantee this feature for the feasible regions they define are the so-called *totally unimodular* matrices, that are characterized by having determinant equal to $+1$ or -1 for each of their square non-singular submatrix. When, on the other hand, the feasible region does not have integer vertices, the integer optimal solution can be found as a vertex of a different polyhedron, as seen in Figure 2.4b, that is given by the set of points obtained as convex combination of two any feasible points of Q. This polyhedron is called the *convex hull* of the feasible points ($\text{conv}(Q)$) and has all vertices in Q, so that solving the linear relaxation of this new formulation we get the feasible optimal solution. Thus, a first attempt at making an effective use of the simplex method also for mixed integer problems consists in providing the formulation of $\text{conv}(Q)$.

We say that an inequality $cx \leq d$ is a *valid inequality* for Q if it is satisfied by every point of Q. We define a face of a polyhedron (in this case $\text{conv}(Q)$) as the set $F = \text{conv}(Q) \cap \{x : cx = d\}$, being $cx \leq d$ a valid inequality for Q. A *facet* is a face of dimension $\dim(\text{conv}(Q)) - 1$. Thus, a formulation of $\text{conv}(Q)$ is given by the set of linear valid inequalities that define its facets. However, this approach is often impractical as there is not a known general method to recover the facets of $\text{conv}(Q)$ once we are given Q.

Thus, an alternative consists in the *cutting plane approach*, that adds suitable inequalities (*cuts*) to the original formulation with the aim of providing a tighter formulation for P, as close to Q as possible. When also this approach is not successful, we proceed with a *branching* approach, that consists in dividing the feasible region into two subsets whose union contains all the feasible mixed integer points. We now see these two methods in details.

2.2.1 Cutting plane approach

Given the set Q, a cut for the set P is an inequality such that, if we add it to the set of constraints defining P, no point of Q is left out while at least one point in P becomes infeasible.

In order to tighten the feasible region given to the simplex method, we need to

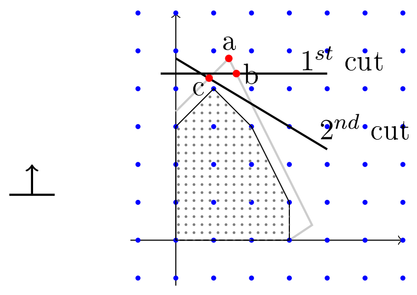


Figure 2.5: Cutting planes algorithm.

find such a cut. The procedure works as shown in Algorithm 2.4. In particular, if the simplex method gives the optimal solution $(\tilde{x}, \tilde{y})$ over the feasible region P and this solution is not feasible for the mixed integer problem, we want to find a cut that is able to separate $(\tilde{x}, \tilde{y})$ from all the points in Q and hence from $\text{conv}(Q)$.

Algorithm 2.4 Cutting Plane method

```
1:  procedure Cutting plane
2:      solve the linear program on $P$ and get the solution $(\tilde{x}, \tilde{y})$,
3:      if $(\tilde{x}, \tilde{y}) \in Q$ then
4:          $(x^{OPT}, y^{OPT}) = (\tilde{x}, \tilde{y})$. STOP.
5:      else
6:          find a cut $\alpha x + \gamma y \leq \beta$ ,
7:          update $P$ to $P \cap \{\alpha x + \gamma y \leq \beta\}$,
8:          repeat from Step 2,
9:      end if
10: end procedure
```

The crucial point of a cutting plane procedure is the separation problem needed to perform Step 6, as there are infinite cuts that separate $(\tilde{x}, \tilde{y})$ and standard routines that provide feasible cuts do not guarantee to return strong valid inequalities, that is, facet-defining inequalities. This approach strongly depends on the formulation of the particular problem.

We illustrate in Figure 2.5 the behaviour of a cutting plane algorithm: we denote with a the first solution obtained by solving the linear relaxation. Then, adding the first cut we can possibly reach b (note that due to the direction of the objective function, there is another suitable vertex to be chosen, but in any case the solution is not integer) and with the second cut we reach c, that is still not integral, so it is possible to keep looking for suitable cuts to be added to te feasible region's description.

Note that the addition of a cut that separates the current optimal solution produces a solution whose value is closer to the optimal one, even if we do not reach the optimal solution itself. It means that we produced a better lower bound.

2.2.2 Branch-and-cut approach

Branching is another way of dealing with integer variables and it is based on a simple idea: suppose x_i is an integer variable that takes a fractional value α in the solution

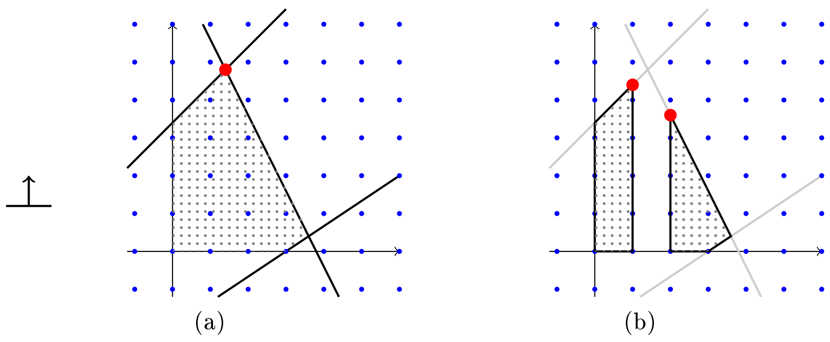


Figure 2.6: branching on an integer variable.

of the linear relaxation. As this makes the solution infeasible for the original problem, we can force x_i to take either a value less or equal to $\lfloor \alpha \rfloor$ or greater or equal to $\lfloor \alpha \rfloor + 1$. This means that we are splitting the feasible region into two polyhedra, whose union contains the optimal integer solution but not the solution of the linear relaxation, as can be seen in Figure 2.6b. At this point, we reoptimize over the two different polyhedra and take the best solution found. Note that while the solution found is integer-infeasible, we keep on branching on different variables.

This recursive procedure can be represented using a tree structure, where each node branches into two nodes, each of which is associated with a portion of the feasible region in which we divided the original one. The solution's value at the parent node is a lower bound (for minimization problems) to the best feasible solution that can be achieved from a particular node downwards. The optimal solution is found at a leaf of this branching tree, as we stop the branching when we get a feasible solution for the mixed integer program. There are other two reasons that allow us to stop the branching: when the problem at that node is infeasible, or when the lower bound at that node is greater than the value obtained by any feasible solution. If one of these three scenarios is found, we can say that we prune the current node, i.e. we are not creating other subtrees starting from it.

Given the importance of computing a good lower bound to prune and reduce the number of nodes to inspect, it can be useful to combine the branching and cutting plane approaches, obtaining the so called *branch-and-cut* algorithm, shown in Algorithm 2.5. In this approach, given a node t that cannot be pruned, we first look for strenghtening cuts, so that we may be able to improve the lower bound at that node and find a feasible optimal solution for the current node. Then, if the solution is still not feasible for the integer problem, we branch and create the child nodes.

Ryan and Foster branching scheme

Introducing constraints on integer variables taking fractional values is the most straightforward way of applying a branching procedure, but we can define a more general approach, in which instead of identifying a fractional variable to be forced to be integer, we recover two inequalities $\alpha x + \beta y \leq \eta$ and $\alpha x + \beta y \geq \sigma$ such that $Q \cap \{(x, y) : \eta < \alpha x + \beta y < \sigma\}$ is empty. Thus, the feasible region Q can be partitioned in the regions

Algorithm 2.5 Branch-and-cut

```
procedure BRANCH-AND-CUT
    consider (2.14)-(2.16) as root node of the branching tree,
    let $P$ be the feasible region, $z^\star$ the current best solution's value,
    while there are unexplored nodes $t$ do
        select one of these nodes $t$,
        if the program in $t$ is infeasible then
            prune $t$,
        else
            solve the linear program in $t$. $\tilde{z}$ is the objective value,
            if $\tilde{z} \geq z^\star$ then
                prune $t$,
            else if $x \in \mathbb{Z}$ then
                prune $t$,
                if $\tilde{z} < z^\star$ then
                    update $z^\star = \tilde{z}$,
                end if
            else
                improve the current lower bound adding cutting planes to $P$,
                if any cut is added, repeat from Step 9
                select a variable $x_i = \alpha$ fractional and create two child nodes:
                    update $P$ to $P \cap \{x_i \leq \lfloor \alpha \rfloor\}$ for the left child,
                    update $P$ to $P \cap \{x_i \geq \lfloor \alpha \rfloor + 1\}$for the right child,
            end if
        end if
    end while
end procedure
```

$Q \cap \{(x, y) : \alpha x + \beta y \leq \eta\}$ and $Q \cap \{(x, y) : \alpha x + \beta y \geq \sigma\}$. A well-known example of such an approach can be found in [45], applied to problems with a set-partitioning structure.

Suppose we have a set of elements $i \in I$, subsets $J \in S \subseteq \mathcal{P}(I)$ and a set-partitioning program

$$\min \quad c^T x \tag{2.18}$$

$$\text{s.t.} \quad \sum_{J \in S} \delta_i^J x_J = 1 \qquad \forall\, i \in I, \tag{2.19}$$

$$x_J \in \{0, 1\} \qquad \forall J \in S \tag{2.20}$$

where δ_i^J is equal to 1 if element i is in subset J and 0 otherwise.

Ryan and Foster [45] noticed that, for any feasible integer solution, a pair of constraints (2.19) can be either fulfilled by the same variable $x_{\bar{J}}$ or by different variables, so that the sum of the values of variables contained in both constraints is integer. In particular, it holds:

Theorem 2.3 ([45]). *In every fractional solution of the set-partitioning linear program,* *there must exists at least a pair of constraints (h, l) for which*

$$0 < \sum_{J \in S} \delta_h^J \delta_l^J x_J < 1$$

Moreover, the contrary also holds: if such sum is integer for any pair of constraints, the solution results to be integer. Thus, the branching strategy they proposed consists in checking whether at the current node there exists such a pair (h, l) and in branching imposing this sum to be either less than or equal to 0 if the constraints are fulfilled by different variables or greater than or equal to 1 otherwise. One node's child is then obtained adding to the current formulation the inequality

$$\sum_{J \in S} \delta_h^J \delta_l^J x_J \leq 0 \tag{2.21}$$

while the other is obtained adding

$$\sum_{J \in S} \delta_h^J \delta_l^J x_J \geq 1. \tag{2.22}$$

Note that these conditions are actually satisfied as equalities, provided we also consider the set-partitioning constraints.

2.2.3 Branch-and-cut-and-price approach

When dealing with a large-scale mixed integer problem in which the number of variables is huge and we need to use a column-generation procedure, applying the branch-and-cut algorithm presented above is not enough to ensure we get the optimal solution. In fact, we recall that the column-generation method ends before generating all the possible variables, so when we branch and we face a new linear program, it is possible that involving some of the missing variables yields a better solution. Thus, the procedure described in the previous section should be slightly modified: we need to ensure that a node is infeasible only if it would be infeasible for the complete program and not only for the RMP, and we need to keep looking for further variables to be added throughout the branching tree.

Given a node of the branching tree, its feasible region is the one of the parent node intersected to the branching constraint. The addition of this constraint can have two effects: it does not interfer with the feasibility of the problem, so that we are still able to find a starting feasible solution from which we look for variables with negative reduced cost, or it can make the problem infeasible. In this second case, we cannot say right away that the node can be pruned by infeasibility, as there can be some missing variables allowed at the node that make the problem feasible again. However, the column-generation method cannot even start, as it requires an initial feasible solution. Such feasible solution has to be recovered and we proceed as follows, exploiting the well-known Farkas lemma, which provides a simple necessary and sufficient condition for the existence of a solution of a system of linear inequalities.

Lemma 2.1 (Farkas' Lemma [16])**.** *A system of linear inequalities $Ax \leq b$ is infeasible if and only if the system $\pi A = 0$, $\pi b < 0$, $\pi \geq 0$ is feasible.*

Although this result is well known, we propose here its proof taken from [16] as it is useful for what follows.

Proof. Assume $\pi A = 0$, $\pi b < 0$, $\pi \geq 0$ is feasible.Then $0 = \pi A x \leq \pi b < 0$ for any x satisfying $Ax \leq b$. Then $Ax \leq b$ is infeasible.

On the other way, assume that $Ax \leq b$ has no solution. Apply the Fourier-Motzkin elimination to $Ax \leq b$ to eliminate all variables $x_1, \ldots, x_n$. It consists in iteratively chosing a variable x_j, partitioning the rows of A according to whether a_i^j is less than, equal to or greater than 0 and then, after normalizing all a_i^j to 1, 0, -1, adding between themself each pair of rows h, k such that $a_h^j = 1$ and $a_k^j = -1$. Then, the new system of inequalities includes all the original ones with $a_i^j = 0$ and all the new ones, that also result in having $a_i^j = 0$, thus in the new system x_j does not appear.
Now, the system obtained $A^0 x \leq b^0$ has the form $0 \leq b^0$ and the system $Ax \leq b$ has a solution if and only if all the entries of b^0 are nonnegative. Since, by hypothesis, that system has no solution, b^0 must have a negative entry, let's say $b_i^0 < 0$. As every inequality of $0 \leq b^0$ is a nonnegative combination of inequalities of the original system, there exists a vector $\pi \geq 0$ such that $0 \leq b_i^0$ is identical to $\pi A x \leq \pi b$. That is, $\pi \geq 0$, $\pi A = 0$, $\pi b = b_i^0 < 0$ is feasible. □

In other words, Lemma 2.1 states that if the RMP is infeasible, then the dual is unbounded (if not infeasible) and for every $\alpha > 0$, $\alpha\pi$ belongs to the feasible region. This means that the vector π is a ray in the dual. Thus, to recover a feasible RMP we want to add a variable with coefficient column a^j such that $\pi a^j > 0$. Such a variable can be found by solving the so-called *Farkas pricing* problem

$$\max_{i \in \{1,\ldots,n\}} \{\tilde{\pi} a_i\} = \min_{i \in \{1,\ldots,n\}} \{-\tilde{\pi} a_i\}, \tag{2.23}$$

where the dual ray $\tilde{\pi}$ can be provided by the simplex method in case of an infeasible program. Note that this results in being the pricing problem (2.13) described in Section 2.1.2 itself, except that we are considering cost coefficients $c_i = 0$ for all $i \in \{1, \ldots, n\}$.

If the Farkas pricing does not find a variable x_i such that $\pi a^j > 0$, we can conclude now that the problem at this node is infeasible. Otherwise, we add to the RMP the variable obtained and repeat the Farkas pricing until we get a feasible RMP. Then, solving the usual $(\overline{PP})$ we recover those variables that where not considered at the parent node but, due to the extra branching constraint, are needed now. Note that, as the RMP at each node possibly contains more variables, the dual has more constraints. So, the PP must be updated at each node to prevent the generation of variables associated to constraints already present in the dual or variables that are forbidden due to the extra branching conditions. This means that the choice of the branching rule in a branch-and-price approach is not separated from the design of the procedure used to solve the pricing problem, that should preserve its performance even after embedding the branching constraints.

2.3 Lagrangian relaxation

It is quite common to incur in Mixed Integer Linear Programs presenting a feasible region that consists of a set of *easy* constraints and a remaining set of *complicating* constraints. This means that, without the complicating constraints, the problem could be easily solved to optimality. Given rational matrices A and E, and rational vectors b and d, consider a program in the form:

$$\min \quad cx \tag{2.24}$$
$$\text{s.t.} \quad Ax \leq b \tag{2.25}$$
$$Ex \leq d \tag{2.26}$$
$$x_i \in \mathbb{Z} \qquad \forall i \in I \subseteq \{1, \dots, n\} \tag{2.27}$$
$$x \in \mathbb{R}^n \tag{2.28}$$

where the system $Ax \leq b$ represents the complicating constraints. A typical example is that of *separable constraints* $Ex \leq d$: the problem would decompose in a number of smaller and independent problems if it were not for the presence of the *linking constraints* $Ax \leq b$. Let $Q = \{x \in \mathbb{R}^n : Ex \leq d, x_i \in \mathbb{Z}, i \in I\}$. We define the *Lagrangian relaxation* as the following problem:

$$(LR(\lambda)) \quad z_{LR}(\lambda) = \min_{q \in Q} cx - \lambda(b - Ax) \tag{2.29}$$

where $\lambda \geq 0$ is a vector of dimension the rows of A. It can be proven that this problem is a relaxation of the original one, in the sense that it can be considered as a way of obtaining a lower bound on the optimal solution's value.

Proposition 2.1 ([16]). *Let z^{OPT} be the optimal solution of (2.24)-(2.28). Then $z_{LR}(\lambda) \leq z^{OPT}$ for every $\lambda \geq 0$.*

Proof. Assume a feasible solution of (2.24)-(2.28) exists and let $\bar{x}$ one of these solutions. Since $\bar{x} \in Q$, $\bar{x}$ is feasible for the lagrangian relaxation and, as $Ax \leq b$ and $\lambda \geq 0$, we have $z_{LR}(\lambda) \geq c\bar{x} + \lambda(b - A\bar{x}) \geq c\bar{x} = z^{OPT}$. □

The tightest upper bound that we can get in this way is given by

$$z_{LD} = \min_{\lambda \geq 0} z_{LR}(\lambda), \tag{2.30}$$

that is called the *lagrangian dual* of the original problem.

The lagrangian dual can be rewritten in the following way, that states how it can be seen as a partial convexification of the original feasible region.

Theorem 2.4 ([16]). *Assume $\{x : Ax \leq b, x \in \text{ conv}(Q)\} \neq \varnothing$. Then $z_{LD} = \max\{cx : Ax \leq b, x \in \text{ conv}(Q)\}$*

Proof. Since $Ex \leq d$ is a rational system, conv(Q) is a rational polyhedron, that we assume is described by the system $Fx \leq g$. Since $Q \neq \varnothing$, by linear duality $z_{LR}(\lambda) = \max\{cx + \lambda(b - Ax) : Fx \leq g\} = \min\{\lambda b + \mu g : \mu F = c - \lambda A, \mu \geq 0\}$ for all $\lambda \geq 0$. It follows that $z_{LD} = \min_{\lambda \geq 0} z_{RL}(\lambda) = \min\{\lambda b + \mu g : \lambda A + \mu F = c, \lambda \geq 0, \mu \geq 0\}$, that is the dual of $\max\{cx : Ax \leq b, Fx \leq g\}$. Their optimal values coicide as the primal is feasible by assumption. □

Let z_{LP} be the optimal value of the linear relaxation of (2.24)-(2.28). It can be proved that the lagrangian dual gives a dominating lower bound.

Corollary 2.1 ([16]). $z^{OPT} \leq z_{LD} \leq z_{LP}$.

Proof. $\text{conv}(\{x \geq 0 : Ax \leq b,\, Ex \leq d, x_i \in \mathbb{Z}\, i \in I\}) \subseteq \text{conv}(Q) \cap \{x \geq 0 : Ax \leq b\} \subseteq \{x \geq 0 : Ax \leq b, Ex \leq d\}$, so maximizing the same objective function cx over these sets gives the statement. □

2.4 Dantzig-Wolfe decomposition

Another possible way of dealing with complicating constraints consists in reformulating the original problem using different variables. We still consider a program in the form (2.24)-(2.28) and the set $Q = \{x \geq 0 : Ex = d, x_i \in \mathbb{Z}, i \in I\}$. Now, let $\{v^k\}_{k\in K}$ be the set of vertices of conv(Q) and $\{r^h\}_{h\in H}$ the set of its extreme rays. Every point in conv(Q) can be written as a convex combination of its vertices plus a conic combination of its extreme rays, meaning that every $x \in \text{conv}(Q)$ can be written as

$$x = \sum_{k\in K} \alpha_k v^k + \sum_{h\in H} \beta_h r^h \quad \text{where} \sum_{k\in K} \alpha_k = 1, \alpha_k \geq 0 \,\forall\, k \in K,\, \beta_h \geq 0 \,\forall\, h \in H.$$

We can thus rewrite the partially convexified problem

$$\max\{cx : Ax \leq b,\, x \in \text{ conv}(Q)\}$$

in the form

$$\begin{aligned} \max \quad & \sum_{k\in K} (cv^k)\alpha_k + \sum_{h\in H} (cr^h)\beta_h && (2.31)\\ \text{s.t.} \quad & \sum_{k\in K} (Av^k)\alpha_k + \sum_{h\in H} (Ar^h)\beta_h \leq b && (2.32)\\ & \sum_{k\in K} \alpha_k = 1 && (2.33)\\ & \alpha \geq 0, \beta \geq 0 && (2.34) \end{aligned}$$

obtaining the formulation for the *Dantzig-Wolfe relaxation*. We can recover a reformulation for the mixed integer problem imposing the integrality conditions

$$\sum_{kinK} v_i^k \alpha_k + \sum_{h\in H} r_i^h \beta_h \in \mathbb{Z} \text{ for } i \in I.$$

Note that this reformulation has the advantage of possibly breaking the complicating structure of the constraints. It has however the drawback of presenting a large number of variables, that makes impractical to consider them all together. A branch-and-price approach is then suitable to solve the reformulated problem.

2.5 Some well-studied optimization problems

In this section we present a selection of well-studied problems that will be useful for building the branch-and-price framework we present in the following chapters, as they have to be solved as subproblems.

2.5.1 Selection problem

Consider a set of elements N, each of them with cost $c_n \geq 0$, $n \in N$ and a collection Σ of subsets of elements of N with a profit of $p_\sigma \geq 0$, $\sigma \in \Sigma$. A selection problem consists in selecting a number of elements in Σ such that the total profit minus the costs is maximum. A formulation for this problem is the following:

$$\max \quad \sum_{\sigma \in \Sigma} p_\sigma y_\sigma - \sum_{n \in N} c_n x_n \tag{2.35}$$

$$\text{s.t.} \quad x_n \geq y_\sigma \qquad \forall\, \sigma \in \Sigma, \forall\, n \in \sigma \tag{2.36}$$

$$x_n,\, y_\sigma \in \{0,1\} \qquad \forall\, n \in N, \forall\, \sigma \in \Sigma \tag{2.37}$$

We build a bipartite network in the following way: consider a source s and a sink t, a first set of nodes representing the subsets $\sigma \in \Sigma$ and the second set representing the elements of N. We connect all nodes of the first set with the source, with directed arcs of capacities p_σ, all other nodes with the sink, with directed arcs of capacities c_n and put arcs with infinite capacity between pairs (σ, n) if $n \in \sigma$.

Given a set of nodes S, we define $\delta^+(S)$ the subset of those arcs that have tail in S and head outside S. An *s-t cut* is a subset of arcs Γ of the described network such that, given a subset of nodes S that contains the source s but not the sink t, $\Gamma = \delta^+(S)$. The *capacity* of the cut is given by the sum of the capacities of the arcs in Γ. The *minimum cut problem* aims at finding the cut that has minimum capacity.

In [4] an explanation on how a selection problem can be solved as a minimum cut problem on the described bipartite network has been provided. The following result holds.

Lemma 2.2 ([4]). *There is a one-to-one correspondence between cuts containing no arcs with infinite capacities and selections.*

But we can say something better:

Lemma 2.3 ([4]). *The minimum cut corresponds to the selection of maximum value.*

Proof. Consider a minimum cut that partitions the set of nodes into the two subsets (Γ, T) and $(\bar{\Gamma}, \bar{T})$ and let (Γ', T'), $(\bar{\Gamma}', \bar{T}')$ be any other cut. As no arc of infinite capacity can belong to the cut, the values of the cuts are related by

$$\sum_{\sigma \in \bar{\Gamma}} p_\sigma + \sum_{n \in T} c_n \leq \sum_{\sigma \in \bar{\Gamma}'} p_\sigma + \sum_{n \in T'} c_n$$

As $\sum_{\sigma \in \Gamma} p_\sigma + \sum_{\sigma \in \bar{\Gamma}} p_\sigma = \sum_{\sigma \in \Gamma'} p_\sigma + \sum_{\sigma \in \bar{\Gamma}'} p_\sigma$ we have that the associated values for the selections are

$$\begin{aligned}\sum_{\sigma \in \Gamma} p_\sigma - \sum_{n \in T} c_n &= \sum_{\sigma \in \Gamma'} p_\sigma + \sum_{\sigma \in \bar{\Gamma}'} p_\sigma - \sum_{\sigma \in \bar{\Gamma}} p_\sigma - \sum_{n \in T} c_n \geq \\ &\geq \sum_{\sigma \in \Gamma'} p_\sigma + \sum_{n \in T} c_n - \sum_{n \in T'} c_n - \sum_{n \in T} c_n = \\ &= \sum_{\sigma \in \Gamma'} p_\sigma - \sum_{n \in T'} c_n\end{aligned}$$

that proves the lemma. □

This proves that it is possible to solve selection problems by solving an associated minimum cut problem, that is equivalent to a maximum flow problem, for which we have strong combinatorial algorithms as Ford-Fulkerson methods.

Maximum flow/minimum cut on networks

Given a digraph (V, D), made of a set of nodes V and directed arcs D, and given two nodes s, t, an *s-t flow* is a non-negative assignment of values x_a for each $a \in D$ such that the amount of flow entering a node equals the flow that exits from that node, except for the *source* s and the *sink* t. Thus, any flow satisfyes the equalities

$$\sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = 0 \quad \forall\, v \in V \backslash \{s, t\}$$

where we denote with $\delta^-(v)$ the set of directed arcs having node v as head and $\delta^+(v)$ the set of nodes having it as tail. From these constraints, we derive that the total amount of flow leaving from s is the same entering the sink t.

Moreover, we can associate to each arc a capacity c_a, $a \in D$ that fixes the maximum amount of flow that can go through the arc. We define then the maximum flow problem as that of finding the maximum value of an s-t flow $f(x)$ that does not exceed the arc capacity, with the following formulation:

$$f(x) = \max \quad \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a \tag{2.38}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = 0 \qquad \forall\, v \in V \backslash \{s, t\} \tag{2.39}$$

$$0 \le x_a \le c_a \qquad \forall a \in D \tag{2.40}$$

Note that, if the capacities are all integer numbers, the matrix describing the program constraints is totally unimodular, so that to solve the integer flow problem it simply suffices to solve its linear relaxation.

On the same network we also look for a cut of minimum capacity. In particular, the following result holds:

Theorem 2.5 (Max-Flow Min-Cut Theorem [16]). *Given a digraph (V, D), two nodes s and t and capacities c_a, $a \in D$,*

$$\max\{f(x) : x \textit{ is a feasible s-t flow}\} = \min\{c(\Gamma) : \Gamma \textit{ is an s-t cut}\}$$

We want to solve the flow problem using an algorithm that exploits the combinatorics behind the flow network. We introduce a new network, the *residual graph* $(V, \bar{D})$ associated to an s-t flow, that has the same set of nodes of the original network and where the set of arcs is determined starting from the flow. Let $a \in D$ be an arc going from node $u \in V$ to node $v \in V$, then we denote with $\bar{a}$ the arc going from v to u. Thus, $\bar{D}$ is defined as the set of arcs

$$\{a \in D : x_a < c_a\} \cup \{\bar{a} : a \in D, x_a > 0\}.$$

It can be proved that a feasible flow is maximum if and only if the residual graph contains no path going from s to t. If such a path exists, it is called an *augmenting*

path, as it can be used to increase the total flow and improve the current solution. This fact provides the idea underlying the well known *Ford-Fulkerson methods* for solving a maximum flow / minimum cut problem.
Note that there are several ways of choosing an augmenting path, one of the easiest consists in looking for such a path on the residual graph. This choice proves to be useful to recover the solution for the minimum cut problem, that can be done looking at the residual graph: one optimal subset of nodes S is given by all the nodes that can be reached from the source s, and then the optimal solution is obtained selecting the arcs in $\delta^+(S)$.

2.5.2 Knapsack problem

This classical problem in Integer Programming has been widely studied both because of its simple structure, that allows to exploit several combinatorial properties, and because it often appears as subproblem in other difficult problems. A complete survey on the structure of the problem, that results to be weakly *NP-hard*, and solution methods can be found in [37]. Given a knapsack of fixed capacity C and n types of objects with integer weights $p_i \leq C$ and utility c_i, with $i \in \{1,\dots,n\}$, the problem consists in maximizing the total utility of the objects we can pack into the knapsack without exceeding the capacity C. The straightforward formulation for this problem is:

$$\max \quad \sum_{i=1}^{n} c_i x_i \tag{2.41}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} p_i x_i \leq C \tag{2.42}$$

$$x \in \mathbb{Z}^n \tag{2.43}$$

where x_i records how many units of object i can be loaded.

Important variants to this model include the case in which at most one item of each type can be selected, leading to the formulation of the 0/1 knapsack problem, where the feasible region becomes

$$x \in \{0,1\}^n.$$

It is quite trivial to solve the linear relaxation of this problem: it suffices to reorder the items in an increasing way according to $\frac{p_i}{c_i}$ and then set $x_i = 1$ for $i \in \{1,\dots,k\}$ in the new order, where k is the index such that

$$\sum_{i=1}^{k} p_i \leq C \quad \text{but} \quad \sum_{i=1}^{k+1} p_i > C,$$

and $x_{k+1} = \frac{C-\sum_{i=1}^{k} p_i}{p_{k+1}}$.

From this we can recover a straightforward greedy algorithm that gives an approximation of the optimal integer solution taking as the optimal value $\tilde{z}$ the maximum between $\sum_{i=1}^{k} c_i$ and c_{k+1} and respectively the solution obtained setting $x_i = 1$ for $i \in \{1,\dots,k\}$ or $x_{k+1} = 1$.

Notice that, supposing $c_i = 1$ for all $i \in \{1, \dots n\}$, when applying the greedy algorithm, the variables result ordered increasingly with respect to their weights. In this case, the solution obtained with the greedy algorithm is also the optimal solution, given by $x_i = 1$ for $i \in \{1, \dots, k\}$. On the other hand, if this condition on the coefficients $c_i, i \in \{1, \dots, n\}$ does not hold, the greedy algorithm described above is a 2-approximation for the knapsack problem [30].

2.5.3 Set-partitioning and set-covering problems

Set-partitioning and set-covering constraints are classical families of constraints that aim at subdividing sets of elements into subsets satisfying specific requirements. Consider a set N of elements $N = \{1, \dots, n\}$ and a family of subsets of N, $\mathcal{F} = \{F_1, \dots, F_m\}$. The set-partitioning constraints aim at selecting among $\mathcal{F}$ subsets such that each element in the set N belongs to exactly one subset. They are defined as follows:

$$\sum_{\substack{j=1 \\ i \in F_j}}^{m} x_j = 1 \quad \forall i \in N \tag{2.44}$$

where x_j, $j \in \{1, \dots, m\}$ are variables that assume value 1 if we select the subset F_j, 0 otherwise. The set -covering constraints, instead, identify a collection of subsets in $\mathcal{F}$ such that each element belongs to at least one subset. Formally, considering the same variables as above, they are defined as:

$$\sum_{\substack{j=1 \\ i \in F_j}}^{m} x_j \geq 1 \quad \forall i \in N \tag{2.45}$$

Introducing costs $c_j \geq 0$ on the subsets F_j, we can formulated the associated set-partitioning problem:

$$\begin{aligned}
\text{(SP)} \ \min \quad & \sum_{j=1}^{m} c_j x_j && && (2.46)\\
\text{s.t.} \quad & \sum_{\substack{j=1 \\ i \in F_j}}^{m} x_j = 1 && \forall i \in N && (2.47)\\
& x_j \in \{0,1\} && \forall j \in \{1, \dots, m\} && (2.48)
\end{aligned}$$

and set-covering problem:

$$\begin{aligned}
\text{(SC)} \ \min \quad & \sum_{j=1}^{m} c_j x_j && && (2.49)\\
\text{s.t.} \quad & \sum_{\substack{j=1 \\ i \in F_j}}^{m} x_j \geq 1 \quad \forall i \in N && && (2.50)\\
& x_j \in \{0,1\} && \forall j \in \{1, \dots, m\} && (2.51)
\end{aligned}$$

Note that, as the set $\mathcal{F}$ can possibly contain all possible subsets of elements of N, we have an exponential number of elements F_j and, thus, of variables x_j. A column-generation method is then a suitable approach to deal with the huge number of variables and find the solution of the linear relaxation of both (SP) and (SC), to be embedded in a branch-and-price approach. Recall that in Section 2.2.2 we presented Ryan-Foster branching strategy for set-partitioning constraints.

In [5] it is shown how, in many applications that require a set-partitioning structure, it can be advisable to consider instead a set-covering formulation. Depending on the problem addressed, in fact, it may be straightforward to recover the optimal solution to the set-partitioning problem from the optimal set-covering one. Besides, the set-covering linear relaxation is usually numerically more stable.

2.6 Implementing tools: SCIP and CPLEX

SCIP (Solving Constraint Integer Programs) is a framework for solving constraint integer programs and has been developed since 2001 at the ZIB (Zuse-Institute Berlin) [1, 24]. It provides a very flexible environment for solving branch-and-bound based search algorithms, integrating together four different methodologies in optimization: *Mixed -Integer Linear Programming*, *Satisfiability Solving*, *Constraint Programming* and *Mixed Integer Non-linear Programming.* SCIP has a plugin-based design, that consists in external user defined callback objects that interact with the main framework. In this way, it is possible to implement:

1. *variable pricers* to dinamically create problem variables,
2. *cut separators* to apply cutting planes to the LP relaxation: some well known families of cuts, such as mixed integer rounding cuts, Gomory mixed integer cuts or clique cuts are already implemented, but also user-defined cuts can be added,
3. *primal heuristics* to search for feasible solutions, so that we can possibly prune branches of the search tree by bounding,
4. *node selectors* to guide the search,
5. *branching rules* to split the problem into subproblems, where an arbitrary number of children per node can be created,
6. *constraints handlers* for dealing with user-defined constraints: together with a general linear constraints handler, we can find specific ones, as the knapsack constraint or set-partitioning constraint handlers, that can exploit the special structure of these constraints,
7. and many others.

We want to point out, moreover, that SCIP is one of the fastest non-commercial solvers for mixed integer programming and mixed integer nonlinear programming, as shown in Figure 2.7, and that its performance can be improved by using CPLEX as solver instead of the default, non-commercial SoPlex.

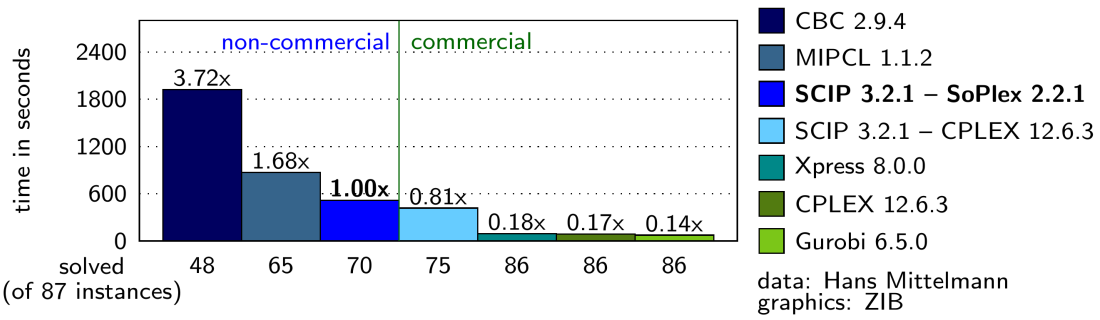


Figure 2.7: SCIP performance compared with commercial and non-commercial solvers [24].

IBM ILOG CPLEX Optimizer [18] provides flexible and high-performance solvers for linear, mixed integer, quadratic and quadratically constrained programming problems.

CPLEX allows to choose between different algorithms that are used to solve linear or quadratic programs with continuous variables:

1. *Simplex Optimizer*: it includes primal method, dual method and a network optimizer especially suited for pure network problems,

2. *Barrier Optimizer*: it provides an alternative to the simplex method and it offers a particularly efficient approach on large, sparse problems (for example, more than 100 000 rows or columns, and no more than perhaps a dozen nonzeros per column) and sometimes on other models as well,

3. *Sifting Optimizer*: it was developed to exploit the characteristics of model with large aspects ratio (that is, a large ratio of the number of columns to the number of rows). In particular, the method is well suited to large aspect ratio models where an optimal solution can be expected to place most variables at their lower bounds.

The *Mixed Integer Optimizer* instead provides fast and robust solutions to the most difficult mixed integer programs using a branch and bound technique integrated with cutting-edge strategy.

CHAPTER 3

State of the Art

The Haplotype Inference by Pure Parsimony (HIPP) problem we defined in Chapter 1 of this thesis is a particular case of a more general problem called Haplotype Inference, that consists in finding a set of haplotypes that is able to resolve a set of given genotypes. In the literature we find different problems related to it: the shared aim is that of finding haplotypes that resolve the given genotypes, but they eventually consider different extra conditions on the haplotypes. For example, the Perfect Phylogeny problem requires in addition that the haplotypes selected have an evolutionary history that can be represented by an appropriate rooted tree. Alternatively, the HIPP problem requires the set of haplotypes to have the minimum cardinality. In this chapter we review the different versions of the HIPP problem and the different solution approaches, including both exact methods and heuristics.

3.1 Haplotype Inference in literature

3.1.1 The Maximum Resolution Problem

Note that, given a set of m genotypes G, it is always possible to find a feasible solution to the Haplotype Inference problem that consists at most in $2\,m$ haplotypes. The first version of the Haplotype Inference problem that has been introduced in literature is the *Maximum Resolution Problem (MRP)* [15]. It does not give any particular constraint on the features that the involved haplotypes should have, but it simply aims at finding the maximum number of genotypes in the set G that can be resolved following the Clark's Inference rule, proposed in [15] and sketched in Algorithm 3.6. First, the procedure identifies the genotypes in the given set that contain at most one heterozygous site: these genotypes are trivially resolved, as they allow at most one possible pair of haplotypes resolving them, that are referred to as fixed haplotypes (see Definition 1.7). The set of haplotypes H used to resolve genotypes in G is initially set to contain these fixed haplotypes. Then, given a genotype with at least two heterozygous sites, the rule tries to resolve it using at least one haplotype in H and deriving its complementary. If this latter haplotype is not already contained in H, it adds it to H

and proceeds to another genotype. The application of this rule to all the genotypes considered in a predefined order yields a procedure that gives a feasible solution to the MRP.

Algorithm 3.6 Clark's heuristic [15]

```
1: procedure CLARK(G)
2:     let $\bar{G} = G$ be the set of unresolved genotypes,
3:     let $H = \varnothing$ be the set that contains the haplotypes considered so far,
4:     for $g$ in $\bar{G}$ do
5:         if $g$ is fixed then
6:             add it to $H$ as fixed haplotypes and remove it from $\bar{G}$,
7:         end if
8:     end for
9:     for $g$ in $\bar{G}$ do
10:        look for a haplotype $h \in H$ compatible with $g$,
11:        if $h$ is found then
12:            compute the complementary one $\bar{h}$ w.r.t. $g$,
13:            if $\bar{h} \notin H$ then and add it to $H$,
14:            end if
15:            delete $g$ from $\bar{G}$
16:        end if
17:    end for
18: end procedure
```

If, at the end of the algorithm, $\bar{G}$ is empty, there are no unresolved genotypes and the haplotypes in set H constitute an optimal solution of the MRP. There is no guarantee that this algorithm is able to resolve all the genotypes as, for example, it cannot even start if no fixed haplotypes are found. This event, however, is assumed to have a very low probability when dealing with realistic samples of human DNA. More likely, it can happen that only a portion of the genotypes is resolved by the set H obtained. Notice that the set H can sensibly vary according to the order chosen for the genotypes. The algorithm should then be applied considering every possible ordering, so that it reaches better solutions.

It is proved that this problem is NP-hard and Maximization Strict-NP complete (Max-SNP complete), meaning that there is a fixed limit on the accuracy of every polynomial-time approximation method for this problem [27].

3.1.2 Perfect Phylogeny Haplotyping

As we already pointed out, the MRP does not take into account any biological assumption, that are useful in leading the algorithms to build a plausible solution. On the contrary, the Perfect Phylogeny Haplotyping (PPH) aims at resolving the m genotypes contained in a set G taking into account two extra assumption on the nature of the sought haplotypes. One of these is the assumption, considered for example in [26], according to which the evolutionary history of $2\,m$ haplotypes (two for each genotype) can be represented by a *coalescent*, a rooted tree in which the root is labeled with a 0-vector of length n, where n is the number of considered SNPs, the leaves represent haplotypes and the edges between two nodes represent the switch of a site value from 0 to 1. In this way, the unique path from the root to one leaf shows exactly which sites have value 1. The *infinite-site* assumption is also made, which states that, due to the relatively small time horizon of interest considered, at each site at most one mutation occurs. The link between this representation of $2m$ haplotypes and a solution to the Haplotype Inference problem lay in the fact that the haplotypes can be seen as the single chains that, once paired, resolve the m genotypes we are considering.

The two given assumptions can be summarized saying that the $2m$ haplotypes can be explained by a *perfect phylogeny*:

Definition 3.1 ([26]). *Given a $2m \times n$ matrix in $\{0, 1\}$, a* perfect phylogeny *for B is a rooted tree T with exactly $2m$ leaves that obeys the following properties:*

1. *each of the $2m$ rows labels exactly one leaf of T,*
2. *each of the n columns labels exactly one edge of T,*
3. *every interior edge (one not adjacent to a leaf) of T is labeled by at least one column,*
4. *for any row i, the columns that label the edges along the unique path from the root to leaf i specify the columns of B that have a value of one in row i in B. In other words, that path is a compact representation of row i.*

At this point, finding a solution of the Haplotype Inference problem is equivalent to finding a perfect phylogeny and a suitable pairing of the leaves such that we can recover every genotype. As it is quite trivial to recover a suitable pairing once we are given the $2m$ haplotypes, the main goal becomes finding the perfect phylogeny.

It is easy to see how these assumptions restrict the number of possible solutions, as shown by the following example that can be found in [26]:

Example 3.1. *Given the genotypes $g_1 = \{22\}$, $g_2 = \{02\}$, $g_3 = \{10\}$, we consider the two different solutions*

$$H' = \{a = \{10\}, a' = \{01\}, b = \{01\}, b' = \{00\}, c = \{10\}, c' = \{10\}\}$$

$$H'' = \{a = \{11\}, a' = \{00\}, b = \{01\}, b' = \{00\}, c = \{10\}, c' = \{10\}\}$$

and note that the second one does not have a perfect phylogeny, as shown in Figure 3.1, because point 2 in the definition of a perfect phylogeny is not satisfied.

The approach using a perfect phylogeny is carried on in [3], which presents an algorithm that determines if there exists a suitable solution for the PPH problem given a set of genotypes and, if this is the case, it yields a data structure containing all the possible solutions. This algorithm is proven to have a $O(nm^2)$ complexity. It is also proved how the number of possible solutions to the PPH is significantly reduced with respect to the Haplotype Inference problem.

In [22], another $O(nm^2)$-time algorithm is presented and the implementation is extended to deal with instances that present some errors in the data, so that it would not be possible to find a solution that exactly fit the perfect phylogeny model, or where some SNP values related to some individuals are missing. A linear-time algorithm for the solution of this problem is presented in [20].

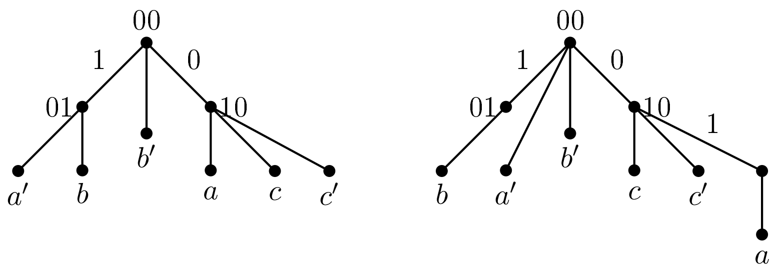


Figure 3.1: An example of solutions with (left) and without (right) perfect phylogeny.

3.1.3 Other Haplotype Inference principles

In [10] we can find a small overview of the use of statistical methods for solving the Haplotype Inference Problem. The main idea underlying this approach is that there is an unknown distribution of the haplotypes among the target population, and each genotype is simply a combination of a pair of randomly chosen haplotypes. Thus, we want to estimate the haplotype frequencies and then infer haplotypes according to these frequencies. Two main approaches have been proposed. The first one in [23] implements an estimation-maximization algorithm to obtain maximum-likelyhood estimates for haplotype frequencies. In practice, starting from an initial arbitrary distribution that represents the unknown true frequencies, an iterative estimation of the haplotype frequencies leads to the most likely ones as convergence is reached. It is possible to choose different starting frequencies. Another approach, presented for example in [38], makes use of a Monte Carlo method after dividing the haplotypes in small fragments. This model proved to be robust to the presence of missing data and to occurrencies of recombination hotspots.

3.2 Haplotype Inference by Pure Parsimony

A classical biological assumption that is widely taken into account in haplotype inference consists in the *parsimony* principle, according to which it is very unlikely that in Nature we find more haplotypes than the necessary ones. In fact, in natural populations the number of haplotypes observed is sensibly smaller than the number of possible ones, thus suggesting that, with a large enough sample of genotypes, we can likely deduce the right haplotypes when considering their minimum possible number.

It has been proved that if each genotype has at most two heterozygous sites, then the HIPP can be solved in polynomial time [32]. On the contrary, the problem is well-known to be not only *NP-hard*, but also *APX-hard* [31]. This means that there exists a constant $\lambda > 1$ such that the existence of a λ-approximation algorithm for HIPP problem would imply P=NP.

The proof for the APX-hardness is made in two steps and exploits a reduction from the *Node-Cover* problem, known itself to be APX-hard (see for example [6]). Note that the proof involves only instances in which each genotype has at most three heterozygous sites, as they are already enough to prove the hardness of the HIPP problem. We define this problem as:

Definition 3.2 (Node-Cover (NC) Problem). *Let $G = (V, E)$ be an undirected simple graph on n nodes and m edges. A Node-Cover is a vertex set $X \subseteq V$ such that every edge in E has at least one endpoint in X.*

We define as *NC2* the Node-Cover problem restricted to those graphs that yield an optimal solution of size at least $n/2$. In [6] is proved that NC2 is APX-hard.

Lemma 3.1 ([6]). *The problem NC2 is APX-hard.*

The proof of this lemma proceeds reducing the APX-hard problem NC to the problem NC2. This can be done via a polynomial algorithm [6] that given a graph $G = (V, E)$ returns an instance $G' = (V', E')$ of NC2 and a set Q such that, if X is a minimum node cover of G', then $X \cup Q$ is a minimum cover of G.

Lemma 3.2 ([31]). *The HIPP problem is APX-hard*

To prove this result a reduction to an APX-hard problem is used too, in particular the HIPP problem is reduced to the NC2 problem. As a result of this complexity, each exact algorithm for the problem will use some sort of enumeration [31].

Different exact formulations have been proposed for HIPP. We can mainly divide them into two different types: those which have an exponential size on the input data, for example having an exponential number of constraints or variables, or both, and those that have a polynomial size. Finally, to be able to get a good feasible solution on bigger instances within a small time orizon, some heuristic algorithms have been proposed. In the following three sections we make an excursus on these different types of models and related solution methods.

3.3 Exponential-size integer formulations

In this section we present several state-of-the-art models for HIPP whose formulations present an exponential number of variables and constraints.

3.3.1 An intuitive formulation

In 2003, Gusfield [25] proposed an intuitive model for the HIPP problem. Although the formulation is easy and understandable, it becomes of impractical use with the increase of the instances' size, as it has an exponential number of both variables and constraints, and no decomposition method to deal with them has been proposed. Consider a single genotype $g^i, i \in I = \{1, \ldots, m\}$, which presents k_i heterozygous sites. It is known that there are $2^{k_i - 1}$ different pairs of haplotypes which resolve that genotype and each of these pairs can be associated to an index $j \in J_i = \{1, \ldots, 2^{k_i - 1}\}$. Thus, the formulation uses a binary variable $y_{i,j}$ that is equal to 1 if the j-th pair is used to resolve genotype g_i. Moreover, being M the number of haplotypes that are compatible with at least one genotype, it is possible to associate to each haplotype involved in any genotype's resolving pair a variable x_h, $h \in H = \{1, \ldots, M\}$ that records if haplotype h is used in the solution. Thus, we denote with *TIP* the formulation proposed for

HIPP, that can be written as:

$$
\begin{aligned}
(TIP)\ \min\ & \sum_{h\in H} x_h && && (3.1)\\
\text{s.t.}\ & \sum_{j\in J} y_{i,j} = 1 && \forall\, i\in I && (3.2)\\
& y_{i,j} \le x_h && \forall\, i\in I, \forall\, j\in J_i, \forall\, h\in H:\ \text{it belongs to pair } j && (3.3)\\
& y_{i,j} \in \{0,1\} && \forall\, i\in I, \forall j\in J_i && (3.4)\\
& x_h \in \{0,1\} && \forall\, h\in H && (3.5)
\end{aligned}
$$

We are here minimizing the total number of haplotypes used, subject to constraints (3.2), that force every genotype to be resolved by a pair of haplotypes, and constraints (3.3), that link variables x_h to the corresponding $y_{i,j}$ variables, so that if a pair indexed by j is used to resolve a genotype, both its haplotypes are counted in the objective function.

From the formulation we can see that, for each genotype g^i, 2^{k_i-1} variables and $2^{k_i-1}+1$ constraints are needed, so that defining $k=\max\{k_i, i\in I\}$ we have an overall of $O(m\,2^{k-1})$ variables and $O(m(2^{k-1}+1))$ constraints, that are in fact exponential on the number of heterozygous sites.

In [25] it is also shown how the actual number of variables and constraints can be reduced, taking into account two simple facts. First, if a pair j related to genotype g^i consists of two haplotypes compatible only with this genotype, we can avoid to add the corresponding variables $y_{i,j}$ and x_h associated to the two haplotypes in j. If it happens that all pairs related to a genotype are removed, it means that we can remove it from the set of genotypes after arbitrarily choosing a pair of haplotypes for it. Otherwise, an optimal solution will always involve one of the pairs left for each genotype. Although this reduction at first requires a complete enumeration of all possible haplotypes involved, preserving the impracticability of the formulation, we can avoid the complete enumeration considering directly only those haplotypes that are compatible with at least two genotypes. That is, instead of building pairs of haplotypes looking at one genotype at a time, we look at a pair of genotypes at a time.

The second way of reducing the number of variables and constraints takes into account the presence of genotypes with no heterozygous site. In this case, the only possible pair is made of two copies of the same haplotype, that is equal to the genotype itself. This means that we remove the homozygous genotype from our problem and the variable corresponding to the associated haplotype from the objective function. The new model that includes these expedients is called *RTIP* formulation.

Anyway, the size of this formulation still increases exponentially with the number of heterozygous sites of the genotypes, thus only relatively small size instances (up to 50 genotypes and 30 SNPs) has been solved.

3.3.2 A set-covering approach.

The statement of the HIPP problem reminds the idea of a set-covering and thus a model can be formulated involving a covering condition (2.45) as seen in Section 2.5.3. We define triples (g^i, p, a) where g^i is a genotype, p is a position of the genotype such that $g_p^i = 2$ and $a \in \{0,1\}$. In the approach presented in [33] the haplotypes' solution set is seeked among the sets of haplotypes for which every triple is *covered* by one of those haplotypes, that means there exists a haplotype h in that set such that it is compatible with g^i and $h_p = a$. This condition, however, is not enough to obtain a feasible solution for the problem, as can be seen in the following example.

Example 3.2 ([33]). *Consider the set of genotypes $G = \{1222, 2122, 2212, 2221\}$. The set of haplotypes $H = \{0111, 1011, 1101, 1110\}$ satisfies the covering condition but is not a valid solution for the HIPP problem as, for example, we cannot find any two haplotypes in H that resolve the first genotype in G.*

The example shows that, besides the constraints regarding the set-covering formulation, we need also other constraints to ensure that every genotype is resolved by two haplotypes. This can be formulated in the following way: for each genotype g^i and set of haplotypes that is insufficient for resolving g^i, there must be at least another haplotype in the solution that is compatible with g^i.
Define the sets H as all the haplotypes that can resolve at least one genotype in G, H' any set of haplotypes insufficient to resolve all the genotypes, $\mathcal{U}(H')$ the set of genotypes unresolved by H' and $H(g^i)$ the haplotypes compatible with g^i. Moreover, define $C(g^i, H') := H(g^i) - H'$ and $\mathcal{N}'$ the set of pairs (g^i, H') such that H' is an insufficient set of haplotypes and $g^i \in \mathcal{U}(H')$.
The formulation for the HIPP problem given in [33] and denoted as *Set Covering Model (SCM)* is the following:

$$
\begin{aligned}
(SCM)\ \min \quad & \sum_{h \in H} x_h && && (3.6)\\
\text{s.t.} \quad & \sum_{h \in H_p^a(g^i)} x_h \geq 1 && \forall\, i \in I, \forall\, p : g_p^i = 2, a \in \{0,1\} && (3.7)\\
& \sum_{h \in C(g^i, H')} x_h \geq 1 && \forall\, (g^i, H') \in \mathcal{N}' && (3.8)\\
& x_h \in \{0,1\} && && (3.9)
\end{aligned}
$$

where $H_p^a(g^i)$ is the set of haplotypes compatible with g^i that has value a in position p.
As we have an exponential number of constraints of type (3.8), we apply a cut-generation procedure in order to add a constraint only if it is necessary. It is easy to see that, in the worst case, just considering the formulation (3.6)-(3.7) we already have $O(2^n)$ variables and $O(m\,n)$ constraints, thus a column-generation approach is necessary to avoid generating all the variables. In [33] the pricing problem, that looks for the best variable to be added by enumerating all the possible haplotypes, is solved in a particular way, presented in the following paragraphs.

Consider $\mathcal{N} \subset \mathcal{N}'$ the subset of constraints (3.8) that have been actually added and $\mathcal{X}$ the set of variables generated at some iteration of the column-generation procedure. Let $(\alpha^\star, \beta^\star, \gamma^\star)$ be the dual values associated respectively to constraints (3.7) with $a = 0$, $a = 1$ and the $|\mathcal{N}|$ constraints (3.8). Looking at the dual of SCM, we build the pricing problem by maximizing

$$\sum_{g \in G(h)} \sum_{p: g_p = 2} [(1 - h_p)\alpha^\star_{g,p} + h_p \beta^\star_{g,p}] + \sum_{(g,H) \in \mathcal{N}, g \in G(h)} \gamma^\star_{g,H} \tag{3.10}$$

over all the possible h. If this maximum is greater than 1, the variable associated to the haplotype found in the solution has to be added to the master problem.

As it is not advisable to consider every h haplotype compatible with at least a genotype in G, in order to reduce the enumeration effort, the structure of the pricing problem is inquired taking into account the following definitions.

Definition 3.3 (Selectable clique [33])**.** *A* selectable-clique*, or* s-clique*, is a subset K of genotypes such that there exists a haplotype h that is compatible with every genotype in K and no other genotype in $G \backslash K$.*

We describe in Algorithm 3.7 the procedure proposed in [33] identifying all the s-cliques of a given set of genotypes G. Starting from a root node, it builds a tree in which each leaf represents an s-clique and each branch fixes uniquely the value of a SNP. Each node t of the tree is identified by a string with values in $\{0, 1, 2\}$, called virtual genotype g^t, a set of genotypes $G_t \subset G$ compatible with it and a set of positions $I_t \subseteq P$ that defines the set of values of the virtual genotype that have been fixed through the branchings. A node is pruned if for all the genotypes in G_t associated to a particular node and all positions $p \in T_t$ it holds that $g_p = g^t_p$. An example of such a tree, obtained applying Algorithm 3.7 to the set of genotypes $G = \{2202, 1222, 1221, 2012, 0210\}$, is shown in Figure 3.2. Note that the size of such a tree is highly dependent on the order

Algorithm 3.7 Find the set of s-cliques of G [33]

```
 1: procedure FIND S-CLIQUES(G)
 2:     label the root node with a virtual genotype of 2s,
 3:     initialize $t = 0$, $I_t = \varnothing$ the set of already branched positions,
 4:     while there are unexplored nodes do
 5:         select node $t$,
 6:         if for all $p \in P = \{1, \dots, n\} \backslash I_t \rightarrow g^t_p = g_p \, \forall \, g \in G_t$ then
 7:             the node represents an s-clique. Prune the node,
 8:         else
 9:             choose $\bar{p}$ that fails this condition and branch:
10:             - one branch $t + 1$ has virtual genotype with 0 at position $\bar{p}$,
11:             - the other $t + 2$ has value 1,
12:             update $I_{t+1} = I_{t+2} = I_t \cup \{\bar{p}\}$,
13:         end if
14:     end while
15: end procedure
```

in which we choose the position $\bar{p}$ for branching. Every leaf of the tree is associated to a pattern, that is an n-string over $\{0, 1, -\}$ that represents its virtual genotype. A haplotype compatible with all the genotypes in an s-clique is called a selector and is derived from any pattern just replacing each $-$ with a 0 or 1. The same s-clique can appear in different leaves, so more than one pattern is associated to the same s-clique. A list of all these patterns for the same s-clique is a *pattern table*.

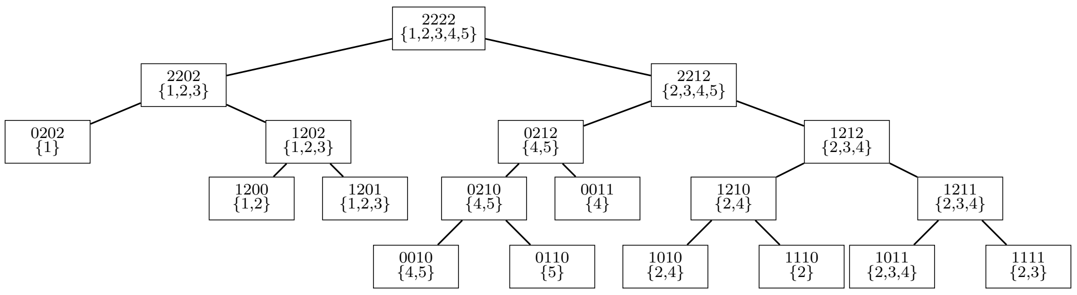


Figure 3.2: Tree of all s-cliques. G contains $g^1 = 2202$, $g^2 = 1222$, $g^3 = 1221$, $g^4 = 2012$, $g^5 = 0210$

Definition 3.4 (Pattern table in standard form [33]). *A pattern table is in* standard form *if:*

- *no pattern is contained in another pattern (that is, given a pattern π^i and a position a, there is not another pattern π^j such that $\pi^i_a = \pi^j_a$ for all a with $\pi^i_a \in \{0, 1\}$)*
- *for each pair of patterns π^i and π^j, there exist at least two positions a, b such that $\pi^i_a \neq \pi^j_a$ and $\pi^i_b \neq \pi^j_b$*

Once we have a pattern table, a compact representation of it is obtained putting it in standard form.

Thus, going back to the pricing problem, we can rewrite for each s-clique K:

$$\alpha(K,p) = \sum_{g \in K :\, g_p = 2} \alpha^\star_{g,p}$$

$$\beta(K,p) = \sum_{g \in K :\, g_p = 2} \beta^\star_{g,p}$$

$$\gamma(K) = \sum_{(g,H) \in \mathcal{N}, g \in K} \gamma^\star_{g,H}$$

$$\lambda(K,h) := \sum_p [(1 - h_p)\alpha(K,p) + h_p \beta(K,p)] + \gamma(K)$$

$$\lambda(K) := \max_{h \in H(K) \setminus \mathcal{X}} \lambda(K,h)$$

where $\lambda(K)$ is the maximum value assumed by the pricing problem's objective function within the selectors of K. At this point, in order to find the best haplotype $\tilde{h}$ it is sufficient to enumerate all the s-cliques and consider each pattern π^i of a pattern table in standard form.

For each s-clique and pattern, we can find the local best haplotype $\bar{h}^{K,i}$ by setting:

$$\bar{h}^{K,i}_p = \begin{cases} 0 & \text{if } \pi^i_p = 0 \\ 1 & \text{if } \pi^i_p = 1 \\ 0 & \text{if } \pi^i_p = - \text{ and } \alpha(K,p) \geq \beta(K,p) \\ 1 & \text{if } \pi^i_p = - \text{ and } \alpha(K,p) < \beta(K,p) \end{cases}$$

Keeping track of the haplotype that maximizes (3.10), we obtain at the end $\tilde{h}$.

Notice that the introduction of the notion of s-cliques allows to decompose the pricing problem into smaller problems. Although this leads to a reduction on the total number of solutions to be investigated, the number of patterns to be enumerated is exponential.

Finally, to find the integer solution, a standard branch-and-bound approach is used, where at each node the branching decision forces the presence or absence of a certain haplotype in the solution forcing the value assumed by a chosen variable. We obtain the optimal value for the linear relaxation by recalling both the column and cut generation.

Results of this state-of-the-art algorithm have been provided on instances with up to 50 genotypes and number of SNPs varying from 20 to almost 200.

3.4 Polynomial-size integer formulations

Exponential-size models can be of difficult practical use, as the large number of variables and constraints affects the use of memory and computational time, forcing to use approaches as columns and rows generation. Efforts have thus been made in order to find formulations that are valid with only a polynomial number of variables and constraints.

3.4.1 A formulation with duplicate haplotypes

One of the first polynomial-size formulations can be found in [12], where variables explicitly represent the haplotypes involved in the solution. Consider the following binary variables:

- $y_{2i-1,p}$, $y_{2i,p}$ for $i \in I = \{1, \ldots, m\}, p \in P$ that represent each site of the two haplotypes resolving genotype g^i,
- $d_{i,j}$ for $i, j \in \{1, \ldots, 2\,m\}$ equal to one if haplotypes indexed by i and j are equal,
- x_i for $i \in \{1, \ldots, 2\,m\}$ equal to one if the haplotype indexed with i is used to resolve at least a genotype and it is not equal to any other haplotype indexed with $j < i$. In practice, these variables are used to count the distinct haplotypes involved in the solution.

Note that this approach, compared with all the other formulations proposed, uses a slightly different notation, as homozygous sites are identified with 0s and 2s, while heterozygous sites with 1s, so that we can use the usual operation of addition to say that two haplotypes h^1, h^2 resolve g if for every site p we have $h^1_p + h^2_p = g_p$. The

PolyIP formulation is then given by:

$$
\begin{aligned}
(PolyIP)\ \min\ & \sum_{i=1}^{2m} x_i && && (3.11)\\
\text{s.t.}\ & y_{2i-1,p} + y_{2i,p} = g_p^i && \forall\, i \in I, \forall\, p \in P && (3.12)\\
& d_{i,j} \geq y_{i,p} - y_{j,p} && \forall\, 1 \leq i < j \leq 2m, \forall\, p \in P && (3.13)\\
& d_{i,j} \geq y_{j,p} - y_{i,p} && \forall\, 1 \leq i < j \leq 2m, \forall\, p \in P && (3.14)\\
& x_i \geq 2 - i + \sum_{j=1}^{i-1} d_{j,i} && \forall\, i \in \{1, \dots, 2m\} && (3.15)\\
& x_i, d_{i,j}, y_{i,p} \in \{0,1\} && \forall\, i,j \in \{1, \dots, 2m\}, \forall p \in P && (3.16)
\end{aligned}
$$

where constraints (3.12) make sure each genotype is properly resolved, constraints (3.13) and (3.14) ensure that variable $d_{i,j}$ is set to 1 as soon as the haplotypes i and j are different and constraints (3.14) fix x_i to 1 if the associated haplotype is different from all the previous ones.

Unfortunately, this polynomia-sizel formulation has a poor linear relaxation, that means that the gap between the optimal solution and the solution of the linear relaxation is usually large. To overcome this issue, in [13] the authors propose a perturbation of the objective function and additional valid cuts. Anyway, also these improvements do not make this approach competitive when compared with the formulation in [25] (for small instances), so that another approach, called *HybridIP*, that exploits ideas from both this polynomial formulation and RTIP, is proposed [13]. Results show that this last approach has a better performance in terms of computational times, in fact both models have been tested on instances with up to 50 genotypes and 10 or 30 SNPs, while the HybridIP formulation was also able to solve instances with 30 genotypes and up to 100 SNPs. Anyway, we do not provide further details on this approach based on a exponential-size model, since it has been outperformed by [33].

3.4.2 A formulation with distinct haplotypes

Another polynomial formulation has been presented in [7]. In this case, we define an upper bound UB on the number of needed haplotypes, that can be either simply twice the number of genotypes or a better upper bound obtained with a heuristic. Thus, we define the set $I_{UB} = \{1, \dots, UB\}$ and the following binary variables:

- $x_i, i \in I_{UB}$ that takes value 1 if the corresponding haplotype is chosen in the solution,
- $y_{i,j}^k, i, j \in I_{UB}, k \in I$ that is equal to 1 if haplotypes i and j are used to resolve genotype g^k,
- $z_{i,p}, i \in I_{UB}, p \in P$ that records the values of each site of the haplotypes.

The formulation obtained is the following:

$$\min \sum_{i \in I_{UB}} x_i \tag{3.17}$$

$$\text{s.t.} \sum_{\{i,j\}} y_{i,j}^k \geq 1 \qquad \forall\, k \in I \tag{3.18}$$

$$\sum_{j \neq i} y_{i,j}^k \leq x_i \qquad \forall\, k \in I, \forall\, i \in I_{UB} \tag{3.19}$$

$$z_{i,p} + \sum_{j \neq i} y_{i,j}^k \leq x_i \qquad \forall\, k \in I, \forall\, i \in I_{UB}, \forall\, p \in P : g_p^i = 0 \tag{3.20}$$

$$z_{i,p} \geq \sum_{j \neq i} y_{i,j}^k \qquad \forall\, k \in I, \forall\, i \in I_{UB}, \forall\, p \in P : g_p^i = 1 \tag{3.21}$$

$$z_{i,p} + z_{j,p} \geq y_{i,j}^k \qquad \forall\, k \in I, \forall\, i,j \in I_{UB}, i \neq j, \forall\, p \in P : g_p^i = 2 \tag{3.22}$$

$$z_{i,p} + z_{j,p} \leq x_i + x_j - y_{i,j}^k \qquad \forall\, k \in I, \forall\, i,j \in I_{UB}, i \neq j, \forall\, p \in P : g_p^i = 2 \tag{3.23}$$

$$x_i, y_{i,j}^k, z_{i,p} \in \{0,1\} \qquad \forall\, k \in I, \forall\, i,j \in I_{UB}, \forall p \in P \tag{3.24}$$

where constraints (3.18) ensure that each genotype is resolved at least by a pair of haplotypes, constraints (3.19) activate the x variables and constraints (3.20) - (3.23) establish the relation between variables x, y and z according to the value of g_p^i.

Both the number of variables (UB x variables, $UB(UB-1)/2$ y variables and nUB z variables) and the number of constraints (m for (3.18), $O(mUB)$ for (3.19) - (3.23)) are polynomial in the input size.

Starting from this formulation it is possible to build a procedure that solves the problem only considering variables y and z in the following way: starting from a good heuristic solution, with objective value $z^{\star}_{HEUR}$, consider $UB = z^{\star}_{HEUR} - 1$ and $x_i = 1$ for $i \in \{1, \ldots, UB\}$. Then, the model (3.17)-(3.24) can be re-written in such a way that the objective function consists of maximizing the number of resolved genotypes among the m considered:

$$\max \sum_{k \in I} \sum_{\{i,j\}} y_{i,j}^k$$

and the constraints (3.18) are re-written as

$$\sum_{\{i,j\}} y_{i,j}^k \leq 1 \qquad \forall\, k \in I$$

If the maximum number of resolved genotypes found solving this model is less than m, we stop the algorithm and consider the preceeding solution as the optimal one, otherwise we decrease again the value of UB, as illustrated in Algorithm 3.8. Further improvements to this model include the introduction of additional constraints that dominate the previous ones and then can substitute current constraints in order to improve the model.

The results obtained from instances with 50 genotypes and 10 or 30 SNPs show how this formulation has better performance than the PolyIP formulation.

Algorithm 3.8 [7]

```
 1: procedure SOLVE
 2:     use a heuristic to get an initial solution and let $UB = z^{\star}_{HEUR} - 1$,
 3:     define the maximization problem and let $z^{OPT} = z^{\star}_{HEUR}$,
 4:     solve the max problem and let $p^{\star}_{UB}$ be the optimal integer solution,
 5:     if $p^{\star}_{UB} < m$ then
 6:         $z^{OPT}$ is the optimal solution. STOP.
 7:     else
 8:         let $z^{OPT} = p^{\star}_{UB}$, $UB = UB - 1$ and go back to step 4,
 9:     end if
10: end procedure
```

3.4.3 A class representative model

An alternative polynomial-size formulation is the one presented in [14]. This formulation attempts to break the simmetries in the problem. Specifically, we can note how, in the formulations proposed so far, we end up with equivalent solutions by swapping the indices of variables associated to two distinct haplotypes. To overcome this issue, the formulation proposed in [14] is based on class representatives: given G the set of genotypes, each subset $S_i \subseteq G$ of genotypes is indexed according to the first genotype g^i, in a predefined order, belonging to it.

In this way, we identify a solution to the HIPP problem with a set of haplotypes and, for each of them, a subset of genotypes resolved by it that can be written in terms of class representatives. We will say that a haplotype of the solution induces a subset of genotypes if it is used to resolve every genotype belonging to the subset. As each genotype is resolved by two haplotypes, it can happen that the same genotype g^i is the first element of two subsets, so that both would be indexed in the same way. To avoid this, a dummy genotype $g^{i'}$ is created, that is a copy of the real one, and it is used to identify the second subset. Thus, subsets S_i are defined, with index i varying in the set $\bar{K} = K \cup K'$, where $K = \{1, 2, \dots, m\}$ and $K' = \{1', 2', \dots, m'\}$. An ordering such that $1 < 1' < 2 < 2' < \dots < m < m'$ is also considered.

The variables x_i, $i \in \bar{K}$ used to describe the formulation record how many haplotypes (at most $2\,m$) are used in the solution, or equivalently how many subsets are induced. In particular, we define binary variables $y^k_{i,j}$, with $i, j \in \bar{K}$ and $k \in K$, that describe how genotype g^k is resolved, that is variable $y^k_{i,k}$ takes value 1 if genotype g^k belongs to subsets S_i and S_j, thus meaning that g^k is resolved by the haplotypes associated to S_i and S_j. If a haplotype belonging to the solution induces a subset S_i, the corresponding binary variables x_i is set to take value 1. Finally, further binary variables $z_{i,p}$, $i \in \bar{K}$, $p \in P$ record the value of the p-th SNP in haplotype i.

Note that a preprocessing step is easily available to reduce the number of variables. In fact, as we have $y^k_{i,j} = 1$ if and only if $y^k_{j,i} = 1$, we can define the y variables only if $i < j \leq k$ or if $i = k$ and $j = k'$. Moreover, variables $y^k_{i,k'}$ with $i < k$ are not needed, as it would be in contradiction with the assumption that the dummy genotypes are used only if needed: if $y^k_{i,k'} = 1$, it means that genotype k belongs to subsets whose first genotype is i and k', and as in a solution each genotype belongs exactly to two subsets, it means that we are not considering any subset whose first genotype is k. Following similar considerations, the following sets of variables do not need to be defined:

$$\begin{aligned}
R_1 &= \{y^k_{i,j} : k \in K, i,j \in \bar{K}, j < i < k\},\\
R_2 &= \{y^k_{i,k'} : k \in K, i \in \bar{K}, i \leq (k-1)'\},\\
R_3 &= \{y^k_{i,i'} : k \in K, i \in \bar{K}, 2 \leq i \leq k-1\}.
\end{aligned}$$

Taking into account compatibility issues, moreover, further sets of variables are not needed:

$$\begin{aligned}
R_4 &= \{y^k_{i,j} : k \in K, i,j \in \bar{K}, p \in P, g^k_p = 2, g^i_p = g^j_p \neq 2\},\\
R_5 &= \{y^k_{i,j} : k \in K, i,j \in \bar{K}, p \in P, g^k_p + g^i_p = 1 \text{ or } g^k_p + g^j_p = 1\}.
\end{aligned}$$

Thus, defining the set $\hat{\mathcal{Y}} = \{y^k_{i,j} : k \in K, i,j \in \bar{K}\} \backslash (R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5)$, the following polynomial integer programming (PIP) formulation for the HIPP problem is given:

$$\begin{aligned}
\text{(PIP)}\quad \min\quad & \sum_{i \in \bar{K}} x_i && && (3.25)\\
\text{s.t.}\quad & x_{i'} \leq x_i && \forall\, i \in K && (3.26)\\
& \sum_{i,j:\, y^k_{i,j} \in \hat{\mathcal{Y}}} y^k_{i,j} \geq 1 && \forall\, k \in K && (3.27)\\
& \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j \geq i} y^k_{i,j} + \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j < i} y^k_{j,i} \leq x_i && \forall\, k \in K, i \in \bar{K} : y^k_{i,j} \in \hat{\mathcal{Y}} && (3.28)\\
& y_{k,k'} \leq x_{k'} && \forall\, k \in K && (3.29)\\
& z_{k,p} + z_{k',p} = 1 && \forall\, k \in K, \forall\, p \in P : g^k_p = 2 && (3.30)\\
& z_{i,p} \leq 1 - \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j \geq i} y^k_{i,j} - \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j < i} y^k_{j,i} && \forall\, k \in K, p \in P, i \in \bar{K} : y^k_{i,j} \in \bar{\mathcal{Y}}, && \\
& && g^k_p = 0, g^i_p = 2 && (3.31)\\
& z_{i,p} \geq \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j \geq i} y^k_{i,j} + \sum_{j:\, y^k_{i,j} \in \hat{\mathcal{Y}}, j < i} y^k_{j,i} && \forall\, k \in K, p \in P, i \in \bar{K} : y^k_{i,j} \in \bar{\mathcal{Y}}, && \\
& && g^k_p = 1, g^i_p = 2 && (3.32)\\
& z_{i,p} \geq y^k_{i,j} && \forall\, k \in K, p \in P, \forall\, i,j : y^k_{i,j} \in \hat{\mathcal{Y}}, && \\
& && g^k_p = 2, g^i_p = 2, g^j_p = 0 && (3.33)\\
& z_{j,p} \geq y^k_{i,j} && \forall\, k \in K, p \in P, \forall\, i,j : y^k_{i,j} \in \hat{\mathcal{Y}}, && \\
& && g^k_p = 2, g^i_p = 0, g^j_p = 2 && (3.34)\\
& z_{i,p} \leq 1 - y^k_{i,j} && \forall\, k \in K, p \in P, \forall\, i,j : y^k_{i,j} \in \hat{\mathcal{Y}}, && \\
& && g^k_p = 2, g^i_p = 2, g^j_p = 1 && (3.35)\\
& z_{j,p} \leq 1 - y^k_{i,j} && \forall\, k \in K, p \in P, \forall\, i,j : y^k_{i,j} \in \hat{\mathcal{Y}}, &&
\end{aligned}$$

$$g_p^k = 2, g_p^i = 1, g_p^j = 2 \quad (3.36)$$

$$z_{i,p} + z_{j,p} \geq y_{i,j}^k \qquad \forall\, k \in K, p \in P, \forall\, i,j : y_{i,j}^k \in \hat{\mathcal{Y}},$$
$$g_p^k = 2, g_p^i = 2, g_p^j = 2 \quad (3.37)$$

$$z_{i,p} + z_{j,p} \leq 2 - y_{i,j}^k \qquad \forall\, k \in K, p \in P, \forall\, i,j : y_{i,j}^k \in \hat{\mathcal{Y}},$$
$$g_p^k = 2, g_p^i = 2, g_p^j = 2 \quad (3.38)$$

$$x_i, y_{i,j}^k, z_{i,p} \in \{0,1\} \qquad \forall\, k \in K, \forall\, i,j \in \bar{K}, \forall\, p \in P \quad (3.39)$$

where constraints (3.26) and (3.29) are needed to ensure that the dummy genotypes are used only when needed, constraints (3.27) ensure that each genotype belongs to two subsets, constraints (3.28) set the value of x_i, constraints (3.30) represent the condition that each heterozygous site must be correctly explained and constraints (3.31)-(3.38) establish the relations between z and y variables.

Slight improvements to this model can be achieved adding strenghtening inequalities. Some of them are in polynomial number, so that they are added all together to the original formulation, while other sets of cuts can be generated by means of a separation oracle.

This formulation proves to outperform all previous models presented in [25], [13] and [7] on the same instances used by Brown and Harrower [13]. In particular, this model can be applied to larger real biological genotype data sets and it is polynomial, so that it can be solved with standard solvers.

3.5 A Heuristic approach

When the instances have a large size and all exact methods fail in finding a solution within a reasonable time limit, a viable way to find a reasonable, although not necessary optimal, solution is to use a heuristic approach.

Among the heuristic approaches to HIPP, we cite [28, 31, 46]. In particular, the heuristic approach presented in [46] under the name of *CollHaps algorithm* proved to give good heuristic solutions, most times coincident to the optimal ones, or in the other cases close to them. These results are achieved on larger instances (up to 103 SNPs and 1000 genotypes) than the ones solved by exact algorithms in the literature so far. Given a set of m genotypes $g^1, \ldots, g^m$ with n SNPs, let V be the total number of heterozygous SNPs involved and x_v, $v \in \{1, \ldots, V\}$ a variable associated to each heterozygous SNP. Let μ be the bijective map that, given a pair (i,p) such that $g_p^i = 2$, returns the corresponding variable index: $v = \mu(i,p)$. Now, we consider two *symbolic haplotypes* for each genotype g^i, defined in the following way:

$$h_p^{2i-1} = \begin{cases} 0 & \text{if } g_p^i = 0 \\ 1 & \text{if } g_p^i = 1 \\ x_{\mu(i,p)} & \text{if } g_p^i = 2 \end{cases}$$

$$h_p^{2i} = \begin{cases} 0 & \text{if } g_p^i = 0 \\ 1 & \text{if } g_p^i = 1 \\ \bar{x}_{\mu(i,p)} & \text{if } g_p^i = 2 \end{cases}$$

where $\bar{x}_{\mu(i,p)} = 1 - x_{\mu(i,p)}$. These haplotypes are used as rows of a $2m \times n$ matrix, called *symbolic haplotype matrix.* According to the parsimony criterion, finding the best set of haplotypes resolving all the genotypes can be seen as finding a variable assignment so that the resulting number of distinct rows in the symbolic haplotype matrix is minimum.

Definition 3.5. *A* collapse rule *is an assignment $\vartheta(V)$ of variables value that leads two symbolic haplotypes to be identical.*

Note that, given two haplotypes h^1 and h^2, we can apply a collapse rule only if for every site p one of the following alternatives holds: $h^1_p = h^2_p$, or only one between h^1_p and h^2_p is a variable, or both are variables but not complementary. In particular, the collapse rule ϑ associated to two symbolic haplotypes h^1 and h^2 can be described in the following way:

- if $h^1_p = x_v$ (resp. h^2_p) and $h^2_p = c$, $c \in \{0, 1\}$ (resp. h^1_p), then $\vartheta(x_v) = c$,
- if $h^1_p = \bar{x}_v$ (resp. h^2_p) and $h^2_p = c$, $c \in \{0, 1\}$ (resp. h^1_p), then $\vartheta(x_v) = 1 - c$,
- if $h^1_p = x_v$ and $h^2_p = x_u$ or $h^1_p = \bar{x}_v$ and $h^2_p = \bar{x}_u$, then $\vartheta(x_u) = x_v$,
- if $h^1_p = x_v$ and $h^2_p = \bar{x}_u$ or $h^1_p = \bar{x}_v$ and $h^2_p = x_u$, then $\vartheta(x_u) = \bar{x}_v$,
- ϑ is the identity on all other cases.

As the same variable will have, in general, several occurrencies in the symbolic haplotype matrix, we need to propagate the rule to all the matrix rows.

Definition 3.6. *A* collapse step *is the propagation of a particular collapse rule to all the rows of the symbolic haplotype matrix.*

In the following example we show how a collapse rule and collapse step work.

Example 3.3. *Consider the following two symbolic haplotypes that represent two rows of a symbolic matrix.*

$$\begin{array}{ccccc} 1 & x_1 & 0 & 0 & x_2 \\ \bar{x}_3 & x_4 & 0 & 0 & 1 \end{array}$$

The collapse rule applied to these two haplotypes consists in assigning the values $x_1 = x_4$, $x_2 = 1$, $x_3 = 0$. Note that the associated collapse step consists in assigning the same values to variables x_1, x_2 and x_3 whenever they appear in the remaining rows of the symbolic matrix.

The CollHaps algorithm tries to produce a feasible assignment of every variable x_v, $v \in V$ iterating the collapse step, thus choosing two different symbolic haplotypes at a time to be made identical, as many times as possible, so that we reduce the number of distinct rows of the symbolic matrix as much as possible.

Note that at each performed collapse step the number of distinct rows decreases by at least one. When no more collapse steps are possible, there can be still some non-assigned entries of the matrix, at this point any assignment to them will not decrease the number of distinct rows anymore. In [46], it is shown that an appropriate sequence of collapse steps can reach the optimal solution of the HIPP problem, but there is no strategy to find such a sequence of steps. Thus, a heuristic approach is needed to produce a sequence of collapse steps, based on a *randomized quasi-greedy routine*: for each pair (i, j) of symbolic haplotypes we define a distance value $d_{(i,j)}$ computed as the number of variables to be set to constant values by the application of the collapse rule to that pair, then probabilities are assigned so that a pair with distance value d has twice the probability of being chosen than a pair with distance value $d + 1$. Pair of haplotypes are then randomly chosen according to these probabilities.

Finally, several runs of the complete algorithm are performed and the best solution so far is stored. Computational experiments on real data and software-generated instances proved that this approach achieves good performance in terms of quality of the solution and computational times.

CHAPTER 4

New formulations for the Haplotype Inference by Pure Parsimony problem

In this chapter we briefly recall the model described in [14] and, after some considerations, we give a slightly improved formulation. Then, we present a new polynomial formulation that involves only two-index variables and we provide a Dantzig-Wolfe decomposition of it, resulting in a formulation with an exponential number of variables. This latter formulation is considered in the remainder of the thesis, in which its structure is investigated to provide a competitive solution approach for the HIPP problem.

4.1 Improvements to the class representatives model

We recall that (PIP), the model described in Section 3.4.3, allows to find a solution to the HIPP problem that associates to each used haplotype a subset of genotypes resolved by it. These subsets are indexed with the identifier of the first genotype contained in them, according to a predefined order. Moreover, for each feasible solution they satisfy the following properties:

- each subset of genotypes shares at least one compatible haplotype,
- each genotype belongs exactly to two of these subsets,
- every pair of subsets intersects in at most one genotype.

This formulation is intrinsically symmetry-breaking. Moreover, an important step to increase the performance of this model consists in restricting the set of indices for which a variable is actually needed, in particular the reduction of the number of y variables to be defined.

We want to identify the same subset $\hat{Y}$ of variables $y_{i,j}^{k}$, that actually need to be defined, in a different way with respect to [14], always taking into account symmetries and compatibility issues. We propose the approach described by Algorithm 4.9

Algorithm 4.9

```
1: initialize $\hat{Y} = \varnothing$
2: for $k \in K$ do
3:     insert $y^k_{k,k'}$ into $\hat{Y}$,
4:     for $i \in \bar{K}$ such that $i < k$ do
5:         for $j \in \bar{K}$ such that $j \neq i'$ and $j <= k$ do
6:             if $g^k$ is compatible with $g^i$ and $g^j$ then
7:                 insert $y^k_{i,k}$ into $\hat{Y}$,
8:             end if
9:         end for
10:    end for
11: end for
```

The resulting modified formulation PIP' we present is the following:

$$\text{(PIP')} \quad \min \sum_{i \in K \cup K'} x_i \tag{4.1}$$

$$\text{s.t.} \quad x_{i'} \leq x_i \qquad \forall\, i \in K \tag{4.2}$$

$$\sum_{i,j:\, y^k_{i,j} \in \hat{Y}} y^k_{i,j} \geq 1 \qquad \forall\, k \in K \tag{4.3}$$

$$\sum_{j:\, y^k_{i,j} \in \hat{Y}} y^k_{i,j} + \sum_{j:\, y^k_{j,i} \in \hat{Y}} y^k_{j,i} \leq x_i \qquad \forall\, k \in K, i \in K \cup K' \tag{4.4}$$

$$y^k_{k,k'} \leq x_{k'} \qquad \forall\, k \in K \tag{4.5}$$

$$z_{i,p} \leq 1 - \sum_{j:\, y^k_{i,j} \in \hat{Y}} y^k_{i,j} - \sum_{j:\, y^k_{j,i} \in \hat{Y}} y^k_{j,i} \qquad \forall\, p \in P, \forall\, k \in K, i \in K \cup K', \quad \text{s.t. } g^k_p = 0, g^i_p \neq 1 \tag{4.6}$$

$$z_{i,p} \geq \sum_{j:\, y^k_{i,j} \in \hat{Y}} y^k_{i,j} + \sum_{j:\, y^k_{j,i} \in \hat{Y}} y^k_{j,i} \qquad \forall\, p \in P, \forall\, k \in K, i \in K \cup K', \quad \text{s.t. } g^k_p = 1, g^i_p \neq 0 \tag{4.7}$$

$$z_{i,p} \geq y^k_{i,j} \qquad \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K' \quad \text{s.t. } y^k_{i,j} \in \hat{Y}, \; g^k_p = 2, g^i_p \neq 0, g^j_p = 0 \tag{4.8}$$

$$z_{j,p} \geq y^k_{i,j} \qquad \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K' \quad \text{s.t. } y^k_{i,j} \in \hat{Y}, \; g^k_p = 2, g^i_p \neq 0, g^j_p = 0 \tag{4.9}$$

$$z_{i,p} \leq 1 - y^k_{i,j} \qquad \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K' \quad \text{s.t. } y^k_{i,j} \in \hat{Y}, \; g^k_p = 2, g^i_p \neq 1, g^j_p = 1 \tag{4.10}$$

$$z_{j,p} \leq 1 - y^k_{i,j} \qquad \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K'$$

$$\begin{aligned}
& && \text{s.t. } y^k_{i,j} \in \hat{Y}, \\
& && g^k_p = 2, g^i_p = 1, g^j_p \neq 1 && (4.11) \\
& z_{i,p} + z_{j,p} \geq y^k_{i,j} && \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K' \\
& && \text{s.t. } y^k_{i,j} \in \hat{Y}, \\
& && g^k_p = 2, g^i_p = 2, g^j_p = 2 && (4.12) \\
& z_{i,p} + z_{j,p} \leq 2 - y^k_{i,j} && \forall\, p \in P, \forall\, k \in K, i, j \in K \cup K' \\
& && \text{s.t. } y^k_{i,j} \in \hat{Y}, \\
& && g^k_p = 2, g^i_p = 2, g^j_p = 2 && (4.13) \\
& x_i, y^k_{i,j}, z_{i,p} \in \{0, 1\} && && (4.14)
\end{aligned}$$

Constraints (4.3) are needed to ensure that each genotype is resolved, that is, it belongs at least to two different subsets. To link the values of variables x_i and $y^k_{i,j}$ we use constraints (4.4), that record whenever the haplotype induced by S_i is used. Constraints (4.5) force $x_{k'}$ to be equal to 1 if the dummy genotype $g^{k'}$ is needed and used. Constraints (4.6)-(4.13) guarantee that the values of the haplotypes that induce the subsets of genotypes are consistent. Namely, that they are compatible with the genotypes of the subsets and they resolve the genotypes for which variables $y^k_{i,j}$ are set to 1.

Observation 4.1. *It is not possible to use the dummy genotype k' without having $y^k_{k,k'} = 1$.*

Proof. Suppose that $y^k_{i,j} = 0$ and that there exist genotypes g^h, g^t compatible with g^k but not compatible between themselves, such that they both are resolved using one haplotype induced by a subset whose least index is k. In this case, we would need the dummy index k'. But, as genotype g^k belongs to only two subsets, those would be S_k and $S_{k'}$, meaning that $y^k_{k,k'}$ must be equal to 1. □

Observation 4.2. *The combinations obtained from constraints (4.6)-(4.13) are all and the only necessary ones.*

Proof. Consider Table 4.1. The first three columns show all the possible combinations of values for g^i_p, g^j_p and g^k_p, while the last column states if the combination is not allowed for compatibility reasons (-) or in which constraints we can find that combination. □

Note that the combinations highlighted in red in Table 4.1 where not considered in model (PIP). As a consequence, the explicit description of haplotypes did not agree with the structure of the genotypes resolved by them.

Observation 4.3. *The following constraints appearing in model (PIP) are redundant:*

$$z_{k,p} + z_{k',p} = 1 \quad \forall\, k \in K, \forall\, p \in P :\ g^k_p = 2$$

Proof. Look at constraints (4.12) and (4.13). If $y^k_{i,j} = 1$ these constraints become exactly equal to (3.31), otherwise what happens to $z_{k'}$ is irrelevant. □

For this reason we omitted them in the new model, (PIP').

g_p^i	g_p^j	g_p^k	constr.
0	0	0	(4.6)
0	0	1	-
0	1	0	-
1	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-
1	1	1	(4.7)
0	0	2	-
1	0	2	(4.8)-(4.11)
0	1	2	(4.9)-(4.10)
1	1	2	-
0	2	0	(4.6)
1	2	0	-
0	2	1	-
1	2	1	(4.7)
2	0	0	(4.6)
2	1	0	-
2	0	1	-
2	1	1	(4.7)
0	2	2	(4.9)
1	2	2	(4.11)
2	0	2	(4.8)
2	1	2	(4.10)
2	2	0	(4.6)
2	2	1	(4.7)
2	2	2	(4.12)-(4.13)

Table 4.1: Combinations of SNPs' values.

4.2 A new intuitive polynomial two-index formulation

In this section we present another possible formulation for the HIPP problem that involves only two-index variables and turns out to be polynomial with respect to both the number of constraints and the number of variables [19].

We want to build a solution that records the structure of the haplotypes involved as well as which haplotypes are in fact used and which genotypes are resolved by them. First of all, we note that if a genotype has no heterozygous sites, than it must be resolved by two identical copies of the same haplotype, that is in fact equal to the genotype. Hence, it is convenient to leave out those genotypes from the model's constraints, but we need to consider the associated haplotypes as they can be used to resolve also other genotypes. Recall that we denote with *fixed genotypes* the ones with no heterozygous sites, and with *fixed haplotypes* the haplotypes associated to them.

Suppose our set G contains some fixed genotypes, we can assume that the non-fixed ones are contained in the subset $K \subseteq G$ and indexed from 1 to $|K|$, while the remaining $m - |K|$ at the end of the set are fixed and resolved by $m - |K|$ associated haplotypes. The remaining genotypes require at most $2|K|$ distinct haplotypes to be resolved. We define the following binary variables:

- x_i, $i \in \{1, \dots, 2|K|\}$ take value 1 if the i-th non-fixed haplotype is used to resolve at least one non-fixed genotype, 0 otherwise,
- y_i^k, $i \in \{1, \dots, m + |K|\}, k \in \{1, \dots, |K|\}$ take value 1 if the i-th haplotype is used to resolve genotype k,
- $z_{i,p}$, $i \in \{1, \dots, 2|K|\}, p \in P$ is the value taken by non-fixed haplotype i in position p,

that allow the HIPP problem to be described by the following model (QP), which presents a set of quadratic constraints:

$$(QP)\ \min \sum_{i=1}^{2|K|} x_i \qquad + (m - |K|) \tag{4.15}$$

$$s.t. \quad \sum_{i=1}^{m+|K|} y_i^k = 2 \qquad \forall\, k \in \{1, \dots, |K|\} \tag{4.16}$$

$$\sum_{i=1}^{2|K|} y_i^k z_{ip} + \sum_{i=2|K|+1}^{m+|K|} y_i^k g_p^i = 1 \quad \forall\, k \in \{1, \ldots, |K|\},\ p \in P :\ g_p^k = 2 \tag{4.17}$$

$$z_{ip} \geq y_i^k \qquad \forall\, i \in \{1, \ldots, 2|K|\},\ k \in \{1, \ldots, |K|\},\ p \in P :\ g_p^k = 1 \tag{4.18}$$

$$z_{ip} \leq 1 - y_i^k \qquad \forall\, i \in \{1, \ldots, 2|K|\},\ k \in \{1, \ldots, |K|\},\ p \in P :\ g_p^k = 0 \tag{4.19}$$

$$y_i^k \leq x_i \qquad \forall\, i \in \{1, \ldots, 2|K|\},\ k \in \{1, \ldots, |K|\} \tag{4.20}$$

$$\sum_{k=1}^{m+|K|} y_i^k \geq x_i \qquad \forall i \in \{1, \ldots, m + |K|\} \tag{4.21}$$

$$z_{ip} \in \{0, 1\} \qquad \forall\, i \in \{1, \ldots, 2|K|\},\ p \in P \tag{4.22}$$

$$y_i^k \in \{0, 1\} \qquad \forall\, i \in \{1, \ldots, m + |K|\},\ k \in \{1, \ldots, |K|\} \tag{4.23}$$

$$x_i \in \{0, 1\} \qquad \forall\, i \in \{1, \ldots, 2|K|\} \tag{4.24}$$

The objective function minimizes the number of non-fixed haplotypes needed to resolve all the non-fixed genotypes plus a constant, that represents the number of fixed haplotypes that are forced into the solution. Constraints (4.16) ensure that exactly two haplotypes are used to resolve any non-fixed genotype and by constraints (4.17) only one among these two haplotypes takes value 1 at each heterozygous site of genotype k. Constraints (4.18) and (4.19) fix the values of the p-th position of a haplotype if it is used to resolve a genotype with a homozygous value in position p. Finally, constraints (4.20) force variable x_i to be equal to 1 if at least one genotype is resolved by the i-th haplotype. Constraints (4.21) imply that, as soon as no genotype is resolved by the i-th haplotype variable, x_i is set to 0. Note that due to the structure of the objective function these constraints are redundant for the formulation and can be omitted.

As we already pointed out, constraints (4.17) are quadratic, so that in order to solve this model using the procedures described in Chapter 2 for integer linear programming, it would be necessary to linearize those constraints, introducing $O(|K|^2 n)$ new variables w_{ip}^k for all $i \in \{1, \ldots, 2|K|\}, k \in \{1, \ldots, |K|\}, p \in P$ such that $g_p^k = 2$. Thus, the original quadratic constraints (4.17) should be substituted by the following ones:

$$\sum_{i=1}^{2|K|} w_{ip}^k + \sum_{i=2|K|+1}^{m+|K|} y_i^k g_p^i = 1 \qquad \forall k \in \{1, \ldots, |K|\}, p \in P : g_p^k = 2 \tag{4.25}$$

$$w_{ip}^k \leq y_i^k \qquad \forall i \in \{1, \ldots, 2|K|\}, k \in \{1, \ldots, |K|\}, p \in P : g_p^k = 2 \tag{4.26}$$

$$w_{ip}^k \leq z_{ip} \qquad \forall i \in \{1, \ldots, 2|K|\}, k \in \{1, \ldots, |K|\}, p \in P : g_p^k = 2 \tag{4.27}$$

$$w_{ip}^k \geq y_i^k + z_{ip} - 1 \qquad \forall i \in \{1, \ldots, 2|K|\}, k \in \{1, \ldots, |K|\}, p \in P : g_p^k = 2 \tag{4.28}$$

From now on we refer to (QPL) as the linearized formulation.

The introduction of the linearizing constraints brings back to a three index formulation, with increased number of variables and constraints, and presents a great number of symmetries. Another possible approach could then be investigated, starting from a reformulation of this model as it is shown in the following section.

4.3 A new exponential-size formulation

We apply a Dantzig-Wolfe decomposition to the formulation (QPL). Notice that, if we consider one haplotype h^i at a time, the (QPL) formulation presents a block structure, where constraints (4.16) and (4.25) are the linking constraints. Moreover, we note that all blocks are identical, thus we expect that this reformulation will break the noticed symmetries. Let s be the total number of heterozygous sites contained in the genotypes, so that for every $i \in \{1, \ldots, 2|K|\}$ we need to define s variables for the linearization of constraints.

We define the set of feasible solutions within a single block as follows, where we collect information coming from contraints (4.18)-(4.21) and (4.26)-(4.28):

$$\begin{aligned} X_i = \{(z_i, y_i, x_i, w_i) \in \{0,1\}^{n+m+1+s} | \; & z_{ip} \geq y_i^k \; \forall \; k, p \text{ if } g_p^k = 1, \\ & z_{ip} \leq 1 - y_i^k \; \forall \; k, p \text{ if } g_p^k = 0, \\ & w_{ip}^k \leq y_i^k \; \forall \; k, p \text{ if } g_p^k = 2, \\ & w_{ip}^k \leq z_{ip} \; \forall \; k, p \text{ if } g_p^k = 2, \\ & w_{ip}^k \geq y_{ip}^k + z_{ip} - 1 \; \forall \; k, p \text{ if } g_p^k = 2, \\ & y_i^k \leq x_i \; \forall k, \\ & \sum_{k=1}^{|K|} y_i^k \geq x_i \} \end{aligned}$$

As for each haplotype i this set is described by the same integer points, we refer to it simply as set X. In particular, we can give an interpretation of the points in X as the vectors z_i that identify the structure of a haplotype and the genotypes that are resolved using that haplotype, determined by those k such that $y_i^k = 1$. Then, if there is at least one genotype of this kind, the value of variable x_i is set to 1. Note that this holds also for fixed haplotypes, as in this case variable x_i does not appear in the objective function. Finally, the values of variables $w_{i,p}^k$ are set according to the values of the other variables.

It is well known [16] that we can obtain the *Dantzig-Wolfe relaxation* of the (QPL) model by solving the following linear program:

$$\min \sum_{i=1}^{2|K|} x_i + (m - |K|) \tag{4.29}$$

$$s.t. \sum_{i=1}^{m+|K|} y_i^k = 2 \qquad \forall \, k \in \{1, \ldots, |K|\} \tag{4.30}$$

$$\sum_{i=1}^{2|K|} w_{ip}^k + \sum_{i=2|K|+1}^{m+|K|} y_i^k g_p^i = 1 \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P : g_p^k = 2 \tag{4.31}$$

$$(z_i, y_i, x_i, w_i) \in \text{conv}(X) \qquad \forall\, i \in \{1,\ldots,2|K|\} \tag{4.32}$$

This model can be further reformulated exploiting the structure of the set conv(X). We consider the set Q of all vertices (z_q, y_q, x_q, w_q) of conv(X). Note that X is a 0/1 polytope, so every point in X is a vertex of conv(X). We can describe every point of conv(X) as a convex combination of points in Q, with appropriate weights λ_i^q:

$$\forall\,(z_i, y_i, x_i, w_i) \in conv(X),\ (z_i, y_i, x_i, w_i) = \sum_{q\in Q} \lambda_i^q (z_q, y_q, x_q, w_q), \quad \sum_{q\in Q} \lambda_i^q = 1. \tag{4.33}$$

In particular,

$$z_{ip} = \sum_{q\in Q} \lambda_i^q z_{qp} \tag{4.34}$$

$$y_i^k = \sum_{q\in Q} \lambda_i^q y_q^k \tag{4.35}$$

$$x_i = \sum_{q\in Q} \lambda_i^q x_q \tag{4.36}$$

$$w_{ip}^k = \sum_{q\in Q} \lambda_i^q w_{qp}^k \tag{4.37}$$

$$\tag{4.38}$$

We define

$$\lambda^q = x_q \sum_{i=1}^{m+|K|} \lambda_i^q \tag{4.39}$$

and we denote with G^q the subset of genotypes of G that are resolved using haplotype z_q, that is the set of genotypes g^k such that $y_q^k = 1$. We now present some lemmas that are needed to provide the formulation of Dantzig-Wolfe relaxation.

Lemma 4.1. *Let c_q be coefficients that take value 1 if the haplotype z_q associated to vertex $q \in Q$ is not fixed, 0 otherwise. By reformulating (4.15), we get the new objective function:*

$$\sum_{q\in Q} c_q \lambda^q + (m - |K|) \tag{4.40}$$

Proof. We define $c_i = 1$ for the non-fixed genotypes $i = 1, \ldots, 2|K|$ and $c_i = 0$ for $i = 2|K|+1, \ldots, m+|K|$. Thus it holds:

$$\sum_{i=1}^{2|K|} x_i = \sum_{i=1}^{m+|K|} c_i x_i \overset{(\star)}{=} \sum_{i=1}^{m+|K|} \sum_{q\in Q} \lambda_i^q x_q c_q = \sum_{q\in Q} \left[x_q \sum_{i=1}^{m+|K|} \lambda_i^q \right] c_q =$$

$$= \sum_{q\in Q} c_q \lambda^q$$

Note that ($\star$) is ensured by (4.36) and by the definitions of c_i and c_q, as if the genotype is fixed the associated haplotype is also fixed $\square$

Lemma 4.2. *By reformulating constraints (4.16) we get the new constraints:*

$$\sum_{q\in Q:\, g^k\in G^q} \lambda^q = 2 \qquad \forall\, k \in \{1,\ldots,|K|\} \tag{4.41}$$

Proof.

$$\begin{aligned}
2 &= \sum_{i=1}^{m+|K|} y_i^k = \sum_{i=1}^{m+|K|} \sum_{q\in Q} \lambda_i^q y_q^k \overset{(\star)}{=} \sum_{i=1}^{m+|K|} \sum_{q\in Q} x_q \lambda_i^q y_q^k \overset{(\diamond)}{=} \\
&= \sum_{i=1}^{m+|K|} \sum_{q\in Q:\, g^k\in G^q} x_q \lambda_i^q = \sum_{q\in Q:\, g^k\in G^q} \left[x_q \sum_{i=0}^{m+|K|} \lambda_i^q \right] = \\
&= \sum_{q\in Q:\, g^k\in G^q} \lambda^q
\end{aligned}$$

As soon as there exists an index k such that $y_q^k = 1$, then also $x_q = 1$ by constraints (4.20) and adding it does not change the value of the sum. Hence, ($\star$) holds. ($\diamond$) holds as we are just writing the same condition in a different way: if $y_q^k = 1$, then genotype g^k is resolved using the haplotype z_q and this is equivalent to say that $g^k \in G^q$. $\square$

Lemma 4.3. *By reformulating constraints (4.17) we get the new constraints:*

$$\sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_p^q=1}} \lambda^q = 1 \qquad \forall\, k \in \{1,\ldots,|K|\},\ \forall\, p\in P:\ g_p^k = 2 \tag{4.42}$$

Proof.

$$\begin{aligned}
1 &= \sum_{i=0}^{2|K|} w_{ip}^k + \sum_{i=2|K|+1}^{m+|K|} y_i^k g_p^i = \\
&= \sum_{i=0}^{2|K|} \sum_{q\in Q} \lambda_i^q w_{qp}^k + \sum_{i=2|K|+1}^{m+|K|} y_i^k z_{ip} \overset{(\star)}{=} \\
&= \sum_{i=0}^{2|K|} \sum_{q\in Q} x_q \lambda_i^q w_{qp}^k + \sum_{i=2|K|+1}^{m+|K|} x_{\bar q} \lambda_i^{\bar q} y_{\bar q}^k z_{\bar q p} \overset{(\diamond)}{=} \\
&= \sum_{i=0}^{2|K|} \sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_{qp}=1}} x_q \lambda_i^q + \sum_{i=2|K|+1}^{m+|K|} \sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_{qp}=1}} x_q \lambda_i^q = \\
&= \sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_p^q=1}} \left[x_q \sum_{i=1}^{2|K|} \lambda_i^q + x_q \sum_{i=2|K|1}^{m+|K|} \lambda_i^q \right] = \sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_{qp}=1}} \left[x_q \sum_{i=1}^{m+|K|} \lambda_i^q \right] = \\
&= \sum_{\substack{q\in Q:\, g^k\in G^q,\\ z_{qp}=1}} \lambda^q
\end{aligned}$$

Note that $y_q^k = 1$ implies that $w_{qp}^k = 1$ and $x_q = 1$ by constraints (4.20) and (4.26). Thus, ($\star$) holds. With respect to the second addend, as $z_{ip} = g_p^i \in \{0,1\}$, z_{ip} is not a convex combination of vertices of conv(X), thus there exists a unique $\bar{q}$ such that $\lambda_i^{\bar{q}} = 1$ for this particular i. Hence we can rewrite

$$x_{\bar{q}}\lambda_i^{\bar{q}}y_{\bar{q}}^k z_{\bar{q}p} = \sum_{q\in Q} x_q \lambda_i^q y_q^k z_{qp}$$

Moreover, ($\diamond$) holds as, for the first addend, we embed in the sum's conditions the meaning of $w_{qp}^k = 1$. □

Lemma 4.4. *From equation (4.33) we get the constraint*

$$\sum_{q\in Q} \lambda^q \leq m + |K| \tag{4.43}$$

Proof.

$$\sum_{q\in Q}\lambda^q = \sum_{q\in Q} x_q \sum_{i=1}^{m+|K|} \lambda_i^q = \sum_{i=1}^{m+|K|}\sum_{q\in Q} x_q\lambda_i^q \leq \sum_{i=1}^{m+|K|}\sum_{q\in Q}\lambda_i^q = \sum_{i=1}^{m+|K|} 1 = m + |K|$$

□

By using these lemmas we can prove the following theorem, that gives us the formulation for Dantzig-Wolfe relaxation of the (QPL) formulation:

Theorem 4.1. *A valid formulation of the Dantzig-Wolfe relaxation (4.29)-(4.32) is given by the following model:*

$$\min \sum_{q\in Q} c_q\lambda^q \qquad + (m - |K|) \tag{4.44}$$

$$s.t. \quad \sum_{q\in Q:\, g^k\in G^q} \lambda^q = 2 \qquad \forall\, k \in \{1,\ldots,|K|\} \tag{4.45}$$

$$\sum_{\substack{q\in Q:\, g^k\in G^q \\ h_p^q=1}} \lambda^q = 1 \qquad \forall\, k\in\{1,\ldots|K|\},\ p\in P:\ g_p^k = 2 \tag{4.46}$$

$$\lambda^q \in [0,1] \qquad \forall\, q\in Q \tag{4.47}$$

where G^q is a subset of genotypes in K and h^q is a haplotype associated with variable λ^q.

Proof. The proof of this Theorem is a direct consequence of the Lemmas 4.1, 4.2, 4.3 and 4.4, where vector h^q represents an haplotype whose coordinates are described by z_q. Note that constraint (4.43) does not appear in the formulation as it is redundant. It remains to prove that variables λ^q take value between 0 and 1, which derives from the following observations:

$$\lambda^q = x_q \sum_{i=1}^{m+|K|} \lambda_i^q \geq 0 \qquad \text{because } x_q \in \{0,1\} \text{ and } \lambda_i^q \geq 0\ \forall\, i, q$$

$$\lambda^q \leq 1 \qquad \text{as a consequence of (4.46)}$$

□

We can now give the *Dantzig-Wolfe reformulation* of (QPL) model.

Theorem 4.2. *A valid reformulation of (QPL) model is given by the following formulation:*

$$\text{(EIP)}\ \min \sum_{q\in Q} c_q\lambda^q \qquad + (m-|K|) \tag{4.48}$$

$$s.t. \sum_{q\in Q:\ g^k\in G^q} \lambda^q = 2 \qquad \forall\, k\in\{1,\ldots,|K|\} \tag{4.49}$$

$$\sum_{\substack{q\in Q:\ g^k\in G^q\\ h^q_p=1}} \lambda^q = 1 \qquad \forall\, k\in\{1,\ldots|K|\},\ p\in P:\ g^k_p=2 \tag{4.50}$$

$$\lambda^q\in\{0,1\} \qquad \forall\, q\in Q \tag{4.51}$$

Proof. Note that the linear relaxation of EIP is exactly the Dantzig-Wolfe relaxation given in the previous theorem. It is known [16] that a valid reformulation of the original model is obtained by imposing the integrality conditions on the original variables, as seen in (4.34)-(4.35):

$$x_i = \sum_{q\in Q} x_q\lambda^q_i \qquad \in\{0,1\} \tag{4.52}$$

$$y^k_i = \sum_{q\in Q} y^k_q\lambda^q_i \qquad \in\{0,1\} \tag{4.53}$$

$$z_{ip} = \sum_{q\in Q} z_{qp}\lambda^q_i \qquad \in\{0,1\} \tag{4.54}$$

and proving that they imply the condition:

$$\lambda^q\in\{0,1\} \text{ for all } q\in Q.$$

Suppose by contradiction that there exists a subset K such that $\lambda^q_i\in]0,1[$ for each $q\in K$. As all vertices of conv(X) have only integer coordinates in $\{0,1\}$, because of (4.52) we must have

$$\sum_{q\in K}\lambda^q_i = 1, \quad \lambda^q_i = 0\ \forall\, q\notin K$$

thus either $x_q = 0$ for all $q\in K$ or $x_q = 1$:

- if $x_q = 0$ for all $q\in K$, then by constraints (4.20) we have also $y^k_q = 0$ for all k and for all $q\in K$. Thus, as the considered vertices in K should be different, we should have different vectors z.
  Anyway, if $\sum_{q\in K} z_{qp}\lambda^q_i = 0$, we have $z_{qp} = 0$ for all $q\in K$, while if $\sum_{q\in K} z_{qp}\lambda^q_i = 1$ we need $z_{qp} = 1$ for each $q\in K$, so that each vertex $q\in K$ results to be equal. This is not possible, so the hypothesis was wrong and all λ^q_i is in $\{0,1\}$.

- if $x_q = 1$ for all $q\in K$, we must have by constraints (4.21) at least one $y^k_q = 1$. But then, we must have $y^k_q = 1$ for all $q\in K$. Now, values of z_{qp} are fixed if there exists a genotype k such that $g^k_p\neq 2$. To ensure the integrality of z_{ip} we should have no genotype such that $g^k_p = 0$. Under this assumption, we still need $z_{qp} = 1$ for all $q\in K$, hence all vertices in K results equal. This is absurd, so every λ^q_i takes value 0 or 1.

Variables λ^q are then integer for every $q \in Q$ by (4.39), as they are defined as sum of integer amounts, and are forced to take values 0 or 1 by the domain of the Dantzig-Wolfe relaxation.

$\square$

Note that model (EIP) corresponds to the exponential-sized formulation for HIPP presented in [19], where the variables represent a pair $q = (h^q, G^q)$ where h^q is a haplotype and $G^q \subseteq G$ is a subset of genotypes compatible with h^q. From now on, we define a set Q of all these pairs, made of a haplotype and a subset of genotypes compatible with it, and we denote each element of this set as a Q-pair. In this way, if we consider the integer model, setting a variable λ^q equal to 1 means that we choose haplotype h^q as one of the two haplotypes used to resolve all the genotypes in G^q. Constraints (4.49) ensure that each genotype is contained in exactly two subsets of genotypes, meaning we are selecting two haplotypes to resolve it, while constraints (4.50) say that, among the two haplotypes chosen for each genotype, only one has value 1 in each heterozygous position p (the other has necessarily value 0).

This formulation has a polynomial number of constraints, and $O(2^{n|K|})$ variables, so that column-generation and branch-and-price approaches are necessary to solve it. These approaches will be presented in the following chapters.

CHAPTER 5

A column-generation approach for the linear relaxation

The first step for solving an integer linear program consists in finding the solution of its linear relaxation, that requires some extra arrangements when computed using a column-generation approach: first of all, we should provide a feasible solution as starting point of the algorithm, then we should provide an efficient procedure to solve the pricing problem and find suitable variables to be added to the RMP (Restricted Master Problem).

The huge number of variables involved in the formulation are likely to yield computational issues, in particular we expect a very slow improvement of the solution value when we are considerably close to the optimum. By using a lower bound as an early termination condition we can reduce this effect, stopping the column generation whenever the difference between the current optimal value and the lower bound is below a tolerance threshold. Moreover, it is usually seen that the dual variable values do not smoothly converge to their optimum, but that, mainly during the first iterations, they significantly oscillate. Thus, a stabilization procedure can be used to bound the values of the dual variables and accelerate the convergence.

In this chapter we introduce two heuristics that yield feasible initial solutions for the HIPP problem, then we present different ways of solving the pricing problem. We then introduce the computation of a lagrangian lower bound used as a termination condition as well as an important tool for the stabilization procedure described in the last section. In order to apply the proposed stabilization technique, we extend the number of cases in which convergence is ensured with respect to previous approaches in literature.

5.1 Heuristics for computing an initial solution of the Restricted Master Problem

An easy way of providing a feasible initial solution to the column-generation procedure consists in building two compatible haplotypes for each genotype. In our case, we just need those for genotypes with at least one heterozygous site, as fixed genotypes are assumed to be already resolved and are already taken into account. In particular, given a genotype g, two haplotypes h' and h'' resolving g are built in the following way:

$$h'_p = \begin{cases} g_p & \text{if } g_p \neq 2 \\ 1 & \text{otherwise} \end{cases} \qquad\qquad h''_p = \begin{cases} g_p & \text{if } g_p \neq 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The drawback of this initial solution is its probably large distance from the optimal solution, as it is not taking into account the compatibilities between genotypes that could spare non-necessary haplotypes. Thus, it is possible to consider alternative initial solutions that exploits the structure of the genotypes and reduce the associated value of the objective function. Such alternative solutions can be found by means of heuristics taken from the literature. In this thesis, we focus on the ability of the proposed column generation approach to converge to the optimal solution and we propose a fast heuristic inspired by the haplotype inference technique presented by Clark in [15] and reported in Algorithm 3.6. As the procedure in [15] does not guarantee that all genotypes are resolved, we modify it as in Algorithm 5.10 in order to obtain in any case a feasible solution, even when no fixed genotypes are given. This procedure follows Clark's heuristic, except for the cases in which we do not find a compatible haplotype for a genotype g: in this case, we arbitrarily choose two haplotypes that resolve g and add them to the set H (see Step 17). Note that to perform this step we can use the approach shown in (5.1).

Algorithm 5.10 Clark's inspired heuristic

```
1: procedure INITIAL SOLUTION
2:     let $\bar{G} = G$ be the set of unresolved genotypes,
3:     let $H = \varnothing$ be the set that contains the haplotypes considered so far,
4:     for $g$ in $\bar{G}$ do
5:         if $g$ is fixed then
6:             add $g$ to $H$
7:             remove $g$ from $\bar{G}$,
8:         end if
9:     end for
10:    for $g$ in $\bar{G}$ do
11:        look for a haplotype $h \in H$ compatible with $g$,
12:        if $h$ is found then
13:            compute the complementary one $\bar{h}$ w.r.t. $g$,
14:            if $\bar{h} \notin H$ then and add it to $H$,
15:            end if
16:        else
17:            add to $H$ any two haplotypes that together resolve $g$,
18:        end if
19:    end for
20: end procedure
```

Moreover, we recall that in order to find the maximum number of genotypes that can be resolved, we should run Clark's heuristic many times, considering all the possible orderings for the set of genotypes G. In this case, however, we choose a predefined order that tries to exploit the compatibility of haplotypes. In particular, a possible choice consists in considering the genotypes in set G rearranged according to the increasing number of heterozygous positions.

The intuition behind such a choice is shown by the following example.

Example 5.1. *Consider a set G of genotypes in a random ordering and find a feasible starting solution applying Algorithm 5.10:*

$$G = \{2021022, 1201210, 2021020, 1101110\}$$
$$\Rightarrow\ H = \{0001000, 1011011, 1001010, 1101110, 1011010\}$$

Consider now the same set G with genotypes rearranged according to the increasing number of heterozygous sites. The initial solution obtained in this case is:

$$G = \{1101110, 1201210, 2021020, 2021022\}$$
$$\Rightarrow\ H = \{1101110, 1001010, 0011000, 0011001\}$$

As we can see, the first ordering does not completely exploit the information given by the genotype with no heterozygous sites for resolving the second genotype, so that the resulting feasible solution needs an extra haplotype.

5.2 Different approaches to solve the Pricing Problem

Given the feasible initial solution, we consider the linear relaxation of (EIP), that we call (ELP), and we build its RMP, where the only variables involved are the ones related to the initial feasible solution and belonging to set $Q_{RM} \subseteq Q$:

$$\min \sum_{q \in Q_{RM}} c_q \lambda^q \qquad + (m - |K|) \tag{5.2}$$

$$s.t. \sum_{q \in Q_{RM}:\, g^k \in G^q} \lambda^q = 2 \qquad \forall\, k \in \{1, \dots, |K|\} \tag{5.3}$$

$$\sum_{\substack{q \in Q_{RM}:\, g^k \in G^q \\ h^q_p = 1}} \lambda^q = 1 \qquad \forall\, k \in \{1, \dots |K|\},\ p \in P : g^k_p = 2 \tag{5.4}$$

$$\lambda^q \in \{0, 1\} \qquad \forall\, q \in Q_{RM} \tag{5.5}$$

We solve this RMP, thus obtaining a feasible primal solution for the linear relaxation. If the associated dual solution is also feasible, then we do not need any other variable and the solution is optimal. If the dual solution is not feasible, instead, we need to find a variable with negative reduced cost to be added to the current set of available variables. This is done by solving a subproblem, called pricing problem, as seen in Section 2.1.2.

We associate dual variables π^k, $k \in \{1, \dots, |K|\}$ and $\mu^k_p, p \in P, k \in \{1, \dots, |K|\}$ respectively to primal constraints (5.3) and (5.4). Observing that constraints $\lambda^q \leq 1$

are redundant due to constraints (4.50), the dual of (ELP) is the following:

$$\max \sum_{k=1}^{|K|} 2\,\pi^k + \sum_{k=1}^{|K|} \sum_{p: g_p^k=2} \mu_p^k \tag{5.6}$$

$$s.t. \sum_{k:\, g^k \in G^q} \pi^k + \sum_{k:\, g^k \in G^q} \sum_{p:\, g_p^k=2, h_p^q=1} \mu_p^k \leq c_q \qquad \forall\, q \in Q \tag{5.7}$$

$$\pi^k \geq 0 \qquad \forall\, k \in \{1, \ldots, |K|\} \tag{5.8}$$

Variables with negative reduced cost are found if there exist violated constraints for the dual problem (5.6)-(5.8). In particular, given the dual variables (π_{RM}, μ_{RM}) related to the solution of the RMP, a Q-pair $q = (h^q, G^q)$ is associated to a suitable variable λ^q for the RMP if it holds:

$$\bar{c}_q = c_q - \sum_{k:\, g^k \in G^q} \pi_{RM}^k - \sum_{k:\, g^k \in G^q} \sum_{p:\, g_p^k=2, h_p^q=1} {\mu_{RM}}_p^k < 0 \tag{5.9}$$

where $\bar{c}_q$ is the reduced cost associated to Q-pair q. If no such Q-pair exists, then the solution obtained from the RMP is optimal for the HIPP problem.

Thus, we either need a tool that gives us a Q-pair q such that $\bar{c}_q < 0$ or proves that no such Q-pair exists. This is the so-called Pricing Problem. Note that these two queries are answered at once if we are able to determine the Q-pair that gives the minimum $\bar{c}_q$: if this value is negative, we found a suitable Q-pair, otherwise we know that it holds

$$\bar{c}_{\tilde{q}} \geq \min_{q \in Q} \bar{c}_q \geq 0,$$

so we have the proof that no other Q-pair $\tilde{q}$ is suitable.

We give the following formulation of the pricing problem (PP), whose solution represents a Q-pair with minimum reduced cost:

$$\text{(PP)} \ \min c(\zeta) - \sum_{k=1}^{|K|} \pi_{RM}^k \chi^k - \sum_{k=1}^{|K|} \sum_{p: g_p^k=2} {\mu_{RM}}_p^k \zeta_p \chi^k \tag{5.10}$$

$$s.t.\ \zeta_p \leq 1 - \chi^k \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P : \ g_p^k = 0 \tag{5.11}$$

$$\zeta_p \geq \chi^k \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P \ : \ g_p^k = 1 \tag{5.12}$$

$$\zeta_p, \chi^k \in \{0, 1\} \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P \tag{5.13}$$

Variables ζ_p represent the coordinates of a haplotype h^q, χ is the characteristic vector of a subset $G^q \subseteq K$ of genotypes and $c(\zeta) = 0$ if ζ is a fixed haplotype and it is equal to 1 otherwise. Constraints (5.11) and (5.12) guarantee the compatibility conditions between the haplotype and the genotypes.

Note that this formulation presents a quadratic objective function, so that in order to apply standard linear programming techniques to exactly solve (PP) a linearization is necessary, as will be shown in Section 5.2.1. This solution approach, however, involves the addition of a new set of variables to the formulation (PP). In Section 5.2.2 we present alternative ways of finding the optimal solution of (PP) exploiting some features of this formulation.

We can further notice that in some cases there can be more efficient ways of finding variables with negative reduced costs that do not yield the minimum value of $\bar{c}_q$. In these cases, if we are able to identify variables to be added to the RMP, we can fastly proceed towards the optimal solution, while when they fail in finding such a variable, we cannot yet yield the proof that no suitable variable exists. In Sections 5.2.3, 5.2.4, 5.2.5 and 5.2.6 we present different heuristic approaches to solve (PP), that aim at quickly finding a Q-pair q with negative reduced cost.

5.2.1 Solve the Mixed Integer Program

As we already noticed, (PP) presents a quadratic objective function. A straightforward way of dealing with it consists in linearizing. We introduce new variables w_p^k for each genotype g^k and each position p such that $g_p^k = 2$. These variables are used in the formulation to represent the quantity $\zeta_p \chi^k$, so that the objective function can be re-written in linear form. We thus introduce extra constraints to ensure that these new variables assume the right value, that is w_p^k must be equal to 0 if at least one among ζ_p and χ^k is equal to 0 and 1 if both take value 1. The linear reformulation of the pricing problem, called (LPP), is given by:

$$\text{(LPP)}\ \min c(\zeta) - \sum_{k=1}^{|K|} \pi_{RM}^k \chi^k - \sum_{k=1}^{|K|} \sum_{p:g_p^k=2} \mu_{RM}{}_p^k w_p^k \tag{5.14}$$

$$s.t.\ \zeta_p \le 1 - \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P :\ g_p^k = 0 \tag{5.15}$$

$$\zeta_p \ge \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P\ :\ g_p^k = 1 \tag{5.16}$$

$$w_p^k \le \zeta_p \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P\ :\ g_p^k = 2 \tag{5.17}$$

$$w_p^k \le \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P\ :\ g_p^k = 2 \tag{5.18}$$

$$w_p^k \ge \chi^k + \zeta_p - 1 \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P\ :\ g_p^k = 2 \tag{5.19}$$

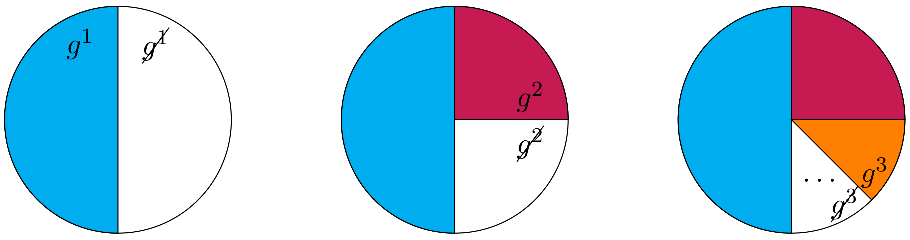
$$\zeta_p, \chi^k, w_p^k \in \{0,1\} \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P \tag{5.20}$$

Note that (LPP) can be solved using standard solvers for integer linear problems.

5.2.2 A Smart Enumeration approach

The procedure presented in this section exploits the particular structure of the variables involved in (LPP): it reduces its size by partitioning the set of feasible solutions and takes advantage from the information deduced.

The main idea underlying this approach is the obvious observation that, given a genotype $g^1 \in K$, the solution (h^q, G^q) of the pricing problem either will include g^1, or not. For compatibility reasons, if $g^1 \in G^q$ we can derive extra information on the shape of the solution, such as the values of some components of h^q, and already exclude from G^q those genotypes that are not compatible with g^1. In this way, the size of (LPP) can be significantly reduced, provided we suppose g^1 belongs to the optimal solution. We then need to look also to what happens when g^1 is not in the solution. Consider Figure 5.1, where the circle represents all the possible solutions (h^q, G^q) of the pricing problem. All the solutions in which $g^1 \in G^q$ are colored in blue in Figure 5.1a. In Figure 5.1b, we can see how the solutions not involving g^1 are



(a) $g^1 \in G^q$ (b) $g^1 \notin G^q$,$g^2 \in G^q$ (c) $g^1 \notin G^q$,$g^2 \notin G^q$, $g^3 \in G^q$

Figure 5.1: Idea of how Smart Enumeration proceeds.

divided into solutions in which $g^2 \in G^q$ (in purple) and those in which $g^2 \notin G^q$, that are again divided into solutions such that $g^3 \in G^q$ (in orange in Figure 5.1c) or not. Thus, by exploring recursively all the genotypes in K and solving the associated pricing problems with additional information, we are able to recover the optimal solution of the original pricing problem. We give the outline of this procedure in Algorithm 5.11. Note that at Step 8 of this algorithm we are solving a linear program deriving from the

Algorithm 5.11 Smart Enumeration for solving (LPP)

```
 1: procedure SMART ENUMERATION
 2:     set $\alpha^{OPT} = +\infty$, $(\zeta^{OPT}, \chi^{OPT}) = 0$,
 3:     given a predefined order $\prec$ on the genotypes,
 4:     for $g^i$ genotype do
 5:         fix $\chi^i = 1$ and $\chi^k = 0 \, \forall \, k : \; g^k \prec g^i$,
 6:         fix $\zeta_p = g^i_p \, \forall p \in P : \; g^i_p \neq 2$,
 7:         set $G_i = \{g^i\} \cup \{g^j \in K | \, g^i \prec g^j$ and $g^i_p + g^j_p \neq 1, \, \forall p \in P\}$
 8:         solve (LPP) restricted to the genotypes in $G_i$
 9:         - get the optimal value $\alpha^i$,
10:         - get the optimal solution $(\zeta^{(i)}, \chi^{(i)})$
11:         if $\alpha^i < \alpha^{OPT}$ then
12:             update $\alpha^{OPT} = \alpha^i$, $(\zeta^{OPT}, \chi^{OPT}) = (\zeta^{(i)}, \chi^{(i)})$,
13:         end if
14:     end for
15: end procedure
```

formulation (LPP) but whose size is reduced, so that the optimal value α^i is obtained by solving the following reduced pricing problem:

$$\alpha^i = \min c(\zeta) - \sum_{p: g^i_p = 2} \mu^i_p \zeta_p - \sum_{k \in G_i} (\pi^k + \sum_{\substack{p:\, g^k_p = 2, \\ g^i_p = 1}} \mu^k_p) \chi^k - \sum_{k \in G_i} \sum_{\substack{p: g^k_p = 2 \\ g^i_p = 2}} \mu^k_p \zeta_p \chi^k \tag{5.21}$$

$$\text{s.t.} \quad \zeta_p = 1 \qquad \text{if } g^i_p = 1 \tag{5.22}$$

$$\zeta_p = 0 \qquad \text{if } g^i_p = 0 \tag{5.23}$$

$$\zeta_p \leq 1 - \chi^k \qquad \forall \, k \in G_i, \forall \, p : \; g^k_p = 0 \tag{5.24}$$

$$\zeta_p \geq \chi^k \qquad \forall \, k \in G_i, \forall \, p : \; g^k_p = 1 \tag{5.25}$$

$$\zeta_p, \chi^k \in \{0,1\} \qquad \forall\, k \in G_i, \forall\, p \tag{5.26}$$

Then, we need to keep recorded the minimum value of these α^i to recover the optimal solution of the pricing problem (LPP).

We point out that, in order to provide a partition of all the feasible solutions, at Step 3 the Smart Enumeration procedure needs to look at each genotype following a predefined order. In particular, we note how considering a genotype with only few heterozygous sites allows us to fix a large number of variables and could possibly reduce the number of compatible genotypes to be considered. It is then advisable to keep the genotypes with many heterozygous sites at the end of the procedure, so that the disadvantage of still having a great number of non-fixed variables is compensated by the fact that only few genotypes are left to be considered. Thus, it makes sense to consider the genotypes ordered according to the increasing number of heterozygous sites.

We still need to prove that this procedure is correct. This is shown in the following proposition.

Proposition 5.1. *The* Smart Enumeration *described in Algorithm 5.11 solves (LPP) to optimality:*

Proof. By fixing variables χ in Step 5, we obtain a partition of the solution space of (LPP) into $|K|$ subsets, as shown in Figure 5.1. Let $(\bar{h}, \bar{G})$ be a Q-pair associated to a feasible solution of the i-th reduced pricing problem. Since $g^i \in \bar{G}$, $\bar{h}$ has to be compatible with g^i so that fixing variables as in Step 6 does not exclude any feasible solution. Now, let $g^j \in \bar{G}$ with $g^j \neq g^i$. From Step 5 we necessarily have $g^i \prec g^j$. Moreover, $g^j_p + g^i_p \neq 1$ for all $p \in P$, as otherwise g^j and g^i cannot be resolved by the same $\bar{h}$. Hence the definition of G_i does not exclude any feasible solution. □

5.2.3 Fixed haplotypes

According to the observation that, in order to find a variable with negative reduced cost, we do not need to solve (PP) to optimality, we introduce here a simple procedure that explores a subset of all the possible Q-pairs looking for a suitable variable to be added to the RMP.

We recall that if a genotype does not contain any heterozygous site, it is resolved by the so-called fixed haplotypes. Note that we can partition the whole set of Q-pairs according to whether h^q is fixed or not. Considering a fixed haplotype, the corresponding pricing problem derived from model (PP) results to be a linear program:

$$(FPP)\min\; -\sum_{k=1}^{|K|} \pi^k_{RM}\chi^k - \sum_{k=1}^{|K|} \sum_{p: g^k_p=2, h^q_p=1} \mu_{RM}{}^k_p \chi^k \tag{5.27}$$

$$s.t.\; \chi^k = 0 \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P:\; g^k_p \neq 2 \wedge g^k_p \neq h^q_p \tag{5.28}$$

$$\chi^k \in \{0,1\} \qquad \forall\, k \in \{1,\ldots,|K|\} \tag{5.29}$$

whose solution can be found by inspection as described by Algorithm 5.12. Namely, starting from the given haplotype we add to its subset of associated genotypes only

Algorithm 5.12 Best candidate fixed haplotype

```
 1: procedure FIXED HAPLOTYPES
 2:     for $g \in G$ compatible with $h^q$ do
 3:         compute $c_k = -{\pi_{RM}}^k - \sum_{p:g_p^k=2,h_p^q=1} {\mu_{RM}}_p^k$,
 4:         if $c_k \leq 0$ then
 5:             set $\chi^k = 1$,
 6:         else
 7:             set $\chi^k = 0$,
 8:         end if
 9:     end for
10: end procedure
```

those genotypes that are compatible with it and give a negative contribution to the objective function. Note that in this way we obtain the best possible subset of genotypes for a specific haplotype, according to the minimality of the reduced cost. Thus, solving this linear problem for each fixed haplotype and storing the best Q-pair found, we identify the variable with minimum reduced cost among those involving fixed haplotypes. If this reduced cost is positive, no variable with fixed haplotype is suitable to be added to the RMP.

5.2.4 A Local Search-based approach

A different heuristic for finding a suitable variable can be made by applying a local search starting from the most promising fixed haplotype $h^{\bar{q}}$ with positive reduced cost $\bar{\alpha}$. This procedure is described in Algorithm 5.13. The initial haplotype is found as part of the optimal solution $\bar{q} = (h^{\bar{q}}, G^{\bar{q}})$ of (FPP) model described in Section 5.2.3. In this case, we define the neighbourhood of this Q-pair as the Q-pairs in which the haplotype $\tilde{h}$ is obtained from $h^{\bar{q}}$ by flipping the value of a position p and the related optimal subset of genotypes is obtained by solving model (FPP) (see Steps 5-6). Whenever a neighbour solution that improves the objective function is found, we apply again the local search starting from this new feasible solution. Note that, if no fixed haplotype is available, the local search can start from a random haplotype. This procedure is not exact, but can possibly provide a Q-pair whose associated variable has a negative reduced cost for the RMP.

5.2.5 An Early-terminated Smart Enumeration

The main drawback of the Smart Enumeration procedure presented in Section 5.2.2 is that to solve exactly (LPP) we need to find the optimal solution of $|K|$ integer linear programs. In order to speed up the process, instead of looking for the minimum reduced cost and considering the associated solution, we can stop the procedure as soon as we find a solution with a negative reduced cost. In fact, in this way we can recover a Q-pair from the solution χ, ζ that is associated with a variable with negative reduced cost, thus we find a suitable variable to be added to the RMP. In the worst case, we need to solve all the $|K|$ problems in order to find a variable to be added, or to be able to say that no such variable exists. This *Early-terminated Smart Enumeration* procedure does not always return the optimal solution of the pricing problem, but as

Algorithm 5.13 Local Search applied to the solution of (FPP)

```
1: procedure LOCAL SEARCH($\bar{q}$)
2:     set $q^{LS} = \bar{q}$, $\alpha^{LS} = \bar{\alpha}$,
3:     set $q^{BEST} = q^{LS}$, $\alpha^{BEST} = \alpha^{LS}$,
4:     for each position $p \in P$ do
5:         $\tilde{h} = h^{q^{LS}}$,
6:         $\tilde{h}_p = 1 - h_p^{q^{LS}}$,
7:         solve (FPP) with $\tilde{h}$ fixed, get
8:         - optimal solution $(\tilde{h}, \tilde{G})$,
9:         - optimal value $\tilde{\alpha}$
10:        if $\tilde{alpha} < \alpha^{BEST}$ then
11:            update: $\alpha^{BEST} = \tilde{\alpha}$, $q^{BEST} = (\tilde{h}, \tilde{G})$,
12:        end if
13:    end for
14:    if $q^{BEST}$ has been updated then
15:        update $q^{LS} = q^{BEST}$,
16:        go back to Step 3,
17:    else
18:        STOP.
19:    end if
20: end procedure
```

soon as we solve all the $|K|$ reduced problems without finding a suitable variable, we know that considering the solution with minimum reduced cost obtained so far we are also considering the global optimal solution of the pricing problem, so that we can assert that no variable is missing and we reached the optimal solution. This means that the correctness of the procedure is preserved.

5.2.6 A Maximum flow approach

By assuming that we already checked that no fixed haplotypes have negative reduced cost, we can restrict the pricing problem to non-fixed haplotypes and rewrite the objective function of (LPP): we ignore the term $c(\zeta)$ and modify the pricing problem into a maximization problem, as follows:

$$\text{(MaxPP)} \quad \max \sum_{k=1}^{|K|} \pi_{RM}^k \chi^k + \sum_{k=1}^{|K|} \sum_{p:g_p^k=2} \mu_{RM}{}_p^k w_p^k \tag{5.30}$$

$$s.t.\ \zeta_p \le 1 - \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P:\ g_p^k = 0 \tag{5.31}$$

$$\zeta_p \ge \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P:\ g_p^k = 1 \tag{5.32}$$

$$w_p^k \le \zeta_p \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P:\ g_p^k = 2 \tag{5.33}$$

$$w_p^k \le \chi^k \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P:\ g_p^k = 2 \tag{5.34}$$

$$w_p^k \ge \chi^k + \zeta_p - 1 \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P:\ g_p^k = 2 \tag{5.35}$$

$$\zeta_p, \chi^k, w_p^k \in \{0,1\} \qquad \forall\, k \in \{1,\dots,|K|\}, p \in P \tag{5.36}$$

Variables w_p^k can be seen as associated to subsets made by the pair (genotype-position) (g^k, p), where if we select a particular subset by setting $w_p^k = 1$, we need to take into account the associated elements and thus set $\chi^k = 1$, $\zeta_p = 1$. Under this consideration, the objective function reminds the one of selection problems shown in Section 2.5.1 that considers elements $n \in N$ with cost $c_n \geq 0$ and a collection Σ of subsets of elements of N with a profit $p_\sigma \geq 0$, $\sigma \in \Sigma$. We recall that the formulation for the selection problem is:

$$\begin{aligned} \max \quad & \sum_{\sigma\in\Sigma} p_\sigma y_\sigma - \sum_{n\in N} c_n x_n && && (5.37)\\ \text{s.t.} \quad & x_n \geq y_\sigma && \forall\, \sigma \in \Sigma, \forall\, n \in \sigma && (5.38)\\ & x_n,\, y_\sigma \in \{0,1\} && \forall\, n \in N, \forall\, \sigma \in \Sigma && (5.39) \end{aligned}$$

where variables x_n take value 1 if we select element $n \in N$ or 0 otherwise, and variables y_σ take value 1 if set $\sigma \in \Sigma$ is selected, 0 otherwise. We note however that there are some differencies, in particular, we do not know the sign of the objective's coefficients and due to the compatibility conditions we have extra constraints (5.31) and (5.32).

In order to see whether we can decompose the pricing problem and embed the compatibility constraints into the objective function, we recall some notions on the s-cliques presented in [33]:

- a set of genotypes C is an s-clique if there exists a haplotype h such that the set of genotypes compatible with it is exactly C,

- a selector for the s-clique C is a haplotype whose compatible genotypes are exactly the ones contained in C. The number of selectors for each s-clique can be very large, thus we can represent them in a compact form (pattern table),

- a pattern s is an n-string over $\{0, 1, -\}$ that represents the hypercube

  $$Q(s) = \{h \in \{0,1\}^n \,|\, h_i = s_i \;\forall\, i :\, s_i \neq -\}.$$

  A pattern table that represents all the selectors of an s-clique is made of a collection of patterns, and it is in standard form if no pattern of the set is contained in another pattern and each pair of patterns differ for at least two positions.

Lemma 5.1. *The set of all possible solutions to (maxPP) can be partitioned according to each s-clique and each pattern belonging to its pattern table in standard form.*

Proof. In [33] it is proved that the set of s-cliques $\mathcal{C}$ induces a partition of the set H_G of all possible haplotypes, thus we can consider an s-clique at a time.

Moreover, if s^1 and s^2 are two patterns deduced from the s-clique tree, they belong to two different branches of the tree, so that there exists at least a position p such that $s_p^1 = 0$ and $s_p^2 = 1$ (or viceversa). Thus, the same haplotype h cannot be deduced from both patterns. This proves that the patterns of a pattern table in standard form induce a partition of all the haplotypes compatible with the genotypes of that s-clique. So, for each s-clique we can consider a pattern at a time. □

Given an s-clique C and a pattern s belonging to its pattern table, we define a constant α_s^k as:

$$\alpha_s^k = \sum_{p=1, s_p \neq -, g_p^k = 2}^{n} \mu_{RM}{}_p^k s_p$$

Observation 5.1. *Let $\mathscr{C}$ be the set of s-cliques associated to G and $\mathscr{S}_C$ be the pattern table in standard form of the s-clique C, obtained from the s-clique tree. Then the solution of (MaxPP) can be computed by decomposing it into smaller programs related to each s-clique $C \in \mathscr{C}$ and each pattern $s \in \mathscr{S}_C$, as follows:*

$$\begin{array}{llll}
\max\limits_{C, s \in \mathscr{S}_C} & \max & \sum\limits_{k:\, g^k \in \bar{C}} \pi_{RM}^k \chi^k + \sum\limits_{k:\, g^k \in \bar{C}} \sum\limits_{p=1, s_p = -}^{n} \mu_{RM}{}_p^k w_p^k + \sum\limits_{k:\, g^k \in \bar{C}} \alpha_{\bar{C},s}^k \chi^k & (5.40) \\
& s.t. & w_p^k \le \zeta_p \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P \,:\, s_p = - & (5.41) \\
& & w_p^k \le \chi^k \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P \,:\, s_p = - & (5.42) \\
& & w_p^k \ge \chi^k + \zeta_p - 1 \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P \,:\, s_p = - & (5.43) \\
& & \chi^k, \zeta_p, w_p^k \in \{0, 1\} \qquad \forall\, k : \, g^k \in \bar{C}, \forall\, p \in P : \, s_p = - & (5.44)
\end{array}$$

Observation 5.2. *According to each s-clique and each pattern, the maximization problem (5.40) is defined only for a subset of the original variables χ^k, $k \in \{1, \ldots, |K|\}$ and $p \in P$, corresponding to the genotypes belonging to the s-clique and the positions in the pattern for which the value is not fixed.*

We recover the complete Q-pair q represented by the solution of the pricing problem in the following way:

$$h_p^q = \begin{cases} \zeta_p & \text{if } s_p = - \\ s_p & \text{otherwise} \end{cases} \qquad \text{and} \qquad g^k \in G^q \text{ if } g^k \in \bar{C} \wedge \chi^k = 1$$

Note that the information contained in constraints (5.31) and (5.32) is now included in the information given by the pattern. We can then focus on how to solve (5.40) - (5.44) given an s-clique and a pattern.

Observation 5.3. *If all the coefficients $\pi_{RM}^k + \alpha_s^k$ are negative and $\mu_{RM}{}_p^k$ are positive, then constraints (5.35) would be redundant and the problem in 5.40 could be solved as a maximum flow problem.*

Following the approach proposed in [41], we can build a network (V, D) in which there are two subsets of nodes V_1, V_2, such that $V = V_1 \cup V_2$, associated respectively to variables χ^k, ζ_p the first subset and to variables w_p^k the second one. The set of directed arcs D of infinite capacity connect nodes w_p^k to node ζ_p (resp. χ^k) if and only if we have the constraint $w_p^k \le \zeta_p$ (resp. $w_p^k \le \chi^k$). The result is a bipartite network. At this point we add a source s and a sink t and link the source to all the nodes of the first set with a directed arc of capacity $\mu_{RM}{}_p^k$, the nodes χ^k to the sink with directed arcs of capacity $-(\pi_{RM}^k + \alpha_{\bar{C},s}^k)$ and nodes ζ_p to the sink with null capacity arcs, as shown in Figure 5.2. The next step consists in finding the maximum flow over this

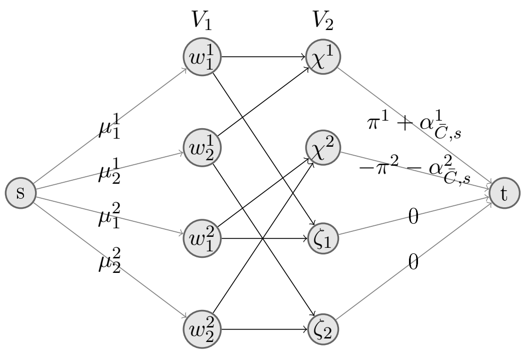


Figure 5.2: Bipartite network when coefficients have the right sign.

network, so that we can recover the solution of our problem looking at the minimum cut identified by the maximum flow: we will set to 1 all variables on the side of the source and to 0 those on the side of the sink.

The special scenario of Observation 5.3 is not likely to occur in our case, since the dual variables of RMP are not constrained in sign. This means that constraints (5.43) are not redundant and the exact problem cannot be solved using a maximum flow algorithm as known from [41, 42]. We can still solve a relaxation of this problem, ignoring terms of the objective function that do not have the required signs. As an alternative, we can try to force the coefficients of the objective function to assume the right sign by modifying the constraints of the RMP of (EIP). These approaches are illustrated in the following two sections.

Ignoring terms of the objective function

If in the objective function we ignore the coefficients that do not assume the required sign, we do not find the optimal solution of (MaxPP), but by applying the maximum-flow approach to a smaller network we recover a heuristic solution that still can provide a variable to be added to the RMP of formulation (EIP). We can proceed in the following way: if a coefficient μ_p^k is negative we remove variables w_p^k and if $\pi^k + \alpha^k_{\bar{C},s} > 0$ we remove variables χ^k. We then build the related network, as shown in Figure 5.3, where the nodes representing the removed variables are linked to source or sink by null weight arcs.

The heuristic solution can be found by solving a maximum flow problem and then looking at which nodes can be reached in the residual graph. The complete Q-pair q is then recovered in the following way:

$$h_p^q = \begin{cases} 1 & \text{if } \zeta_p \text{ is reached in the residual graph} \\ 0 & \text{otherwise} \end{cases}$$

$$g^k \in G^q \quad \text{if } \ \chi^k \text{ is reached in the residual graph}$$

If the Q-pair has negative reduced cost we have heuristically found a variable to add to the RMP.

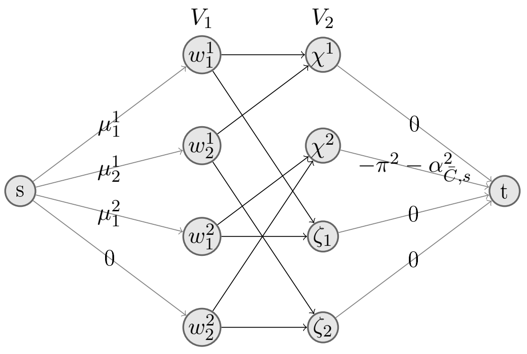


Figure 5.3: Bipartite network in which variables are ignored.

Constraining the signs of dual variables

Another way of controlling the dual variables' signs in (5.40) consists in modifying the constraints of the master problem, replacing equality constraints with suitable inequalities. We rewrite the (EIP) formulation in the following way, obtaining model (FullEIP):

$$\text{(FullEIP)}\ \min \sum_{q\in Q} c_q\lambda^q \qquad + (m - |K|) \tag{5.45}$$

$$\sum_{q: g^k \in G^q} \lambda^q \diamondsuit 2 \qquad \forall\, k = 1, \dots, |K| \tag{5.46}$$

$$\sum_{q: g^k \in G^q, h_p^q = 1} \lambda^q \heartsuit 1 \qquad \forall k = 1, \dots, |K|, p \in P : g_p^k = 2 \tag{5.47}$$

$$\sum_{q: g^k \in G^q, h_p^q = 0} \lambda^q \clubsuit 1 \qquad \forall k = 1, \dots, |K|, p \in P : g_p^k = 2 \tag{5.48}$$

$$\lambda^q \in \{0, 1\} \tag{5.49}$$

where we can substitute $\diamondsuit$, $\heartsuit$, $\clubsuit$ with the symbols $=$, $\leq$, $\geq$ in order to constrain the signs of the dual variables π^k_{RM}, ${\mu_{RM}}^k_p$, ${\eta_{RM}}^k_p$ associated respectively to constraints (5.46), (5.47) and (5.48). Note that, if we replace all symbols with $=$, one set of constraints among (5.46), (5.47) and (5.48) is redundant, in particular ignoring the second one we obtain exactly the former (EIP) formulation of the HIPP problem. Deriving the dual problem with all these three sets of constraints we get the following expression for the objective function of the pricing problem:

$$c_q - \sum_{k:\, g^k \in G^q} \pi^k_{RM} - \sum_{k:\, g^k \in G^q} \sum_{p:\, h_p^q = 1, g_p^k = 2} {\mu_{RM}}^k_p - \sum_{k:\, g^k \in G^q} \sum_{p:\, h_p^q = 0, g_p^k = 2} {\eta_{RM}}^k_p \tag{5.50}$$

We associate binary variables χ^k to each genotype, that take value 1 if genotype g^k is included in the subset of genotypes, and ζ_p that records the values of each position of a haplotype. Thus, the objective function of the new pricing problem can be written as follows:

$$c_q - \sum_{k=1}^{|K|} \pi_{RM}^k \chi^k - \sum_{k=1}^{|K|} \sum_{p:\, g_p^k=2} \mu_{RM}{}_p^k \zeta_p \chi^k - \sum_{k=1}^{|K|} \sum_{p:\, g_p^k=2} \eta_{RM}{}_p^k (1-\zeta_p)\chi^k \tag{5.51}$$

giving the following formulation for the related pricing problem:

$$\begin{aligned}
\min \quad & c(\zeta) - \sum_{k=1}^{|K|} (\pi_{RM}^k + \sum_{p:\, g_p^k=2} \eta_{RM}{}_p^k)\chi^k \quad - \sum_{k=1}^{|K|} \sum_{p:\, g_p^k=2} (\mu_{RM}{}_p^k - \eta_{RM}{}_p^k)\zeta_p \chi^k && (5.52)\\
\text{s.t.} \quad & \zeta_p \le 1 - \chi^k \qquad \forall\, k \in \{1,\dots,|K|\},\ p \in P : g_p^k = 0 && (5.53)\\
& \zeta_p \ge \chi^k \qquad \forall\, k \in \{1,\dots,|K|\},\ p \in P : g_p^k = 1 && (5.54)\\
& \chi^k, \zeta_p \in \{0,1\} \qquad \forall\, k \in \{1,\dots,|K|\},\ p \in P && (5.55)
\end{aligned}$$

For ease of notation, we define $\alpha^k = \sum_{p:\, g_p^k=2} \eta_{RM}{}_p^k$.
We would like to have negative coefficients $\pi_{RM}^k + \alpha^k$ and positive $\mu_{RM}{}_p^k - \eta_{RM}{}_p^k$. In order to see if this is possible, we consider all the combinations that can be assumed by the verses of constraints (5.46), (5.47) and (5.48), represented by $\Diamond$, $\heartsuit$, and $\clubsuit$. Notice that, by considering combinations in which a single set of constraints is not included, it is possible to have not only relaxations of the HIPP problem, but also valid formulations. For this reason, these cases are also taken into account. In Table 5.1 we list all the possible combinations for these symbols, denoting with the symbol $-$ when we are not considering that particular set of constraints. A ✓ is associated to those coefficients that assume the desired sign, a $\times$ to the ones that do not have the right sign. The combinations that give only a relaxation for the HIPP problem are highlighted in red. It is clear how, preserving the validity of the formulation, we are not able to constrain the signs of the coefficients, while it would be possible to have at least a set of coefficient of the right sign only if we consider a relaxation of (EIP) and hence of its RMP.

5.3 Adding multiple variables per iteration

We proposed different ways of solving the pricing problem (PP) associated to the (EIP) formulation for the HIPP problem, both based on exact methods and heuristics, in order to identify a variable to be added to the RMP. We propose here two different procedures that allow us to consider, at each iteration of the column-generation method, additional Q-pairs that can have higher possibilities of representing non-zero variables in the optimal solution of (ELP). The first consists in looking for a particular Q-pair with a larger genotypes subset, while the second one provides several different Q-pairs, in which the subset of genotypes can have larger as well as smaller size.

5.3.1 Adding variables with a larger genotypes' subset

If we exclude the approach based on network flows, we note that all the proposed methods for solving the pricing problem have the following common feature:

$\Diamond$	$\heartsuit$	$\clubsuit$	$\pi^k_{RM} + \alpha^k$	$\mu_{RM}{}^k_p - \eta_{RM}{}^k_p$	$\Diamond$	$\heartsuit$	$\clubsuit$	$\pi^k_{RM} + \alpha^k$	$\mu_{RM}{}^k_p - \eta_{RM}{}^k_p$
$=$	$=$	$\geq$	×	×	$\geq$	$\geq$	$\geq$	×	×
$=$	$=$	$\leq$	×	×	$\geq$	$\geq$	$\leq$	×	✓
$=$	$\geq$	$=$	×	×	$\geq$	$\leq$	$\geq$	×	×
$=$	$\leq$	$=$	×	×	$\geq$	$\leq$	$\leq$	×	×
$=$	$\geq$	$\geq$	×	×	$\leq$	$=$	$=$	×	×
$=$	$\geq$	$\leq$	×	✓	$\leq$	$=$	$\geq$	×	×
$=$	$\leq$	$\geq$	×	×	$\leq$	$=$	$\leq$	✓	×
$=$	$\leq$	$\leq$	×	×	$\leq$	$\geq$	$=$	×	×
$\geq$	$=$	$=$	×	×	$\leq$	$\leq$	$=$	×	×
$\geq$	$=$	$\geq$	×	×	$\leq$	$\geq$	$\geq$	×	×
$\geq$	$=$	$\leq$	×	×	$\leq$	$\geq$	$\leq$	✓	✓
$\geq$	$\geq$	$=$	×	×	$\leq$	$\leq$	$\geq$	×	×
$\geq$	$\leq$	$=$	×	×	$\leq$	$\leq$	$\leq$	✓	×
$\geq$	$\geq$	-	×	✓	$=$	-	$\leq$	×	✓
$\geq$	$=$	-	×	×	$\leq$	-	$\geq$	×	×
$\geq$	$\leq$	-	×	×	$\leq$	-	$=$	×	×
$=$	$\geq$	-	×	✓	$\leq$	-	$\leq$	✓	✓
$=$	$=$	-	×	×	-	$\geq$	$\geq$	×	×
$=$	$\leq$	-	×	×	-	$\geq$	$=$	×	×
$\leq$	$\geq$	-	✓	✓	-	$\geq$	$\leq$	✓	✓
$\leq$	$=$	-	✓	×	-	$=$	$\geq$	×	×
$\leq$	$\leq$	-	✓	×	-	$=$	$=$	×	×
$\geq$	-	$\geq$	×	×	-	$=$	$\leq$	✓	×
$\geq$	-	$=$	×	×	-	$\leq$	$\geq$	×	×
$\geq$	-	$\leq$	×	✓	-	$\leq$	$=$	×	×
$=$	-	$\geq$	×	×	-	$\leq$	$\leq$	✓	×
$=$	-	$=$	×	×	$=$	$=$	$=$	×	×

Table 5.1: Possible combinations of equalities and inequalities for (FullEIP) constraints

Observation 5.4. *Given the Q-pair q obtained from solving (PP) with one of the proposed exact or heuristic approaches, each genotype $g \in G^q$ gives a non-positive contribution in the computation of the reduced cost.*

In this way we discard those Q-pairs in which the subset of genotypes is larger as it includes also genotypes that give a positive contribution to the reduced cost, still maintaining its overall negativity. We recall that, once we fixed a haplotype h, the contribution c_k of a genotype g^k to the value of the objective function in the pricing problem is given by

$$c_k = -\Big(\pi^k + \sum_{p:\, g^k_p = 2, h_p = 1} \mu^k_p\Big). \tag{5.56}$$

As the aim of the HIPP problem is to minimize the number of haplotypes needed to resolve all the genotypes, it is clear that to achieve this minimum, even in the linear relaxation, we need to resolve as many genotypes as possible with the same haplotype.

Thus, given a Q-pair $(\bar{h}, \bar{G})$ such that all genotypes in $\bar{G}$ give a negative contribution, we want to enlarge the subset by as many genotypes as possible. This can be achieved by solving the following integer linear program:

$$\max \sum_{k:\, g^k \in G(\bar{h}) \setminus \bar{G}} x^k \tag{5.57}$$

$$\text{s.t.} \quad c(\bar{h}) - \sum_{k:\, g^k \in \bar{G}} (\pi^k + \sum_{p:\, g_p^k = 2, \bar{h}_p = 1} \mu_p^k) - \sum_{k:\, g^k \in G(\bar{h}) \setminus \bar{G}} (\pi^k + \sum_{p:\, g_p^k = 2, \bar{h}_p = 1} \mu_p^k) x^k \leq 0 \tag{5.58}$$

$$x^k \in \{0, 1\} \quad \forall\, k : g^k \in G(\bar{h}) \setminus \bar{G} \tag{5.59}$$

where we recall that $G(\bar{h})$ is the set of genotypes in K compatible with $\bar{h}$ and variables x^k are binary, taking value 1 if we consider genotype g^k in the enlarged subset, 0 otherwise.

Notice that it is exactly the formulation of a knapsack problem (see Section 2.5.2), in which every item has the same utility 1 and weight given by (5.56). Thus, we can easily compute the optimal solution applying a simple heuristic:

1. consider the genotypes suitable to be added to $\bar{G}$, that are those in $G(\bar{h}) \setminus \bar{G}$ and arrange them in increasing order with respect to (5.56),
2. add to $\bar{G}$ the first d genotypes if the constraint (5.58) is satisfied, but it would be violated with the addition of the $(d + 1)$-th genotype.

5.3.2 Adding a selected set of extra variables

Once one of the methods proposed for the pricing problem has identified a promising haplotype $\bar{h}$, we can recover several Q-pairs with a negative reduced cost: if $\bar{G} \subseteq K$ is the set of genotypes compatible with $\bar{h}$ that gives a negative contribution to the reduced cost, then a suitable subset to be paired with $\bar{h}$ is made of a subset of $\bar{G}$ plus some genotypes in $G(\bar{h}) \setminus \bar{G}$ such that the overall reduced cost is still negative. As we can choose $2^{|\bar{G}|} - 1$ different sets just for the subsets of $\bar{G}$, adding the variables associated to all these Q-pairs can easily be impractical, so that we choose just a few of them by the simple expanding heurisitic shown in Algorithm 5.14.

Algorithm 5.14 Add multiple variable at a time

```
 1: procedure ADD VARIABLES
 2:     initialize $S = 0$ if $\bar{h}$ is fixed, $S = 1$ otherwise and $G^\star = \varnothing$,
 3:     for $g^k \in G(\bar{h})$ do
 4:         compute the value $c_k$, as in (5.56),
 5:         if $S + c_k$ is negative then
 6:             add $g^k$ to $G^\star$,
 7:             update $S = S + c_k$,
 8:             add a variable associated to the $Q$-pair $(\bar{h}, G^\star)$,
 9:         end if
10:     end for
11: end procedure
```

At Step 2, we initialize the reduced cost S associated to the Q-pair we are generating according to whether $\bar{h}$ is fixed or not. By inspection of all the genotypes g^k compatible with $\bar{h}$, we build sets $G^\star$ and Q-pairs $(\bar{h}, G^\star)$ whose related reduced cost c_k is negative (see Steps 5-8. In this way, we consider both Q-pairs with a larger and smaller subset of genotypes associated, when compared with $\bar{G}$.

Note that the order given to the genotypes affects this choice of the multiple variables to be added to the RMP, as, according to the values taken by the dual variables, we can find more or less suitable subsets of genotypes. However, it is not straightforward to identify an a priori ordering of the genotypes to be preferred to another one.

5.4 Convergence issues

The solutions of large-scale problems are usually highly degenerate, thus while applying a column-generation procedure the solution process can be very slow both at the beginning and in the final iterations. In particular, at the end of the procedure there is a so-called *tailing off* effect: only small improvements are achieved at each iteration, so that the process of reaching the optimal solution becomes very long. Moreover, at the beginning a *head in* problem can occur, i.e. the column-generation procedure can take several iterations before finding a variable that, added to the RMP, is able to improve the objective value. This problem mainly arises from the unstable behaviour of the dual variables' values.

With the aim of overcoming these two issues, we derive a lower bound on the optimal solution of (EIP) by means of a Lagrangian relaxation. It is then used to reduce the tailing-off effect, as an alternative termination condition that stops the column generation when the difference between the optimal value and the lower bound is sufficiently small. As for the head-in problem, instead, we apply a stabilization tecnique that prevents the dual variables from oscillating too much and makes use of the computation of the lower bound to accelerate the convergence of the column generation.

We gather together in Table 5.2 the main notation that we use throughout this chapter.

In order to derive the lower bound, we consider (EIP) and we add a constraint to it that forces the number of haplotypes used to be less than or equal to a value M big enough to ensure that the constraint is redundant. From now on we then consider the following (NewEIP) formulation:

$$(NewEIP)\ \min \sum_{q\in Q} c_q \lambda^q \qquad + (m - |K|) \tag{5.60}$$

$$s.t. \sum_{q\in Q:\, g^k \in G^q} \lambda^q = 2 \qquad \forall k = 1, \dots, |K| \tag{5.61}$$

$$\sum_{\substack{q\in Q:\, g^k \in G^q,\\ h_p^q = 1}} \lambda^q = 1 \qquad \forall k = 1, \dots |K|,\ p \in P : g_p^k = 2 \tag{5.62}$$

$$\sum_{q\in Q} \lambda^q \le M \tag{5.63}$$

$\rho_{RM} = (\pi_{RM}, \mu_{RM}, \nu_{RM})$	dual variables for the RMP of (NewELP)
z_{RM}	optimal value of the RMP of (NewELP)
$\rho_{ST} = (\pi_{ST}, \mu_{ST}, \nu_{ST})$	stabilized dual variables
$d(\rho)$	value of the objective function for (NewELP)'s dual computed in ρ
$\bar{\rho} = (\bar{\pi}, \bar{\mu}, \bar{\nu})$	stabilized dual variables
z_{PP}	optimal value of (NewPP)
q^{PP}	optimal solution of (NewPP)
$v(q)$	value of the objective function of (NewPP) computed in $q \in Q$
z_{sPP}	optimal value of (sPP)
q^{sPP}	optimal solution of (sPP)
$\tau > 0$	tolerance for stabilization termination
$\epsilon > 0$	tolerance for early termination

Table 5.2: Notation

$$\lambda^q \in \{0,1\} \qquad \forall\, q \in Q \tag{5.64}$$

Its linear relaxation (NewELP) is obtained replacing the domain constraints (5.64) with $\lambda^q \geq 0$ for all $q \in Q$. We recall that the constant $m-|K|$ represents the number of fixed haplotypes. Then, at each iteration of the column-generation procedure we define $M = m-|K|+z_{RM}$, z_{RM} being the optimal value of the RMP of (NewELP). The RMP is now defined starting from the full formulation of (NewELP) and it considers only a subset of variables λ^q. Note that each feasible solution of (ELP) is also feasible for (NewELP) and viceversa, in fact constraint (5.63) is redundant for this formulation. In particular, heuristics that provide an initial solution for the column-generation approach applied to (ELP) also provide a feasible initial solution for the column generation applied to (NewELP). As for the structure of the pricing problem, instead, the addition of a new constraint to (ELP) leads to consider also an extra dual variable. We associate dual variables π^k_{RM} to constraints (5.61), ${\mu_{RM}}^k_p$ to constraints (5.62) and one variable ν_{RM} to constraint (5.63). The objective function of the (NewELP) dual problem becomes

$$z_{RM} = \sum_{k=1}^{|K|} 2\,\pi^k_{RM} + \sum_{k=1}^{|K|} \sum_{p:\, g^k_p = 2} {\mu_{RM}}^k_p + M\,\nu_{RM}$$

while the formulation of the pricing problem as an integer porgramming model (NewPP) is the following:

$$(NewPP)\ \min c^q - \sum_{k=1}^{|K|} \pi^k_{RM} \chi^k - \sum_{k=1}^{|K|} \sum_{p: g^k_p=2} {\mu_{RM}}^k_p \zeta_p \chi^k - \nu_{RM} \tag{5.65}$$

$$s.t.\ \zeta_p \leq 1 - \chi^k \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P : g^k_p = 0 \tag{5.66}$$

$$\zeta_p \geq \chi^k \qquad \forall\, k \in \{1, \ldots, |K|\}, p \in P : g^k_p = 1 \tag{5.67}$$

$$\chi^k, \zeta_p \in \{0,1\} \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P \tag{5.68}$$

5.4.1 A Lagrangian lower bound

Introducing formulation (NewELP) we can define a particular Lagrangian function and, from it, we derive a lower bound on the solution value of (NewELP).

Proposition 5.2. *Given the dual variables π_{RM}, μ_{RM}, at each iteration of the column-generation procedure applied to (NewELP), a lower bound on the optimal solution is given by:*

$$LB(\pi_{RM}, \mu_{RM}) = z_{RM} + Mz_{PP} \tag{5.69}$$

Proof. We consider a Lagrangian relaxation of (NewELP), that provides a lower bound on its optimal value. In particular we consider the one obtained relaxing constraints (5.61) and (5.62). Consider as Lagrangian coefficients the values (π_{RM}, μ_{RM}) that are part of the dual variables obtained solving the current RMP of formulation (NewELP), thus we can write the Lagrangian function $L(\pi_{RM}, \mu_{RM})$ with respect to these values as follows:

$$\begin{aligned} L(\pi_{RM}, \mu_{RM}) = \min \quad & \sum_{q\in Q} c_q \lambda^q - \sum_{k=1}^{|K|} \pi_{RM}^k \Big(\sum_{q:\, g^k \in G^q} \lambda^q - 2\Big) - \sum_{\substack{k=1\\ p:\, g_p^k=2}}^{|K|} \mu_{RM}{}_p^k \Big(\sum_{\substack{q:\, g^k\in G^q\\ h_p^q=1}} \lambda^q - 1\Big) \\ \text{s.t.} \quad & \sum_{q\in Q} \lambda^q \le M \\ & \lambda^q \ge 0 \;\forall\, q \in Q \end{aligned}$$

that we can rewrite as:

$$L(\pi_{RM}, \mu_{RM}) = \sum_{k=1}^{|K|} 2\pi_{RM}^k + \sum_{\substack{k=1\\ p:g_p^k=2}}^{|K|} \mu_{RM}{}_p^k + \tag{5.70}$$

$$+ \min_{\substack{\sum_{q\in Q}\lambda^q \le M\\ \lambda^q\ge 0\;\forall\, q\in Q}} \Big\{ \sum_{q\in Q} c_q\lambda^q - \sum_{k=1}^{|K|} \pi_{RM}^k \sum_{q:\, g^k\in G^q} \lambda^q - \sum_{\substack{k=1\\ p:\, g_p^k=2}}^{|K|} \mu_{RM}{}_p^k \sum_{\substack{q:\, g^k\in G^q\\ h_p^q=1}} \lambda^q \Big\} \tag{5.71}$$

The first part of the Lagrangian function, described in (5.70), is easily computed, as it is equal to $z_{RM} - M\nu_{RM}$. Instead, the second part shown in (5.71) requires to solve a linear program. We rewrite it in the following way:

$$\min_{\substack{\sum_{q\in Q}\le M\\ \lambda^q\ge 0\;\forall\, q\in Q}} \left\{ \sum_{q\in Q} \left[c_q - \sum_{\substack{k\le|K|\\ q:\, g^k\in G^q}} \pi_{RM}^k - \sum_{\substack{k\le|K|,\, q:\, g^k\in G^q\\ h_p^q=1,\, p:\, g_p^k=2}} \mu_{RM}{}_p^k \right] \lambda^q \right\} \tag{5.72}$$

Consider the particular $\tilde{q}$ that is found as solution of the program

$$\min_{q\in Q} \{ c_q - \sum_{\substack{k\le|K|\\ q:\, g^k\in G^q}} \pi_{RM}^k - \sum_{\substack{k\le|K|,\, q:\, g^k\in G^q\\ h_p^q=1,\, p:\, g_p^k=2}} \mu_{RM}{}_p^k \} \tag{5.73}$$

and the feasible solution of (5.72) given by $\lambda^q = M$ if $q = \tilde{q}$, $\lambda^q = 0$ otherwise. No other feasible values of λ^q, $q \in Q$ can further improve the objective function, thus we can say that this solution is optimal for the linear program in (5.72). Moreover, the function to be minimized in (5.73) is almost equal to the objective function of (NewPP), apart from a constant ν_{RM}. Thus, the optimal value obtained by solving (5.73) is exactly equal to $z_{PP} + \nu_{RM}$.

We can now rewrite the Lagrangian function considering these results and obtaining the following lower bound:

$$L(\pi_{RM}, \mu_{RM}) = z_{RM} - M\nu_{RM} + M(z_{PP} + \nu_{RM}) = z_{RM} + Mz_{PP}$$

□

Note that this same lower bound can be derived in another, more intuitive way: Let z_{PP} be the value of the optimal solution of (NewPP). If $z_{PP} \geq 0$, then the dual constraints are all satisfied, hence the column generation can stop because the dual solution is also feasible. Otherwise, we have $z_{PP} < 0$ and we can consider the following dual solution:

$$(\pi_{RM}, \mu_{RM}, \nu_{RM} + z_{PP})$$

As $\nu_{RM} + z_{PP}$ is still negative and the dual constraints are satisfied for each $q \in Q$, we deduce that this is a feasible dual solution and in particular that it has value

$$\sum_{k=1}^{|K|} 2\pi_{RM}^k + \sum_{k=1}^{|K|} \sum_{p:\, g_p^k = 2} \mu_{RM}{}_p^k + M(\nu_{RM} + z_{PP}) = z_{RM} + Mz_{PP}.$$

From the weak duality property, we know that this value gives a lower bound on the solution of (NewELP), and in fact we obtained exactly the lower bound in (5.69).

5.4.2 A stabilization method to smooth dual variables' values

It has been observed that the values assumed by the dual variables at each iteration of a column-generation procedure usually do not smoothly converge towards the optimal ones [35]. This computational issue occurs also in our case, while solving model (NewELP), thus we apply a stabilization technique in order to reduce the oscillatory behaviour of the dual variables. Among the different methods proposed in literature to stabilize a column-generation procedure, we recall [21, 39, 40, 44, 39, 48]. We start from the approach presented in [48, 39], since it does not require any change to the primal or dual problem, and it just needs a parameter Δ to be tuned and the availability of an easily computable lower bound, e.g. like (5.69), for our problem. The procedure solves the pricing problem using as coefficients a convex combination between the values ρ_{RM} of the current optimal dual variables and a stability center $\bar{\rho}$:

$$\rho_{ST} = \Delta\, \rho_{RM} + (1 - \Delta)\bar{\rho} \tag{5.74}$$

Thus, the pricing problem solved within the stabilization method is the following (sPP):

$$(sPP)\ \min c^q - \sum_{k=1}^{|K|} \pi_{ST}^k \chi^k - \sum_{k=1}^{|K|} \sum_{p:g_p^k=2} \mu_{ST}{}_p^k \zeta_p \chi^k - \nu_{ST} \tag{5.75}$$

$$s.t.\ \zeta_p \le 1 - \chi^k \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P : g_p^k = 0 \tag{5.76}$$

$$\zeta_p \ge \chi^k \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P : g_p^k = 1 \tag{5.77}$$

$$\chi^k, \zeta_p \in \{0,1\} \qquad \forall\, k \in \{1,\ldots,|K|\}, p \in P \tag{5.78}$$

The procedure we propose to solve (NewELP) using stabilized column generation is described in Algorithm 5.15. Notice that it requires an exact method for the pricing problem. The first steps of the algorithm (Steps 2 and 3) initialize the parameters,

Algorithm 5.15 Stabilized algorithm - exactly solved pricing problem

```
 1: procedure STABILIZATION
 2:     set $\Delta = \Delta_0$, $0 < \Delta_0 < 1$, $\tau \ge 0$ and $\epsilon > 0$,
 3:     initialize $\bar\rho = \rho_0$, $LB(\bar\pi, \bar\mu) = -\infty$, initial set of variables $\bar Q \subseteq Q$
 4:     repeat
 5:         solve RMP $\to$ get $(\rho_{RM}, z_{RM})$,
 6:         set $\rho_{ST} = \Delta\, \rho_{RM} + (1-\Delta)\bar\rho$ ,
 7:         solve (sPP) $\to$ get $q^{sPP}$, $z_{sPP}$,
 8:         if $z_{sPP} < 0$ and $v(q^{sPP}) < 0$ then                          ▷ Case A
 9:             add variable $\lambda^{q^{sPP}}$ to $\bar Q$,
10:             compute $LB(\pi_{ST}, \mu_{ST}) = z_{ST} + M\, z_{sPP}$;
11:         else if $z_{sPP} \ge 0$ and $v(q^{sPP}) < 0$ then                   ▷ Case B
12:             add variable $\lambda^{q^{sPP}}$ to $\bar Q$,
13:             compute $d(\rho_{ST})$,
14:         else if $z_{sPP} < 0$ and $v(q^{sPP}) \ge 0$ then                   ▷ Case C
15:             compute $LB(\pi_{ST}, \mu_{ST}) = z_{ST} + M\, z_{sPP}$;
16:         else                                                                ▷ Case D
17:             compute $d(\rho_{ST})$,
18:             set $\Delta = 1$ for the next column-generation iteration
19:         end if
20:         if $LB(\pi_{ST}, \mu_{ST}) > LB(\bar\pi, \bar\mu)$ then
21:             update: $\bar\rho = \rho_{ST}$ and consequentely $LB(\bar\pi, \bar\mu) = LB(\pi_{ST}, \mu_{ST})$
22:         end if
23:         if $[(z_{RM} - LB(\bar\pi, \bar\mu))/LB(\bar\pi, \bar\mu)] \le \tau$ then
24:             $\Delta = 1$
25:         end if
26:     until $z_{RM} - LB(\bar\pi, \bar\mu) < \epsilon$ or $d(\rho_{ST}) = z_{RM}$
27: end procedure
```

that are the stabilization parameter Δ, set to the initial value $\Delta_0 \in]0,1[$, the tolerance used to stop the stabilization $\tau \ge 0$ and the tolerance to stop the column generation $\epsilon \ge 0$. The initial values of the stability center and the lower bound are initialized too. We are then ready to start the column-generation procedure, solving the RMP from which we obtain the dual variables ρ_{RM}. We compute the stabilized dual variables and solve the pricing problem (sPP) (Steps 5-7). Let q^{sPP} be the optimal solution of (sPP). We denote with $v(q^{sPP})$ the value of the reduced cost of q^{sPP} for (NewELP). According to the signs assumed by the optimal solution z_{sPP} and by $v(q^{sPP})$, we face four cases. They have to be treated differently, as a result of Theorem 5.1 which, as we are going to prove, guarantees the convergence of the algorithm to the optimal solution of (NewELP). In particular:

Case A $z_{sPP} < 0$ and $v(q^{sPP}) < 0$: we are able to find a variable to be added to the current RMP and to compute the lower bound (see Step 9),

Case B $z_{sPP} \geq 0$ and $v(q^{sPP}) < 0$: we find a variable to be added to the RMP, but dual variables ρ_{ST} are feasible for the dual of (NewELP), so we cannot compute the lower bound (5.69). We compute instead the dual objective value $d(\rho_{ST})$, (see Step 12),

Case C $z_{sPP} < 0$ and $v(q^{sPP}) \geq 0$: in this case the Q-pair found by solving (sPP) does not have a negative reduced cost for (NewELP), but there can still be other Q-pairs that yield a negative reduced cost. Thus we say that a *misprice* happens. Anyway, having $z_{sPP} < 0$ we can compute the lower bound (see Step 15),

Case D $z_{sPP} \geq 0$ and $v(q^{sPP}) \geq 0$: in this case a misprice can happens too, as the positive optimum value of the (sPP) model does not say anything on the optimality of (NewELP) (see Step 15).

When the difference between the primal objective value and the lower bound is small enough, according to the rule in Step 23, we disable the stabilization by setting $\Delta = 1$. In this way we recover the correctness of the column-generation procedure. Moreover, we expect that also the dual variables are close to the optimal ones, as we stop the stabilization when we are close to the optimal solution. Thus, keeping the stabilization active would not improve much the convergence at this stage.

A possible improvement of this stabilization procedure consists in updating the value of the stabilization parameter Δ, so that it increases its value while we are proceeding towards the optimal solution. In this way, the stability center has a great influence in the first iterations, bounding the values of the dual variables, while after a while we can rely more on the real values ρ_{RM}. We consider as an example the following rule to increase the value of Δ [47]:

$$\text{if} \quad \frac{z_{RM} - LB(\bar{\rho})}{z_{RM}} < 1 - \Delta_0 \quad \text{then} \quad \Delta = 1 - \frac{z_{RM} - LB(\bar{\rho})}{z_{RM}}.$$

Theorem 5.1. *Algorithm 5.15 applied to (NewELP) converges to its optimal solution.*

The proof of Theorem 5.1 requires some additional results. Note that similar results are presented in [39], where they are needed to ensure the convergence of the stabilized column-generation when a misprice happens. In our case, the proof of these results have to be extended, since we use a different definition of the lower bound.

Lemma 5.2. *Let q^{sPP} be the Q-pair found as optimal solution of (sPP), with associated negative value z_{sPP}. If the variable $\lambda^{q^{sPP}}$ does not have a negative reduced cost, then $LB(\pi_{ST}, \mu_{ST}) \geq LB(\bar{\pi}, \bar{\mu}) + \Delta(z_{RM} - LB(\bar{\pi}, \bar{\mu}))$.*

Proof. Define

$$f(\rho, q) = c_q - \sum_{\substack{k=1\\ k:\, g^k \in G^q}}^{|K|} \pi^k - \sum_{\substack{k=1\\ k:\, g^k \in G^q}}^{|K|} \sum_{p: g_p^k = 2, h_p^q = 1} \mu_p^k - \nu$$

representing the value of the objective function of the pricing problem with coefficients ρ and computed in the Q-pair q. Note that this function is linear with respect to π and μ, so that by (5.74) it holds:

$$f(\rho_{ST}, q) = \Delta f(\rho_{RM}, q) + (1 - \Delta) f(\bar{\rho}, q).$$

Moreover, we can rewrite $z_{sPP} = f(\rho_{ST}, q^{sPP})$, $z_{ST} = d(\rho_{ST})$ and $v(q^{sPP}) = f(\rho_{RM}, q^{sPP})$. Then we can write:

$$\begin{aligned} LB(\pi_{ST}, \mu_{ST}) &= z_{ST} + M z_{sPP} = d(\rho_{ST}) + M f(\rho_{ST}, q^{sPP}) = \\ &= \Delta d(\rho_{RM}) + (1-\Delta) d(\bar{\rho}) + \Delta M f(\rho_{RM}, q^{sPP}) + (1-\Delta) M f(\bar{\rho}, q^{sPP}) = \\ &= \Delta (z_{RM} + M f(\rho_{RM}, q^{sPP})) + (1-\Delta)(d(\bar{\rho}) + M f(\bar{\rho}, q^{sPP})) \geq \\ &\geq \Delta z_{RM} + (1-\Delta)(d(\bar{\rho}) + M \min_{q \in Q} f(\bar{\rho}, q)) = \\ &= \Delta z_{RM} + (1-\Delta) LB(\bar{\pi}, \bar{\mu}) \end{aligned}$$

where the inequality holds because $f(\rho_{RM}, q^{sPP}) = v(q^{sPP}) \geq 0$ by hypothesis and $f(\bar{\rho}, q^{sPP}) \geq \min_{q \in Q} f(\bar{\rho}, q)$. □

Proposition 5.3. *When a misprice as in* ***Case C*** *happens, the gap* $z_{RM} - LB(\bar{\pi}, \bar{\mu})$ *is reduced by at least a factor of* $1/(1 - \Delta)$.

Proof. The sequence $\{z_{RM}^k\}_k$, where k indexes the iterations of the stabilized column-generation procedure, is not increasing. Thus, we have

$$\begin{aligned} z_{RM}^{k+1} - LB(\bar{\pi}^{k+1}, \bar{\mu}^{k+1}) &\leq z_{RM}^k - LB(\bar{\pi}^k, \bar{\mu}^k) \leq z_{RM}^k - LB(\pi_{ST}^k, \mu_{ST}^k) \overset{(*)}{\leq} \\ &\leq z_{RM}^k - LB(\bar{\pi}^k, \bar{\mu}^k) - \Delta(z_{RM}^k - LB(\bar{\pi}^k, \bar{\mu}^k)) = (1-\Delta)(z_{RM}^k - LB(\bar{\pi}^k, \bar{\mu}^k)) \end{aligned}$$

where inequality (*) holds by Lemma 5.2. Hence

$$\frac{z_{RM}^k - LB(\bar{\pi}^k, \bar{\mu}^k)}{z_{RM}^{k+1} - LB(\bar{\pi}^{k+1}, \bar{\mu}^{k+1})} \geq \frac{1}{1 - \Delta}$$

□

We prove now Theorem 5.1.

Proof of Theorem 5.1. We prove that at each iteration of the column-generation procedure we do not get stuck in a non-optimal solution. Recall that, after solving the pricing problem (sPP), Algorithm 5.15 distinguishes four cases.

In **Case A**, we are able to add a variable and to compute the lower bound $LB(\pi_{ST}, \mu_{ST})$, that can possibly lead to update the stability center. Thus, The RMP that we solve at next iteration will be different and will produce different dual variables

ρ_{RM}. If the lower bound is increased, the (sPP) changes also thanks to the change of the stability center.

In **Case B**, although we cannot compute the lower bound, we add a variable to the current formulation of the RMP, obtaining different values ρ_{RM} for the next iteration.

The proof of convergence for **Case C** is based on Lemma 5.2 and Proposition 5.3. These results prove that, every time a misprice of this kind takes place, the lower bound increases, so that according to the stabilization algorithm we need to update the stability center. This means that the following iteration of the column-generation procedure will solve (sPP) with different coefficients ρ_{ST}, even if the dual variables ρ_{RM} are the same. Moreover, according to Proposition 5.3, in these cases the lower bound increases by a factor big enough to ensure the convergence of the lower bound to the optimal solution.

In **Case D**, we are actually suspending the stabilized procedure and solving the pricing problem (NewPP) instead of (sPP). Thus, we either prove that no other variable is needed to solve (NewELP) or we find a new variable to be added to the RMP. In the first case the column-generation procedure stops, in the second case we reenable the stabilization procedure, starting from different dual variables ρ_{RM}. □

In [40] another way of applying this stabilization technique is proposed. It exploits the advantage of using a lagrangian dual problem, which guarantees that the stability center always remains dual feasible. In this approach, considerations on the relative positions between dual variables ρ_{RM}, stability center $\bar{\rho}$ and stabilized variables ρ_{ST} allow to discard **Case C**, in which $z_{PP} \leq 0$, while $v(q^{sPP}) > 0$, meaning that the constraint found cannot cut off the current dual solution in the dual space. Moreover, the properties of lagrangian dual guarantee that, adding in any case the variable resulting from the pricing problem, at each iteration the difference between ρ_{ST} and ρ_{RM} decreases, so that **Case D** is not critical and convergence is ensured. Our procedure, instead, is able to use even non dual-feasible stability centers, since we prevent adding variabled with positive reduced cost. In this way, we directly consider as lower bound the value obtained from a Lagrangian relaxation of the primal problem (NewELP). Moreover, the approach we propose for **Case D** has a double advantage: together with avoiding to stop at a non-optimal solution, it provides an easy check to know if we are at the optimum, as we are assuming that the scenario in which the reduced costs associated to both ρ_{ST} and ρ_{RM} are positive happens only when we are close to the optimum.

5.4.3 Stabilized column generation and heuristic pricing

The proof of correcteness of Algorithm 5.15 assumes that the value of the optimal solution of (sPP) is provided by an exact pricing procedure. In fact, if we apply heuristic pricing to find variables with negative reduced cost we are not able to compute the lower bound according to (5.69). Notice, however, that the use of heuristics can always be allowed to find and add variables with negative reduced cost, since we can still recover the convergence of the stabilized algorithm by guaranteeing that we solve at least once (sPP) during the column-generation procedure.

The heuristic for fixed haplotypes and the local search-based heuristic (see respectively Sections 5.2.3 and 5.2.4) do not provide the dual information needed by our stabilization, hence we cannot compute the lower bound, neither we can update the stability center. In the application of the Early-terminated Smart Enumeration, instead, some dual information is available, related to the solution of (sPP) on a subset of the solution space. However, this cannot be directly used for lower bound and stability center updating, as defined by Step 21 of Algorithm 5.15. This may prevent updating the stability center as long as the method provides negative reduced cost variables, which is likely to happen at the early stages of column generation, when, in fact, stabilization would be beneficial. We thus define a new criterion to suitably update the stability center when Early-terminated Smart Enumeration is applied. The modified stabilization procedure is shown in Algorithm 5.16.

Algorithm 5.16 Stabilized algorithm - heuristic pricing

```
1: procedure STABILIZATION
2:     set $\Delta = \Delta_0$, $0 < \Delta_0 < 1$, $\tau \geq 0$ and $\epsilon > 0$,
3:     initialize $\bar{\rho} = \rho_0$, $LB(\bar{\pi}, \bar{\mu}) = -\infty$, initial set of variables $\bar{Q} \subseteq Q$
4:     repeat
5:         solve RMP $\rightarrow$ get $(\rho_{RM}, z_{RM})$,
6:         set $\rho_{ST} = \Delta\, \rho_{RM} + (1 - \Delta)\bar{\rho}$ ,
7:         solve (sPP) $\rightarrow$ get $q^{sPP}$, $\tilde{z}_{sPP}$,
8:         if $\tilde{z}_{sPP}$ is optimal then
9:             apply Algorithm 5.15
10:        else
11:            if $v(q^{sPP}) \leq 0$ then
12:                add variable $\lambda^{q^{sPP}}$ to $\bar{Q}$
13:            end if
14:            if $\tilde{z}_{sPP} \leq 0$ then
15:                update: $\bar{\rho} = \rho_{ST}$
16:            end if
17:        end if
18:        if $[(z_{RM} - LB(\bar{\pi}, \bar{\mu}))/LB(\bar{\pi}, \bar{\mu})] \leq \tau$ then
19:            $\Delta = 1$
20:        end if
21:    until $z_{RM} - LB(\bar{\pi}, \bar{\mu}) < \epsilon$ or $d(\rho_{ST}) = z_{RM}$
22: end procedure
```

Note that the stabilized column-generation procedure is initialized as in Algorithm 5.15. The first difference arises at Step 7, where instead of the optimal solution to the pricing problem we get non-optimal $\tilde{z}_{sPP}$, associated to the Q-pair q^{sPP}. Remember that the Early-terminated Smart Enumeration stops as soon as it finds a Q-pair q such that $v(q) < 0$, with no guarantee on the sign of $\tilde{z}_{sPP}$. Hence, we cannot compute the current lower bound, and in particular we are not sure that the current stabilized variables improve the value of $LB(\bar{\pi}, \bar{\mu})$. As a consequence, Step 21 of Algorithm 5.15 cannot be performed and we decide instead to update the stability center if $\tilde{z}_{sPP}$ is also negative. The procedure's termination conditions are the same proposed in Algorithm 5.15.

CHAPTER 6

Branch-and price frameworks for the Integer Program

In Chapter 5 we presented different solution approaches for (ELP), or equivalently (NewELP). In this chapter we embed them in a branch-and-price algorithm in order to find the optimal solution of the integer formulation denoted as (NewEIP). We build a sequence of branchings that iteratively partition the feasible region into subsets, in which specific constraints progressively restrict the research of integer solutions. For each of these subsets we then solve the associated linear relaxation. The standard branching procedure for binary variables consists in finding a variable that assumes a fractional value in the solution of the linear relaxation at the current node and forcing it to take value either 0 or 1. For (NewEIP), fixing one variable to 0 does not provide a significant information on the optimal solution, while a variable with value 1 has a great impact on the value taken by many other variables. This implies that the standard branching procedure is likely to generate a highly unbalanced branching tree. Therefore in Section 6.1 we investigate a family of linear inequalities that provides a partition of the set of feasible integer solutions used for implementing an ad-hoc branching rule for (NewEIP), based on the relations between (NewEIP) constraints and the ones of set-partitioning models.

In Section 6.2, the same relations are used to devise a more general framework for finding an optimal solution to the Haplotype Inference by Pure Parsimony problem by solving a relaxation of (NewEIP). The same branching rule proposed for (NewEIP) is applied in a more general context, that takes care of the procedures needed to recover the optimal solution for the HIPP problem.

6.1 Branching rule for the Haplotype Inference by Pure Parsimony problem

We recall that (NewEIP) is defined as follows:

$$\text{(NewEIP)}\ \min \sum_{q\in Q} c_q\lambda^q \qquad + (m - |K|) \tag{6.1}$$

$$s.t. \quad \sum_{q\in Q:\, g^k\in G^q} \lambda^q = 2 \qquad \forall\, k \in \{1,\dots,|K|\} \tag{6.2}$$

$$\sum_{\substack{q\in Q:\, g^k\in G^q\\ h^q_p=1}} \lambda^q = 1 \qquad \forall\, k \in \{1,\dots |K|\},\ p\in P:\ g^k_p = 2 \tag{6.3}$$

$$\sum_{q\in Q} \lambda^q \le M \tag{6.4}$$

$$\lambda^q \in \{0,1\} \qquad \forall\, q\in Q \tag{6.5}$$

It is worth noticing that constraints (6.3) are similar to the constraints that are found in set-partitioning formulations, as seen in Section 2.5.3. Thus, we look for a branching rule that takes into account this set-partitioning structure, using observations that recall Ryan and Foster's idea [45], in order to obtain a more balanced tree.

6.1.1 The branching rule

In any integer solution of the HIPP problem, any pair of constraints (6.3) that are related to the same position p and different genotypes g^s and g^t will be satisfied either by the same variable or by different ones. This means that we can either find a variable that takes value 1 and whose associate subset of genotypes contains both genotypes, or such variable does not exist. This condition can be written as follows:

$$\sum_{\substack{q:\, g^s, g^t\in G^q\\ h^q_p=1}} \lambda^q \in \{0,1\} \quad \forall\, s,t \in \{1,\dots,|K|\}, \forall\, p\in P:\ g^s_p = g^t_p = 2 \tag{6.6}$$

Observation 6.1. *Due to the assumptions on set K, each genotype in K has at least one heterozygous position and, to ensure that g^s and g^t are compatible but not equal, there must be a position p in which a genotype is heterozygous and the other is homozygous.*

For ease of notation, from now on, we refer to a pair of compatible and distinct genotypes g^s and g^t and a position p such that $g^s_p = 2$.

We can see how in a feasible solution of (NewEIP) there can be at most one variable that takes value 1 and whose associate subset of genotypes contains a chosen pair of genotypes, as it is shown by Lemma 6.1.

Lemma 6.1. *Let g^s, g^t be two compatible genotypes. Then for any feasible solution of (NewEIP) it holds*

$$\sum_{q:\, g^s,\, g^t\in G^q} \lambda^q \le 1$$

Proof. We rewrite the sum dividing it in the following two sums:

$$\sum_{q:\, g^s,\, g^t\in G^q} \lambda^q = \sum_{\substack{q:\, g^s,\, g^t\in G^q\\ h^q_p=0}} \lambda^q + \sum_{\substack{q:\, g^s,\, g^t\in G^q\\ h^q_p=1}} \lambda^q \tag{6.7}$$

As g^s, g^t are compatible and they do not coincide, w.l.o.g. there exists a position p such that $g^s_p = 2$ and $g^t_p = \beta$, with β either equal to 0 or 1. Thus, one of the two addends in (6.7) must be equal to 0, as the set $\{q : g^s, g^t \in G^q, h_p^q = 1 - \beta\}$ is empty due to compatibility reasons. It follows that

$$\sum_{q:\, g^s, g^t \in G^q} \lambda^q = \sum_{\substack{q:\, g^s, g^t \in G^q \\ h_p^q = \beta}} \lambda^q \leq \sum_{\substack{q:\, g^s \in G^q \\ h_p^q = \beta}} \lambda^q = 1$$

□

To guarantee the integrality of the final solution we need to enlarge the set of conditions to be investigated: from constraints (6.2) and (6.3) we can derive another family of set-partitioning redundant inequalities for (NewEIP):

$$\sum_{\substack{q:\, g^s \in G^q \\ h_p^q = 0}} \lambda^q = 1.$$

Then, considering the statement of Lemma 6.1 and either a constraint in (6.2) or (6.3) we obtain the following conditions, that implies that either the two genotypes belong to the same Q-pair or not:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h_p^q = 1}} \lambda^q \in \{0, 1\} \qquad \forall\, s, t \in \{1, \dots, |K|\},\, p \in P : g_p^s = 2 \tag{6.8}$$

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h_p^q = 0}} \lambda^q \in \{0, 1\} \qquad \forall\, s, t \in \{1, \dots, |K|\},\, p \in P : g_p^s = 2 \tag{6.9}$$

The branching strategy we propose is then based on identifying any violated condition (6.8) or (6.9), and forcing the associated sum to take either value 0 or 1. We distinguish the branching strategy according to whether we also have g_p^t equal to 2 or if $g_p^t \in \{0, 1\}$. In fact, if $g_p^t = \beta$ with $\beta \in \{0, 1\}$, for compatibility reasons only one of the sets $\{q \in Q : g^s, g^t \in G^q \wedge h_p^q = 1\}$ and $\{q \in Q : g^s, g^t \in G^q \wedge h_p^q = 0\}$ is not empty, thus it makes sense to check only one between conditions (6.8) and (6.9). In this case, the branching at the node produces two child nodes as follows:

DIFFER g^s and g^t do not belong to the same Q-pair,

SAME g^s and g^t belong to the same Q-pair.

If $g_p^t = 2$, both genotypes can be contained in a Q-pair q having $h_p^q = 0$ or $h_p^q = 1$, so that a single child node is not enough to describe correctly the case in which both genotypes are resolved by a common haplotype. Thus, we need to generate three different child nodes to properly give a partition of the feasible region:

BI-DIFFER g^s and g^t do not belong to the same Q-pair,

SAME0 g^s and g^t belong to the same Q-pair and the associated haplotype has value 0 in position p,

SAME1 g^s and g^t belong to the same Q-pair and the associated haplotype has value 1 in position p.

Note that, by Lemma 6.1, knowing that g^s and g^t have a common haplotype h' such that, in position p, $h'_p = 0$ (resp. 1), we can also say that no other common haplotype is possible. Thus, in the obtained branch, we can also fix that no haplotype h'' such that $h''_p = 1$ (resp. 0) resolves both g^s and g^t.

Algorithm 6.17 Branching procedure for (NewEIP)

```
1: procedure BRANCHING PROCEDURE
2:     choose $g^s$, $g^t$ and $p$ such that at least one among (6.8) and (6.9) is violated,
3:     branch:
4:     if $g^t_p \in \{0,1\}$ then
5:         create two child nodes of type DIFFER and SAME,
6:     else
7:         create three child nodes of type BI-DIFFER, SAME0 and SAME1,
8:     end if
9: end procedure
```

The above observations are summarized in Algorithm 6.17, which takes a node of the branching tree and the solultion of the related (NewELP) as input and create two or three child nodes. The branching constraints that characterize these nodes are chosen looking for pairs of genotypes g^s, g^t and a position p that violate at least one condition between (6.8) and (6.9). In particular, each type of child node is obtained adding the following constraints:

$$\begin{array}{ll}
\text{DIFFER} & \displaystyle\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = \beta}} \lambda^q \leq 0 \\
\text{SAME} & \displaystyle\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = \beta}} \lambda^q \geq 1 \\
\text{BI-DIFFER} & \displaystyle\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \leq 0 \ \wedge \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \leq 0 \\
\text{SAME0} & \displaystyle\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \geq 1 \ \wedge \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \leq 0 \\
\text{SAME1} & \displaystyle\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \leq 0 \ \wedge \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \geq 1
\end{array}$$

6.1.2 Proof of correctness

We now prove the correctness of the proposed procedure. First, in Theorems 6.1 and 6.2 we prove that the integrality of the final solution is ensured, by considering separately the variables λ^q having associated subset of genotypes G^q with cardinality greater than 1 and variables that involve only one genotype. Finally, in Theorem 6.3 we use these results to prove the correctness of the branching procedure given in Algorithm 6.17.

Theorem 6.1. *Suppose that for every pair g^s, g^t of compatible genotypes and for every position p such that $g^s_p = 2$ conditions (6.8) and (6.9) hold. Then $\lambda^q \in \{0,1\}$ for all $q = (h^q, G^q) \in Q$ such that $|G^q| > 1$.*

Proof. Fix a genotype g^s, a position p such that $g^s_p = 2$ and another genotype g^t compatible with g^s. Recall that such a position p can always be found as $g^s \in K$, according to observation 6.1. We consider the constraint (6.3), that refer to haplotypes which take value 1 at the chosen position p, associated to genotype g^s and we split it into two addends:

$$1 = \sum_{\substack{q:\, g^s \in G^q \\ h^q_p = 1}} \lambda^q = \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q + \sum_{\substack{q:\, g^s \in G^q, g^t \notin G^q \\ h^q_p = 1}} \lambda^q \tag{6.10}$$

By hypothesis, the first addend can be either equal to 1 (**Case 1**) or 0 (**Case 2**) by hypothesis, forcing the second one to be respectively equal to 0 or 1.

Case 1: All variables involved in the second addend are equal to 0, and then integer. We can focus on the variables appearing in the first addend.
Suppose, by absurd, that this sum involves at least one fractional variable $\lambda^{\bar{q}}$. Thus, as the whole sum takes value one, there must be at least another fractional variable $\lambda^{\tilde{q}}$. That is, we found Q-pairs $\bar{q}$ and $\tilde{q}$ such that $g^s, g^t \in G^{\bar{q}}$ and $g^s, g^t \in G^{\tilde{q}}$, $h^{\bar{q}}_p = h^{\tilde{q}}_p = 1$ and $\lambda^{\bar{q}}$, $\lambda^{\tilde{q}} \in]0,1[$.

There are two subcases:
1.A) $G^{\bar{q}} \neq G^{\tilde{q}}$, then (w.l.o.g) $|G^{\bar{q}}| > 2$ as g^s and g^t are already included in $G^{\bar{q}}$.
Then there exists another genotype $g^f \in G^{\bar{q}}$ such that $g^f \notin G^{\tilde{q}}$ and it holds by hypothesis:

$$\sum_{\substack{q:\, g^s, g^f \in G^q \\ h^q_p = 1}} \lambda^q = \begin{cases} 0 & (*) \\ 1 & (**) \end{cases}$$

If $(*)$ holds, then $\lambda^{\bar{q}} = 0$, that is absurd. So $(**)$ holds and, rewriting (6.10) according to g^s, g^f and position p, we have as consequence that

$$\sum_{\substack{q:\, g^s \in G^q, g^f \notin G^q \\ h^q_p = 1}} = 0,$$

that is even absurd as $\tilde{q} \in \{q \in Q : g^s \in G^q, g^f \notin G^q, h^q_p = 1\}$ and $\lambda^{\tilde{q}} > 0$.
1.B) $G^{\bar{q}} = G^{\tilde{q}}$.
We have $h^{\bar{q}} \neq h^{\tilde{q}}$, in particular w.l.o.g. there exists $\hat{p}$ such that $h^{\bar{q}}_{\hat{p}} = 1$ and $h^{\tilde{q}}_{\hat{p}} = 0$. It follows that $g^s_{\hat{p}} = g^t_{\hat{p}} = 2$. By hypothesis, we know that:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_{\hat{p}} = 1}} \lambda^q = \begin{cases} 0 & (\star) \\ 1 & (\star\star) \end{cases}$$

The case $(\star)$ is absurd, because $\bar{q} \in \{q \in Q : g^s, g^t \in G^q, h^q_{\hat{p}} = 1\}$ and $\lambda^{\bar{q}} > 0$. Then $(\star\star)$ holds.
Remember that $\bar{q}$ belongs to $\{q \in Q : g^s, g^t \in G^q, h^q_{\hat{p}} = 1\}$ and $\lambda^{\bar{q}}$ is fractional, while $\tilde{q}$ does not. This implies that there must be at least another $\hat{q}$ with associated variable fractional such that $g^s, g^t \in G^{\hat{q}}$, $h^{\hat{q}}_{\hat{p}} = 1$. There are again two cases:

1.B.a) if $G^{\hat{q}} \neq G^{\bar{q}}$, then we can apply case **A** to $\lambda^{\bar{q}}, \lambda^{\hat{q}} \in]0,1[$, $g^s, g^t \in G^{\bar{q}}$ and $g^s, g^t \in G^{\hat{q}}$, $g^s_{\hat{p}} = 2$, $h^{\bar{q}}_{\hat{p}} = h^{\hat{p}}_{\hat{p}} = 1$, obtaining that it is absurd to consider $\lambda^{\bar{q}}, \lambda^{\hat{q}}$ not integer, so that also $\lambda^{\tilde{q}}$ must be integer.

1.B.b) if $G^{\hat{q}} = G^{\bar{q}}$ $(= G^{\tilde{q}})$, we remember that $h^{\bar{q}}_{\hat{p}} = h^{\hat{q}}_{\hat{p}} = 1$, while $h^{\tilde{q}}_{\hat{p}} = 0$. Then it holds:

$$1 \geq \sum_{q:\, g^s,\, g^t \in G^q} \lambda^q = \sum_{\substack{q:\, g^s,\, g^t \in G^q \\ h^q_{\hat{p}}=1}} \lambda^q + \sum_{\substack{q:\, g^s,\, g^t \in G^q \\ h^q_{\hat{p}}=0}} \lambda^q$$

where the inequality holds because of Lemma 6.1.The first addend of the sum on the right is integer by hypothesis and in particular, as $\bar{q}$ and $\hat{q}$ belong to $\{q \in Q : g^s,\, g^t \in G^q, h^q_{\hat{p}} = 1\}$, it must be equal to 1, thus the second addend must be equal to 0. But $\tilde{q}$ belongs to $\{q \in Q : g^s,\, g^t \in G^q,\, h^q_{\hat{p}} = 0\}$ and $\lambda^{\tilde{q}} > 0$, so we have an absurd.

Thus, all variables appearing in the first addend in (6.10) are integer.

Case 2: All variables for which $q \in \{q \in Q : g^s,\, g^t \in G^q, h^q_p = 1\}$ are equal to 0. We can rewrite the second addend of (6.10) in the following way:

$$\sum_{\substack{q:\, g^s \in G^q, g^t \notin G^q \\ h^q_p=1}} \lambda^q = \sum_{\substack{q:\, g^s \in G^q, g^t \notin G^q \\ |G^q|>1,\, h^q_p=1}} \lambda^q + \sum_{\substack{q:\, G^q=\{g^s\} \\ h^q_p=1}} \lambda^q. \tag{6.11}$$

For each Q-pair $q^\star$ whose associated variable is present in the first addend in (6.11), there exists another genotype $g^r \in G^q$ such that $g^s_p = 2$ and $g^r_p \neq 0$ (for compatibility reasons). Thus, we can rewrite (6.10) in the following way:

$$1 = \sum_{\substack{q:\, g^s \in G^q \\ h^q_p=1}} \lambda^q = \sum_{\substack{q:\, g^s, g^r \in G^q \\ h^q_p=1}} \lambda^q + \sum_{\substack{q:\, g^s \in G^q, g^r \notin G^q \\ h^q_p=1}} \lambda^q$$

so that $\lambda^{q^\star}$ belongs to the first term, that we already proved in Case 1 contains only integer variables. Thus, $\lambda^{q^\star}$ is integer.

As far as now we only considered variables such that the associated haplotypes assume value 1 at position p. We observe that

$$\sum_{\substack{q:\, g^s \in G^q \\ h^q_p=0}} \lambda^q = 1 \tag{6.12}$$

are implicit constraints for (NewEIP). We can thus repeat the same proof starting from (6.12), and complete the proof, as we deduce that any variable λ^q of (NewEIP) such that $|G^q| > 1$ is involved in at least one constraint of type (6.3) or (6.12). □

Theorem 6.2. *Suppose that for every pair g^s, g^t of compatible genotypes and for every position p such that $g^s_p = 2$ conditions (6.8) and (6.9) hold. Then if the solution is still fractional, there is an equivalent integer solution.*

Proof. As Theorem 6.1 holds, if there exists a Q-pair $\bar{q} = (h^{\bar{q}}, G^{\bar{q}})$ such that $\lambda^{\bar{q}} \in]0, 1[$, then $G^{\bar{q}}$ must contain only one genotype $\bar{g}$. Moreover, as soon as $\bar{g}$ appear in another Q-pair $q = (h^q, G^q)$ such that $|G^q| > 1$, we have $\lambda^q = 1$ and due to the formulation's constraints (6.2) and (6.3) we have that $h^{\bar{q}}$ must be equal to the complementary of h^q. This implies that, in order to satisfy constraints (6.3), there must exists at least another fractional variable with associated haplotype $\bar{h}$. In order to be fractional, by Theorem 6.1 this new Q-pair should be equal to $(h^{\bar{q}}, \{\bar{g}\})$. That is, exactly equal to $\bar{q}$, thus the absurd. It follows that $\bar{g}$ cannot be included in any Q-pair whose associated subset of genotypes contains also other genotypes.

Thus, $\bar{g}$ is included in a collection of k Q-pairs $q^1 = (h^1, \{\bar{g}\}), \dots, q^k = (h^k, \{\bar{g}\})$. But it is then possible to choose any two Q-pairs with associated variable equal to 1 that resolve $\bar{g}$. □

The proof of the correctness of the proposed branching strategy is given in Theorem 6.3.

Theorem 6.3. *The branching strategy for (NewEIP) proposed in Algorithm 6.17 is correct.*

Proof. To be correct, this branching scheme should:

- cut off the current solution,
- provide a partition of the feasible region,
- guarantee integrality of the solution.

As for the first requirement, it is trivial to show that imposing one of the preceeding branchings, according to the values of g^s_p and g^t_p, we cut off the current solution that violates condition (6.8) or (6.9).

With regards to the partitioning of the feasible region, consider a feasible solution $\bar{Q} = \{(h^q, G^q) \in Q : \lambda^q = 1\}$ of (EIP). We need to show that this solution is not cut off by any branching constraint and that it belongs to exactly one branch. Consider the branching constraint defined by genotypes g^s and g^t and position p. We have two possibilities:

- the two genotypes do not belong to a same Q-pair in $\bar{Q}$. If moreover $g^s_p \neq 2$ or $g^t_p \neq 2$, then the solution satisfies the branching constraint of type DIFFER. Otherwise, the solution satisfies the branching constraint of type BI-DIFFER.
- there exists one $q^\star \in \bar{Q}$ such that $g^s \in G^{q^\star}$, $g^t \in G^{q^\star}$. Note that by Lemma 6.1 it is also unique. If $g^s_p \neq 2$ or $g^t_p \neq 2$, the solution satisfies the branching constraint of type SAME, otherwise it satisfies the branching constraints SAME0 or SAME1.

In any case, the branching constraints are not violated by the feasible integer solution. Moreover, each feasible solution belongs exactly to one branch, as constraints describing two different branches of the same node do not allow common integer solutions.

Finally, the integrality of the solution is ensured by Theorems 6.1 and 6.2, in which we proved that when no suitable branching constraint is found either the solution obtained is integer or there is an equivalent integer solution, easy to be recovered. □

6.1.3 Branching priorities

A branching priority is needed to choose, at each node, which pair of genotypes g^s, g^t and which position p such that $g^s_p = 2$ we should select to apply the next branching among all the triplets (s, t, p) for which at least one among (6.8) and (6.9) is violated. We present here two possible alternatives.

Priority A): choose the pair g^s, g^t and the position p such that the sum

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = \beta}} \lambda^q \qquad \text{for } \beta \in \{0, 1\}$$

is the most fractional:

$$(g^s, g^t, p) = \operatorname{argmax}_{(g^s, g^t, p)} \left\{ \max_{\beta \in \{0,1\}} \{ \min\{ \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = \beta}} \lambda^q, 1 - \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = \beta}} \lambda^q \} \} \right\};$$

Priority B): select the the pair g^s, g^t and the position p that involve the maximum number of variables:

$$\operatorname{argmax}_{(g^s, g^t, p)} \left\{ \max_{\beta \in \{0,1\}} \{ \max\{ |\{ \lambda^q \in]0, 1[:\ g^s, g^t \in G^q, h^q_p = \beta \}| \} \} \right\}$$

The first priority scheme is rather intuitive, and often used in branch-and-bound contexts. The second one can be useful to involve a number as large as possible of variables in the branching, so that any branch that fixes (6.8) or (6.9) to, e.g., 0 is possibly able to set to 0 a larger number of variables.

6.2 Solution method based on a set-covering-like relaxation

We note that constraints (6.2) of (NewEIP) are similar to set-partitioning constraints. It is well known that, whenever we can substitute a set-partitioning (2.44) formulation with a set-covering one (2.45), this second one is to be preferred, in a column-generation context, as its linear relaxation is numerically more stable and it is possible to construct an integer solution given the linear programming relaxation [5]. Taking advantage of this observation, we substitute (6.2) with constraints that recall a set-covering structure. In this section, then, we consider the linear program, that we denote with (EIPR), obtained by substituting in the (NewEIP) formulation constraints (6.2) with the corresponding greater-or-equal constraints:

$$\text{(EIPR)} \ \min \sum_{q \in Q} c_q \lambda^q \qquad + (m - |K|) \tag{6.13}$$

$$s.t. \sum_{q \in Q:\, g^k \in G^q} \lambda^q \geq 2 \qquad \forall\, k \in \{1, \ldots, |K|\} \tag{6.14}$$

$$\sum_{\substack{q \in Q:\, g^k \in G^q \\ h^q_p = 1}} \lambda^q = 1 \qquad \forall\, k \in \{1, \ldots |K|\},\ p \in P:\ g^k_p = 2 \tag{6.15}$$

$$\sum_{qinQ} \lambda^q \leq M \tag{6.16}$$

$$\lambda^q \in \{0,1\} \qquad \forall\, q \in Q \tag{6.17}$$

We have already noticed in Section 5.2.6 (see Table 5.1) that (EIPR) is a relaxation of (NewEIP), in the sense that any genotype g is allowed to appear in more than two Q-pairs. If this happens, we say that this genotype is *over-resolved.*

6.2.1 The drawbacks of the set-covering-like relaxation

Note that, in our case, the structure of this relaxed formulation is more complicated and we cannot entirely rely on the relation between set-partitioning and set-covering constraints. In fact, the solution obtained from (EIPR) can be infeasible for the HIPP problem. The main issues that arise from using (EIPR) formulation are presented in the following observations.

Observation 6.2. *The model (EIPR) does not preserve the feasibility and the optimality of the solution for the HIPP problem.*

Example 6.1. *Consider the set of genotypes G made of*

$$g^1 = 2012000,\ g^2 = 0012202,\ g^3 = 2012202.$$

One optimal solution of (NewEIP), with objective value equal to 4, can be the one that assigns value 1 *to the variables associated to the following Q-pairs:*

$$q^1 = (1010000, \{g^1\}),\ q^2 = (0011000, \{g^1, g^2\}),$$
$$q^3 = (0010101,\ \{g^2, g^3\}),\ q^4 = (1011000, \{g^3\}).$$

Instead, considering the relaxed constraints we could get the solution, with optimal value equal to 3, that set to value 1 the variables associated to Q-pairs

$$q^1 = (1010000, \{g^1, g^2\}),\ q^2 = (0011000, \{g^1, g^2, g^3\}),\ q^3 = (0010101,\ \{g^2, g^3\}).$$

However, in this case genotype g^3 is not resolved using only two haplotypes out of the three considered, so that the solution is not optimal and neither feasible for our original problem.

Observation 6.3. *The proof of Theorem 6.1 that ensures integrality of variables λ^q with $|G^q| > 1$ for (NewEIP) does not hold for (EIPR).*

In particular, Lemma 6.1 is not valid for (EIPR). This lemma states that given a pair of genotypes g^s and g^t, it holds

$$\sum_{q:\, g^s, g^t \in G^q} \lambda^q \leq 1$$

and it is crucial to prove case **1.B** of Theorem 6.1, which proves that there cannot be two fractional variables associated to the same subset of genotypes.

We note, however, that in general there is a simple condition, together with the branching rule of Algorithm 6.17, that guarantees integrality of some variables even if (EIPR) is considered.

Observation 6.4. *Given a Q-pair $q = (h^q, G^q)$, if there exist g^s, $g^t \in G^q$ and a position p such that $g^s_p = 2$ and $g^t_p = 1$ or viceversa, then the associated variable λ^q is integer and belongs to $\{0, 1\}$ at the end of the branching procedure presented in Section 6.1.*

Proof. Exploiting the availability of constraints

$$\sum_{\substack{q:\, g^s \in G^q \\ h^q_p = 1}} \lambda^q = 1$$

and the fact that, for compatibility reason, each Q-pair containing both g^s and g^t must have $h^q_p = 1$, we have that Lemma 6.1 still holds, so that the proof of Theorem 6.1 is still valid and each λ^q involved in the sum

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q$$

is integer and belongs to $\{0, 1\}$. □

This condition alone is not enough to ensure integrality of all variables, as there can be variables λ^q such that each pair of genotypes g^s, $g^t \in G^q$ has the following property:

$$(g^s_p, g^t_p) = \{(1,1),\ (2,0),\ (2,2),\ (0,2),\ (0,0)\}, \quad \forall p \in P.$$

Thus, when the branching procedure presented in Algorithm 6.17 cannot identify suitable branching constraints, we cannot assume directly neither that we reached a feasible solution for the HIPP problem, nor that we have an integer solution. This means that we need to further explore the branching subtree rooted at the current node.

6.2.2 Recovering the optimal integer solution

In order to get the optimal solution for the HIPP problem, we embed the branch-and-price procedure proposed in Section 6.1 in a more general branch-and-price framework that exploits the information obtained from solving model (EIPR) and apply a suitable procedure that recovers a feasible solution for (NewEIP) whenever necessary. This procedure can be seen as an extended branching procedure.

Definition 6.1 (Quasi-integer solution). *We say that a solution to (NewEIPR) is* quasi-integer *if for each Q-pair q such that $|G^q| > 1$ we have that λ^q is integer.*

A possible scheme is given by Algorithm 6.18 and detailed in the following. Assume that in a particular node the branching rule is not able to find a constraint to provide a branching (see Step 3 of Algorithm 6.18). The current solution may be not feasible for the HIPP problem and further steps need to be taken. Two scenarios are possible: either the current node has a quasi-integer solution, or not.

In the first case (Step 4), Theorem 6.2 provides us with an integer solution that can be recovered without increasing the objective value. However, this solution can still be infeasible for the HIPP problem and extra tests, that will be seen in Algorithm 6.19, are performed to see if it is possible to recover an equivalent (an thus, optimal) feasible solution.

Algorithm 6.18 Branching procedure for (EIPR)

```
 1: procedure BRANCHING PROCEDURE - (EIPR)
 2:     look for branching constraints,                          ▷ Algorithm 6.17
 3:     if no branching is performed then
 4:         if solution is quasi-integer then
 5:             look for feasible solution for HIPP,             ▷ Algorithm 6.19
 6:             if an equivalent solution is found then
 7:                 prune the node. STOP.
 8:             end if
 9:         end if
10:         for $g$ genotype do
11:             compute $S(g) = \sum_{q\colon g \in G^q} \lambda^q$ ,
12:             if $S(g) > 2$ then
13:                 substitute constraint (6.14) with (6.2) related to $g$,
14:             end if
15:         end for
16:     end if
17: end procedure
```

If the current node does not provide a quasi-integer solution, we modify the formulation as follows (Steps 10-15): we look for each equality constraint of type (6.2) that is violated in the solution and we put it in place of the corresponding inequality constraint of type (6.14). Notice that the proposed branching procedure makes the approach based on (EIPR) converge to the optimal solution. In fact, either it provides an optimal feasible solution for a subtree, or it continues adding constraints of type (6.2) until the formulation (NewEIP) is obtained, which guarantees convergence to the optimal solution.

We go back now to Step 5, that starting from an integer non-feasible solution looks for a feasible solution to HIPP, applying Algorithm 6.19.

Observation 6.5. *Given an integer non feasible solution for HIPP found at a node of the branching tree, it is easy to check whether we can recover a feasible optimal solution.*

Definition 6.2 (Redundant haplotype). *A haplotype h is redundant for a genotype g if for each position $p \in P$ such that $g_p = 2$, $h_p = 0$.*

Algorithm 6.19 Recovering feasibility of the solution

```
1: procedure RECOVER FEASIBILITY(quasi-integer solution)
2:     let $H = \{h \text{ haplotype } : \exists\, q \in Q, h^q = h, \lambda^q = 1\}$
3:     for all genotypes $g \in K$ do
4:         compute $\bar{Q}(g) = \{q : g \in G^q\}$, set  $d(g) = |\bar{Q}(g)|$,
5:         if $d(g) \geq 3$ then
6:             if $\exists\, \tilde{q} \in \bar{Q} : \forall p : g_p = 2$ we have $h_p^{\tilde{q}} = 0$ then
7:                 $G^{\tilde{q}} = G^{\tilde{q}} \backslash \{g\}$,  $\bar{Q}(g) = \bar{Q}(g) \backslash \{\tilde{q}\}$,
8:                 $d(g) = d(g) - 1$,
9:             end if
10:            if $d(g) \geq 3$ and this same haplotype appears in others $Q$-pairs
11:            containing $g$ then
12:                decrease $d$ as long as $d \geq 2$
13:            else if $d(g) \geq 3$ then
14:                for all $q \in \bar{Q}(g)$ do
15:                    compute $\bar{h}^q$ the haplotype complementary to $h^q$,
16:                    if $\bar{h}^q = h^{\hat{q}} \in H$ then
17:                        set $\bar{Q}(g) = \{q, \hat{q}\}$,  $G^{\hat{q}} = G^{\hat{q}} \cup \{g\}$,
18:                        break.
19:                    end if
20:                end for
21:                if no complementary haplotype is in $H$ then
22:                    we cannot find an equivalent feasible solution. STOP.
23:                end if
24:            end if
25:        end if
26:    end for
27: end procedure
```

Note that there is only one redundant haplotype for each genotype.

Starting from an integer non feasible solution for HIPP of value $\bar{z}$, we can check if there exists a feasible solution of equivalent objective value by applying Algorithm 6.19. The algorithm considers each genotype and distinguishes the following cases:

1. a genotype g appears in three or more Q-pairs, but the haplotype associated to some of these is redundant. Thus, we can simply remove g from the Q-pairs associated to the redundant haplotype as long as we keep g in at least two Q-pairs. Note that by the structure of the objective function of (EIPR) we can find more than one Q-pair associated to the redundant haplotype for g only if such a haplotype is fixed,

2. a genotype g appears in three or more Q-pairs, but there is no redundant haplotype associated to any of these Q-pairs. In this case, we can delete genotype g from each Q-pair in the solution but one, and then consider the complementary haplotype. Two subcases appear:

a) if we are able to find such a complementary haplotype among the ones already used in the solution, then we add g to the associated Q-pair,

b) otherwise the integer solution cannot be modified into a feasible one and other procedures are needed.

CHAPTER 7

Implementation

In this chapter we show more in detail how the proposed decomposition approach for solving exactly the Haplotype Inference by Pure Parsimony problem has been implemented. In particular, we first focus on how the linear relaxation is solved, and then we proceed to the implementation of the branching strategy, that requires a special attention on how we manage the branching decisions.

7.1 The linear relaxation of the exponential-size formulation

Solving the linear relaxation of our problem is the starting point to apply the branching procedure and solve the integer problem itself. Here we present the procedure implemented to obtain the exact solution for the (NewEIP) and (EIPR) linear relaxations introduced in Sections 5.4 and 6.2.

7.1.1 Choosing an order for the genotypes

A preliminary aspect to be addressed is the order in which we consider the genotypes in the set G. As we showed in the previous chapters, the heuristic inspired by Clark's algorithm to find a feasible starting solution, the Smart Enumeration used to solve the pricing problem and the procedure used to add multiple variables at each iteration are highly affected by the order we give to the genotypes.

Note that in Sections 5.1 and 5.2.2 we already proposed a possible order on the genotypes to be preferred for finding initial solutions closer to the optimum and to achieve a better performance of the Smart Enumeration procedure. This proposed order is exactly the same, and consists in sorting the genotypes according to the increasing number of heterozygous sites.

We also noticed that an order of the genotypes that possibly improves the choice of the variables to be added at each iteration of the column-generation procedure depends on the values assumed by the dual variables at each iteration and it is therefore difficult to fix. To avoid to reorder many times the genotypes throughout the column-generation procedure, spending running time in sorting, we consider also in this case the ordering according to the increasing number of heterozygous sites.

7.1.2 Overview of the procedure to solve the linear relaxation

The outline of the procedure used to perform the column generation and solve the linear relaxation of models (NewEIP) and (EIPR) is basically the same, provided we consider the RMP associated to each one of them. It is presented in the flow-chart of Figure 7.1. We refer to the notation given in Table 5.2. Starting from the initial solution, we first solve the RMP, built from the variables defined by the initial solution. If the stabilization approach is not enabled, we solve the pricing problem (NewPP) seen in Section 5.4 and add the found variable q^{PP} if it has negative reduced cost. If we do not find a variable with negative reduced cost, or if the difference between the current optimal solution z_{RM} and the lower bound LB is below the tolerance ϵ, we stop the column generation. Otherwise, if the stabilization technique is used, we need to compute the stabilized variables and solve the pricing problem (sPP) also presented in Section 5.4. Note that at this point we propose again the four cases arising from the combinations of signs of the (sPP) optimal solution, z_{sPP}, and the reduced cost associated to Q-pair q^{sPP}, $v(q^{sPP})$ (see Table 5.2). If they are both positive we suspend the stabilization for one iteration. Otherwise, we are able to solve the updated RMP and obtain a different solution. Notice that, as it is shown also in the chart, if we do not find a variable with negative reduced cost but $z_{sPP} < 0$, then it is not necessary to solve again the RMP, as the solution will not change, as well as the dual variables' value. What changes, in this case, is the stability center. Thus, we can directly compute the new stabilized variables and solve again the pricing problem (sPP).

To solve the pricing problem, we solve in sequence:

1. the pricing problem with fixed haplotypes,
2. the local search applied to the best fixed haplotype,
3. the exact pricing problem linearized *or* the exact Smart Enumeration *or* the Early-terminated Smart Enumeration,

and we stop as soon as one of the three procedures gives back a variable with negative reduced cost. We recall that the Early-terminated Smart Enumeration is also able to state if no negative recuded cost variable exists (in this case the early terminating condition is simply not reached). We are not considering an implementation for the maximum flow approach for solving the pricing problem, as preliminary results showed that, for both (NewPP) and (sPP), the coefficient of the objective function does not assume the values required to solve exactly the pricing problem. Thus, we would need to solve a large number of integer linear programs per iteration to be only able to obtain an upper bound on the optimal solution of the pricing problem.

Note that when we apply the stabilization technique, the exact pricing procedures are embedded in the overall column generation Algorithm 5.15, whereas the Early-terminated Smart Enumeration approach requires a different procedure as presented in Algorithm 5.16.

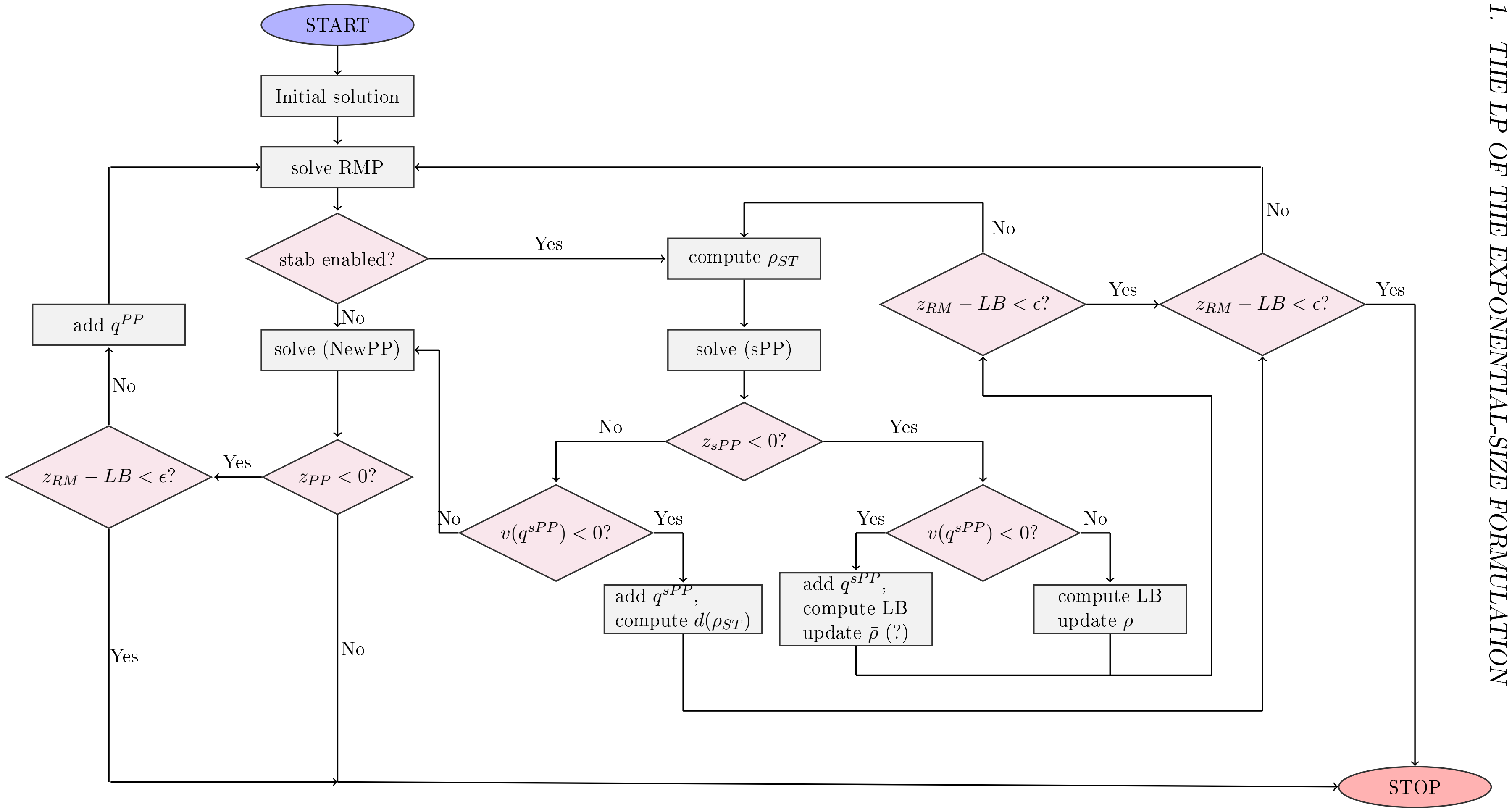


Figure 7.1: Flow chart of the procedure to solve the linear relaxation at the root node of the search tree.

As for the Smart Enumeration (Early-terminated or not), note that once we partitioned the set of possible solutions, the last reduced pricing problem that should be solved is trivial, as there is only one genotype to be considered and it reduces in finding the best haplotype compatible with it. Thus, we solve the pricing problem related to this genotype first, then the remaining reduced pricing problems are solved according to the order given to the genotypes.

7.2 The branch-and-price framework

We show in Figure 7.2 the procedure applied to solve the HIPP problem using the branch-and-price approach, adapted for both models (NewEIP) and (EIPR). Note that the RMP in a particular node is defined by the RMP defined in the parent node to which the node's branching constraints are added. As we are not considering in the RMP all the variables defined by the formulations, it could happen that the node is infeasible. In this case, we apply the Farkas pricing as seen in Section 2.2.3: it consists in solving the pricing problem (NewPP) using the farkas multipliers, provided for example by the last tableau of the simplex method, without considering costs c_q associated to each Q-pair. It returns a variable q^{fPP} and an optimumm value z_{fPP}. If $z_{fPP} < 0$, we add variable q^{fPP} to the formulation at the node. At this stage, we can prune the node only if, after adding all the variables found in this way with negative reduced cost, the node is still infeasible.

Once we have a feasible RMP, its solution process follows the steps seen in Figure 7.1: we need to apply a column-generation procedure to find variables with negative reduced cost. However, the presence of branching constraints at the current node requires extra precautions while solving the pricing problem, to avoid generating variables that are not compliant with the branching constraints. This aspect will be seen in detail later.

Differences on the branching procedures apply to whether we are solving (NewEIP) or (EIPR) formulations. For the first model, the branching step consists in finding a branching constraint according to Algorithm 6.17. When we do not find a branching constraint in this way, we saw in Section 6.1 that we can have an integer or quasi-integer solution, meaning that we can have only fractional variables which involve only one genotype at a time and that it is always possible to recover a feasible and optimal solution for the HIPP problem. Thus the extra check on the feasibility shown in Figure 7.2 is automatically satisfied and simply fix quasi-integer solutions to feasible ones. Instead, if we are solving the (EIPR) program the feasibility check on quasi-integer solution is not trivial, and if it is not satisfied we cannot yet prune the node. The branching step then follows Algorithm 6.18, that may ask for changing the structure of the RMP in the subtree, in order to recover the integer optimal solution.

7.2.1 Embedding branching constraints in the pricing problem

Particular attention has been paid on how the branching decisions are implemented. We recall that these inequalities force the sum of variables involving the same pair of genotypes to be either equal to 0 or to 1. This approach, however, is highly discouraged in a column-generation framework as the addition of constraints to the RMP implies alterations of the pricing problem that can be difficult to handle. In our case, in fact,

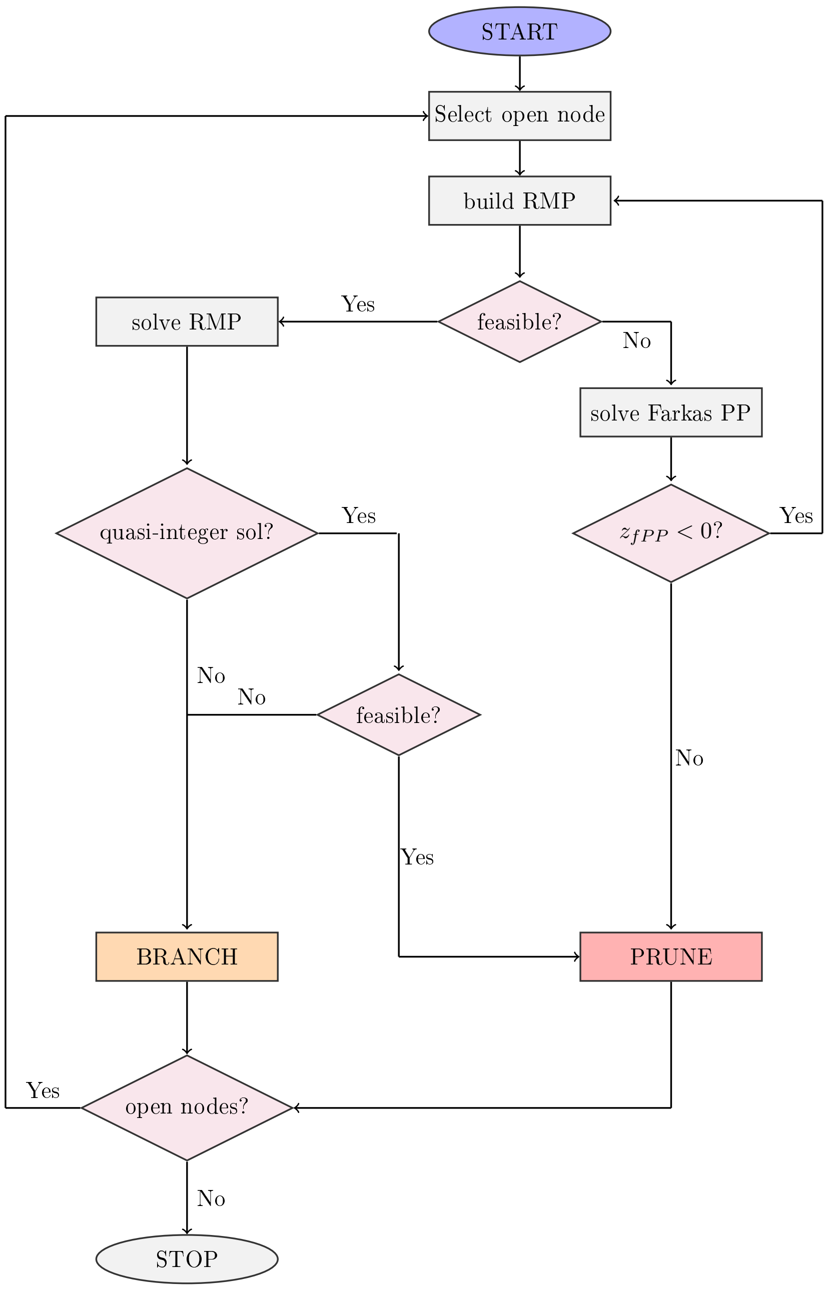


Figure 7.2: Branch-and-price fow chart.

consider $\eta_p^{s,t}$ as the dual variable associated to the new constraint

$$\sum_{\substack{q: g^s, g^t \in G^q \\ h_p^q = \beta}} \lambda^q \geq 1 \ (\leq 0).$$

By adding it to the RMP we are adding a term $\eta_p^{s,t}\chi^s\chi^t\zeta_p$ to the objective function of the pricing problem. This corrupts its structure, that cannot be solved anymore with the proposed approaches.

Thus, the classical alternative consists in including the branching conditions in the pricing problem, preserving its structure as much as possible. To do so, each time we reach a new node we need to consider the set of branching constraints, determined by two genotypes g^s, g^t, a position p and a constraint's type TYPE. We recall that we have these different types of branching constraints, as seen in Section 6.1:

DIFFER when $g_p^s \neq 2$ or $g_p^t \neq 2$ and we do not allow the two genotypes to be resolved by a common haplotype,

SAME when $g_p^s \neq 2$ or $g_p^t \neq 2$ and we want the two genotypes to be resolved by a common haplotype,

BI-DIFFER when $g_p^s = 2$, $g_p^t = 2$ and we do not allow the two genotypes to be resolved by a common haplotype,

SAME0 when $g_p^s = 2$, $g_p^t = 2$ and we want the two genotypes to be resolved by a common haplotype h such that $h_p = 0$,

SAME1 when $g_p^s = 2$, $g_p^t = 2$ and we want the two genotypes to be resolved by a common haplotype h such that $h_p = 1$,

At this point, a first step for the solution of the current node's linear relaxation consists in fixing to 0 those variables already existing in the RMP that are not compliant with the branching constraints for the current node. This means that

- for every branching constraint of type DIFFER or BI-DIFFER, we fix to 0 those variables that contain both g^s and g^t in the subset of genotypes,
- for every other branching constraint, we fix to 0 the variables associated to $q = (h^q, G^q)$ whose subset of genotypes contains exactly one between g^s and g^t. For types SAME0 and SAME1 there is the extra requirement that variables fixed to 0 also need to have respectively $h_p^q = 0$ or $h_p^q = 1$.

In order to solve the pricing problem and avoid generating a Q-pair that violates some branching constraints, we need to translate the information contained in the inequalities showed in Section 6.1 and corresponding to the different types of branching constraints, into constraints to be added to the pricing problem. We recall that, once we identified a suitable branching combination involving g^s, g^t and position p, at least one between g_p^s and g_p^t is equal to 2. We have then the following three cases:

- if $g_p^s = 0$ or $g_p^t = 0$ we have only two child nodes:

- the child of type DIFFER. We translate the branching constraint as follows, in order to obtain an inequality to be added to the pricing problem:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \leq 0 \quad \Rightarrow \quad \chi^s + \chi^t - \zeta_p \leq 1$$

Note that, if $\zeta_p = 1$, this constraint is redundant.

- the child of type SAME. We need to add two constraints to the pricing problem, that guarantee that the value taken by variables χ^s and χ^t is the same:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \geq 1 \quad \Rightarrow \quad \begin{cases} \chi^s - \chi^t - \zeta_p \leq 0 \\ -\chi^s + \chi^t - \zeta_p \leq 0 \end{cases}$$

Also in this case, note that these constraints become redundant if $\zeta_p = 1$.

- if $g^s_p = 1$ or $g^t_s = 1$, we have still two child nodes, similar to the ones generated before with the difference that, now, the constraints are defined for $\zeta_p = 1$ and must be redundant when $\zeta_p = 0$:
  - the child of type DIFFER:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \leq 0 \quad \Rightarrow \quad \chi^s + \chi^t + \zeta_p \leq 2$$

  - the child of type SAME:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \geq 1 \quad \Rightarrow \quad \begin{cases} \chi^s - \chi^t + \zeta_p \leq 1 \\ -\chi^s + \chi^t + \zeta_p \leq 1 \end{cases}$$

- if $g^s_p = g^t_p = 2$, we need to generate three different nodes:
  - the child of type BI-DIFFER. This condition is ensured adding a constraint to the pricing problem as follows:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \leq 0 \quad \wedge \quad \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \leq 0 \quad \Rightarrow \quad \chi^s + \chi^t \leq 1$$

  - the child of type SAME0. We require now three more inequalities for the pricing problem:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \geq 1 \quad \wedge \quad \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \leq 0 \quad \Rightarrow \quad \begin{cases} \chi^s - \chi^t - \zeta_p \leq 0 \\ -\chi^s + \chi^t - \zeta_p \leq 0 \\ \chi^s + \chi^t + \zeta_p \leq 2 \end{cases}$$

  - the child of type SAME2, similar to the previous one:

$$\sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 0}} \lambda^q \leq 0 \quad \wedge \quad \sum_{\substack{q:\, g^s, g^t \in G^q \\ h^q_p = 1}} \lambda^q \geq 1 \quad \Rightarrow \quad \begin{cases} \chi^s + \chi^t - \zeta_p \leq 1 \\ \chi^s - \chi^t + \zeta_p \leq 1 \\ -\chi^s + \chi^t + \zeta_p \leq 1 \end{cases}$$

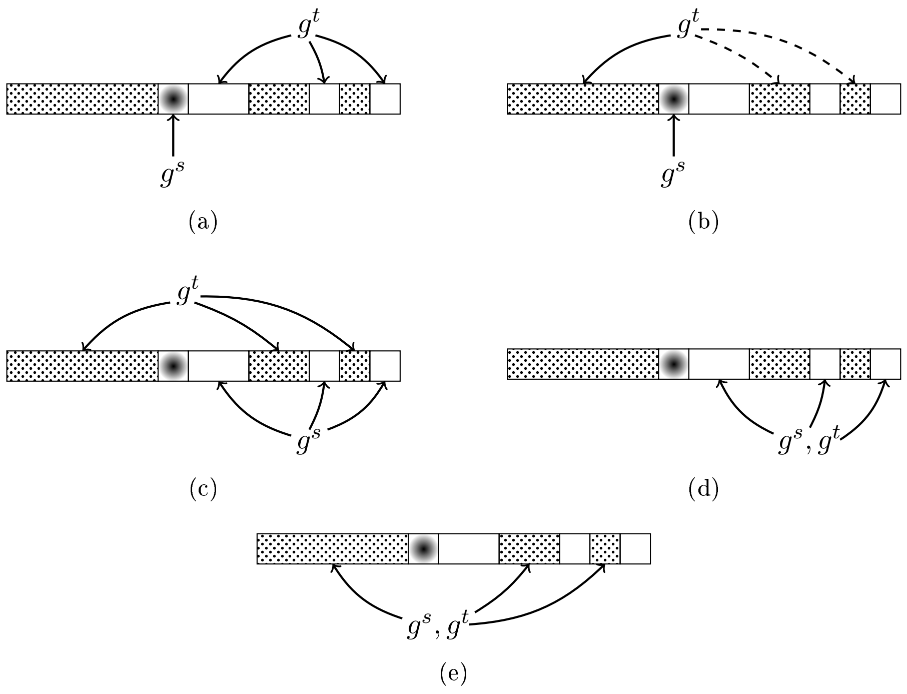


Figure 7.3: Cases arising during the Smart Enumeration approach

7.2.2 Embedding branching constraints in the Smart Enumeration approach

The so far presented expedients to avoid the generation of forbidden variables are enough to obtain the optimal solution of the RMP's linear relaxation if we are solving the pricing problem as an integer problem by simply linearizing it. In this case, in fact, all the pricing problem's variables and constraints are defined. When, however, we apply the Smart Enumeration procedure, the addition of these constraints is not sufficient any more. In fact, as we are reducing the number of genotypes considered in order to find a suitable Q-pair, it is not obvious that both genotypes g^s and g^t involved in the forbidden Q-pairs are taken into account in the pricing problem.

Suppose that at the current node the branching constraint involves genotypes g^s and g^t and position p. For ease of explanation, we fix here different roles for the genotypes g^s and g^t, but note that for the implementation we should consider that these roles can be reversed. We can identify five different cases, according to where the genotypes are located, as shown in Figure 7.3. Each subfigure of Figure 7.3 represents the set of genotypes sorted according to the predefined order during the solution of a reduced pricing problem. The big spot represents the genotype $\bar{g}$ we force to be included in the solution by the partitioning strategy, the blank area represents the genotypes considered for the reduced pricing problem, that are the ones in the set

$$G_{\bar{g}} = \{g^r \in K \,|\, g^s \prec g^r \text{ and } g^s_p + g^r_p \neq 1 \,\forall\, p \in P\},$$

while the dotted area represents the genotypes that we already excluded from this computation due to compatibility reasons or because we already analyzed them.

Case	TYPE	
(a)	DIFFER/BI-DIFFER	fix $\chi^t = 0$
	SAME	fix $\chi^t = 1$
	SAME0	add the constraint $\chi^t + \zeta_p = 1$
	SAME1	add $\chi^t - \zeta_p = 0$
(b)	DIFFER/BI-DIFFER	do nothing
	SAME	the pricing problem is infeasible
	SAME0	fix $\zeta_p = 1$
	SAME1	fix $\zeta_p = 0$
(c)	DIFFER/BI-DIFFER	do nothing
	SAME	fix $\chi^s = 0$
	SAME0	add the constraint $\chi^s - \zeta_p \leq 0$
	SAME1	add $\chi^s + \zeta_p \leq 1$
(d)	all	add the constraints for the complete pricing problem
(e)	all	do nothing

Table 7.1: Cases for pricing problem in Smart Enumeration approach

Subfigure 7.3a shows the case in which $\bar{g} = g^s$ and $g^t \in G_{\bar{g}}$. The reduced pricing problem is defined only on the subset of genotypes identified by $G_{\bar{g}}$. If we do not allow both genotypes to appear in the solution, we need to force the variable χ^t associated to g^t to take value 0. Otherwise, if the branching constraint is of type SAME, we need to force the presence of genotype g^t in the solution by setting $\chi^t = 1$. If the branching constraint is of type SAME0 or SAME1 we need to add an extra constraint to the pricing problem that forces the presence of g^t in the solution only if the haplotype's value in position p is compliant with the branching constraint.

Subfigure 7.3b represents the case in which $\bar{g} = g^s$ but $g^t \notin G_{\bar{g}}$. In this case if the branching constraint is of types DIFFER or BI-DIFFER, we do not need to force the value of any variable. Otherwise, if the constraint is of type SAME the pricing problem results to be infeasible, while for the SAME0 and SAME1 types we need to force the value of the haplotype in position p.

In Subfigure 7.3c only g^s belongs to $G_{\bar{g}}$ and g^t is not included in the definition of the pricing problem. Thus, while branchings of type DIFFER or BI-DIFFER do not require extra constaints, type SAME requires χ^s to be forced to 0 and types SAME0 and SAME1 require extra constraints that fix χ^s to 0 if ζ_p takes value respectively 0 or 1.

Finally Subfigures 7.3d and 7.3e show the cases in which respectivly both g^s and g^t belong to $G_{\bar{g}}$ or none of them belong to $G_{\bar{g}}$. In the first case, we just need to consider the additional constraints as seen for the complete pricing problem, in the second case instead this particular branching constraint has no influence on the current pricing problem, so no extra constraints are needed.

We summarize these different cases and how to deal with them in Table 7.1.

CHAPTER 8

Computational results

The algorithms presented in the previous chapters have been implemented in C++ using the SCIP 3.2 libraries [1, 2], and Cplex 12.4 [18] as solver. They have then been tested on instances both collected from real biological data or generated by using ad-hoc utilities. The tests have been run on an Intel Premium Dual Core E2160 1.8 GHz processor with 4Gb RAM with a time limit of two hours. In this chapter we compare the performances of the different algorithms introduced for solving the linear relaxation of (NewEIP) and (EIPR), with particular attention to the different approaches to solve the pricing problem, as well as the behaviour of the proposed branching procedures. The parameters needed, in particular the stabilization parameter, have been tuned using the **irace** package [36]. We recall that the branch-and-price solution process follows the diagram in Figure 7.2. It uses the procedures seen in Chapter 5 to solve the linear relaxation with column-generation procedures and applies the branching strategy shown, respectively, in Algorithm 6.17 for the (NewEIP) model and in Algorithm 6.18 for the (EIPR) formulation. Moreover, Algorithm 6.17 looks for branching constraints that force two genotypes to belong to the same Q-pair or not. This branching rule is enough to guarantee optimality. Algorithm 6.18 instead needs to modify the formulation of (EIPR) whenever at a node of the branching tree no other branching constraint is found and the solution is still non feasible for the HIPP problem.

The polynomial formulation (PIP'), used for comparison, has been implemented with standard mixed integer programming solvers. We recall that the model presented in Section 4.1 is based on a representation of the solution using class representatives: the selected haplotypes induce a subset of genotypes S_i, where i is the index associated to the smallest genotype in S_i, on a predefined order. Each genotype in S_i is resolved using the haplotype that induces it.

8.1 Alternative algorithm configurations

In Table 8.1 we summarize the combinations of algorithms that we used: the first column gives an identifying name for each of these combinations, the second and

	Stab		PP			# variables	
	F	V	LIN	SM	ESM	larger	multi
LIN	×	×	✓	×	×	×	×
LINs	✓	×	✓	×	×	×	×
LINsl	✓	×	✓	×	×	✓	×
LINsm	✓	×	✓	×	×	×	✓
LINvm	×	✓	✓	×	×	×	✓
SM	×	×	×	✓	×	×	×
SMs	✓	×	×	✓	×	×	×
SMsl	✓	×	×	✓	×	✓	×
SMsm	✓	×	×	✓	×	×	✓
SMvm	×	✓	×	✓	×	×	✓
ESMsm	✓	×	×	×	✓	×	✓

Table 8.1: Overview of algorithms' configurations.

	#genotypes		#SNPs		# instances	% heterozygous sites		
Class	min	max	min	max		avg.	min	max
uniform	21	43	30	50	45	28.81	18.13	40.00
non-uniform	20	43	30	50	45	17.76	7.40	28.13
manygen	80	120	20	30	100	19.52	12.06	32.47
hapmap	5	46	30	75	24	36.99	18.76	55.70

Table 8.2: Classes of instances.

third columns show which methods use stabilization with fixed (F) or variable (V) parameter Δ, the following three columns show which solution method is applied to solve the pricing problem (the linearization of the quadratic problem - LIN, the Smart Enumeration - SM or the Early-terminated Smart Enumeration - ESM), and the last two columns show whether we are adding more than one variable per iteration, and in this case which approach have been used between the proposed two: the one that adds only one more variable with associated the largest subset of genotypes possible, or the one that adds a set of more variables as seen in Section 5.3.

8.2 Instance benchmarks

We report in Table 8.2 the different classes of instances used to test the algorithms. The instances from the first three proposed classes, that is classes *uniform*, *non-uniform* and *manygen*, have been generated using *ms* [29], a computer program that generates samples of haplotypes' sets drawn from an evolving population. The haplotypes are generated by following a coalescent theory and, according to what can be observed in nature, the same haplotype can be repeated several times. The program allows the user to decide the size of the samples to be generated, together with a large number of options with which a recombination level can be chosen and migrations or other demographic events can be taken into account. After building the set of haplotypes in this way, they are paired randomly to generate genotypes.

We consider portions of the DNA in which recombination does not occur. Thus, we chose the instances included in the *uniform* and *non-uniform* classes among the respective classes presented in [13], selecting those for which recombination is not considered. As described in [13], genotypes for the uniform instances have been paired choosing uniformly among the distinct haplotypes generated by *ms*, while genotype for the non-uniform class are built giving to each haplotype a weight proportional to the number of times it can be found among the set produced by *ms*. The number of genotypes per instance that are generated vary between 30 and 50, and there can be duplicated genotypes.

We generated new instances within the *manygen* class, that are characterized by a larger number of genotypes and a limited number of SNPs. In particular, we built a set of 100 instances of 80, 90, 100, 110 or 120 genotypes and 20 or 30 SNPs (10 instances per type) as follows: let m be the final target number of genotypes, we generate 10 samples of $3m$ haplotypes with either 20 or 30 SNPs using *ms*. For each of these sets, we randomly pair, in the non-uniform way, two haplotypes to create a genotype until we obtain a set of m distinct genotypes. In this way, our instances do not include twice the same genotype, which is more compliant with the definition of HIPP.

The *hapmap* class, finally, represents real biological data. These data have been available since the end of the first phase of the HapMap project [13, 17]. These instances consider inputs from chromosomes 10 and 21, over all four HapMap populations. For each input length they selected a continuous collection of SNPs where the probability of incurring in recombination was very low. The number of genotypes in these instances ranges between 5 and 46, while the number of SNPs considered varies between 30, 50 and 75.

8.3 Choosing parameters and algorithm configuration

Preliminary results on the different classes of instances showed how the approach presented in this thesis becomes competitive, compared with state-of-the-art algorithms, when we consider a larger number of genotypes and a limited number of genotypes, namely when we consider instances belonging to the *manygen* class. For this reason, the tuning of the algorithm has been performed on sampling instances that own these features.

8.3.1 Tuning of the parameters

The tuning of parameters has been carried out using the **irace** package [36]. It implements the *iterated racing* procedure as an `R` package [43] and it builds upon the **race** package [8]. The main purpose of **irace** is to automatically configure optimization algorithms by finding the most appropriate settings given a set of tuning instances for an optimization problem. Iterated racing is a method for automatic configuration that consists of three steps:

1. sampling new configurations according to a particular distribution,
2. selecting the best configurations from the newly sampled ones by means of racing,

3. updating the sampling distribution in order to bias the sampling towards the best configurations.

We have used the **irace** package with the purpose of tuning the parameter Δ needed by the stabilization of the dual variables within the resolution of the linear relaxation of the RMP (see Section 5.4). We used a set of sampling instances made of 2 instances per type sharing the characteristics of the *manygen* class (80, 90, 100, 110 or 120 genotypes and 20 or 30 SNPs), with a total of 20 instances and we asked the tool to compare the objective function value obtained after 40 seconds of running. Note that for this tuning we solved the pricing problem using the Smart Enumeration approach, as it will be shown soon that it is more efficient than the LIN approach. Moreover we added only one variable per iteration.

The algorithm gave back the value $\Delta = 0.13$ as best candidate, that will be the one considered from here on.

Afterwards, we used the same tool to choose which among the two branching priorities proposed in Section 6.1 is the most efficient. To this end, we solved the same sample instances using the Early-terminated Smart Enumeration, with a time limit of 400 seconds that in most cases is enough to obtain the optimal solution, so we are sure we reached the time in which branching is applied. The outcome shows that there is no priority that clearly outperforms the other one, so from now on we will use the first one, that consists in choosing the most fractional candidate branching constraint.

8.3.2 Comparing alternative formulations

The first results that we present here, related to the instances belonging to the *manygen* class, aim at showing how the relaxation of constraints (6.2) included in (NewEIP) into constraints

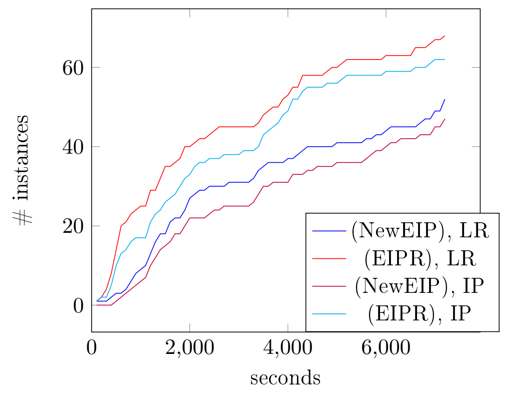
$$\sum_{q:\, g^k \in G^q} \lambda^q \geq 2 \qquad \forall\, k \in \{1, \dots, |K|\}$$

included in (EIPR) affects the computational times and the structure of the solution. In Table 8.3 we report and compare the performances of the branch-and-price algorithm applied to the (NewEIP) and (EIPR) formulations proposed in Chapter 6: in the first column we show the identifier of the methods on which we tested these differences, then the following four columns are dedicated to the results on the linear relaxation, two columns for (NewEIP) and two columns for (EIPR), and the last four to the integer problem, also divided into two parts for (NewEIP) and (EIPR). In each of these cases we compare the percentage of instances solved within the time limit and the average time in seconds used by the solved instances. We note that, with respect to (EIPR) formulation, there is an overall increase on the number of instances solved within the time limit, 12.00% for the linear relaxation and 12.00% also for the integer problem, and the average computational time decreases respectively of the 24.91% and 26.07%. In Figure 8.1 we give a graphical representation of these results where, for each subfigure, we show the number of instances solved within a certain amount of time.

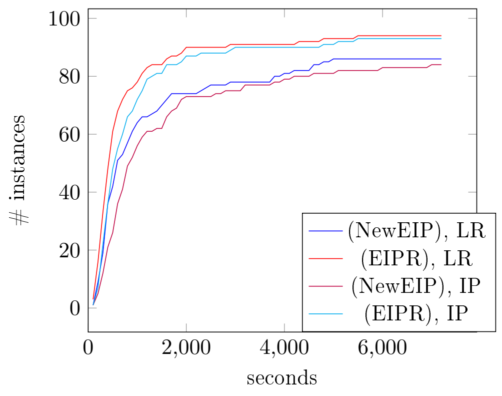
Seen that the improvement in efficiency is significant, we also want to check how the optimal solution obtained from the (EIPR) formulation has been reached. That is,

	LR				IP			
	(NewEIP)		(EIPR)		(NewEIP)		(EIPR)	
	%solved	time	%solved	time	%solved	time	%solved	time
LINsm	52.00	2948.33	68.00	2284.10	47.00	3149.13	62.00	2396.06
SMsm	86.00	1000.73	94.00	673.19	84.00	1196.14	93.00	816.17

Table 8.3: Improvement of performance with relaxed constraints.



(a) LIN approach for PP



(b) SM approach for PP

Figure 8.1: Improvement of performance with relaxed constraints.

we want to check whether some constraints needed to be forced into equalities or not. If not, we want to know how we recovered the optimal feasible solution. In Table 8.4, column *%int* shows the percentage of quasi-integer solutions obtained only applying the branching rule presented in Algorithm 6.17 and in column *%feas* we show how many of these, in percentage, already yield a feasible solution for the HIPP problem. The next column *%W* shows the percentage of over-resolved genotypes, that we recall are those genotypes that appear in more than two Q-pairs, while the following three columns show how the steps of the fixing procedure shown in Algorithm 6.19 possibly recover the feasibility of the current solution. Column *%R* shows the percentage of over-resolved genotypes that appear in a Q-pair whose haplotype is redundant for the genotype considered, column *%MR* shows the percentage of extra redundant (and then, fixed) haplotypes associated to the genotype and column *%E* shows the percentage of over-resolved genotypes that actually are not resolved directly by using the information in the proposed Q-pairs, but they can be resoved using one of the haplotypes associated to it and another haplotype already in the solution. Finally, the last column *%NR* shows the percentage of over-resolved genotypes that are still unresolved and call for a modification on the current formulation that brings back equality constraints. As can be easily seen, for the *manygen* class of instances we are always able to recover a feasible solution, without modifying any constraint of type (6.14). Note that the percentages of columns *%R* and *%E* sum to a value greater than 100%, as some genotypes are both non resolved and associated to a redundant haplotype. Moreover, we can note how there is a progressive deterioration on the feasibility of solutions between the LINsm, SMsm and ESMsm approaches.

From now on, therefore, all the results proposed have been obtained applying the

	%int	%feas	%W	%R	%MR	%E	%NR
LINsm	100.00	3.23	19.41	92.55	0.00	10.20	0.00
SMsm	100.00	2.15	16.56	89.92	0.00	12.19	0.00
ESMsm	100.00	1.00	16.92	85.11	0.00	17.07	0.00

Table 8.4: Non-optimality of solution obtained applying only Algorithm 6.17.

	(PIP')	(EIPR)
uniform	2.56	0.00
non-uniform	2.46	0.26
manygen	0.15	0.00
hapmap	5.67	1.90

Table 8.5: Percent optimality gap of the linear relaxation.

	(PIP')			(EIPR) + SMsm			(EIPR) + ESMsm		
	%solved	time	#nodes	%solved	time	#nodes	%solved	time	#nodes
uniform	100.00	8.18	1.84	82.22	935.22	1.38	86.67	266.38	1.15
non-uniform	100.00	26.05	11.13	100.00	472.48	1.00	93.00	542.68	1.81
manygen	97.00	2448.53	5.24	93.00	816.17	5.08	100.00	119.5	4.42
hapmap	87.5	143.42	55.95	62.5	129.18	1.00	50.00	445.64	4.75

Table 8.6: Solving to integrality: comparison on different classes of instances.

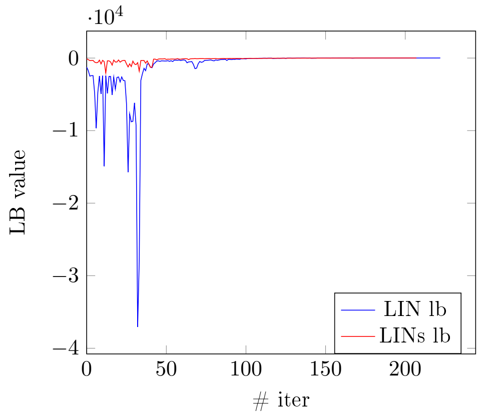
branch-and-price algorithm to the (EIPR) model.

8.4 Computational results on different benchmarks

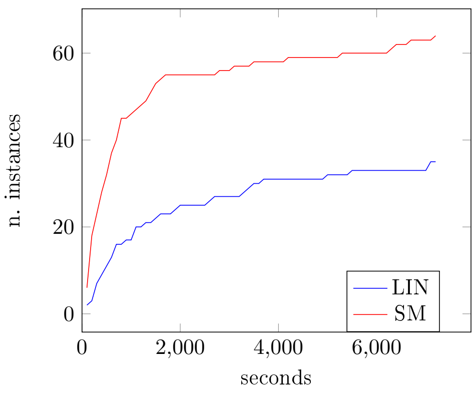
We present some preliminary results on the different classes of instances.

In Table 8.5 we show the performances of our approach on the four sets of instances *uniform*, *non-uniform*, *manygen* and *hapmap*. The table reports the percent opitmality gap at the root node, computed as $100\,(LR - opt)/opt$, where LR is the value of the linear relaxation and opt the optimal integer solution value. We compare two HIPP formulations: (PIP') and (EIPR), and give the average gap for the instances solved within the time limit. The results show that the exponential-size formulation (EIPR) is tighter than the polinomial-size (PIP').

In Table 8.6 we present the performances of the (PIP') model compared with model (EIPR) solved using the Smart Enumeration or the Early-terminated Smart Enumeration. The computational times here considered only take into account the time needed to perform the procedure proposed in Algorithm 6.17 in Section 6.1. However, we can already see how for the instances belonging to the *uniform*, *non-uniform* and *hapmap* classes, in which we have a limited number of genotypes, model (PIP') outperforms our proposed approach. Due to this observation, we give in the following sections detailed results regarding only the *manygen* class.



(a) Effects of stabilization



(b) Effects of Smart Enumeration

Figure 8.2: Effects of stabilization technique and Smart Enumeration procedure.

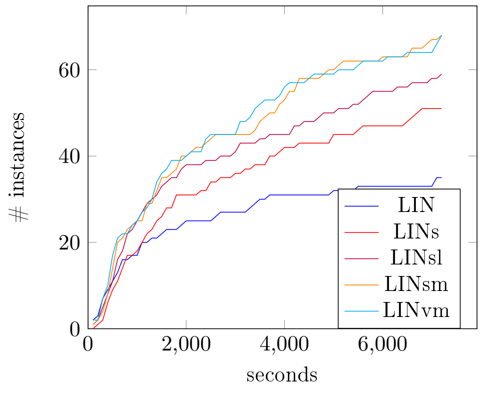
8.5 Computational results on the *manygen* benchmark

The results on the *manygen* instances are presented in two steps: first we show the results obtained from solving the linear relaxation of model (EIPR) and we compare the performances of all algorithm configurations shown in Table 8.1. Then, we select the most promising configurations to be tested on the complete branch-and-price algorithm presented in Figure 7.2.

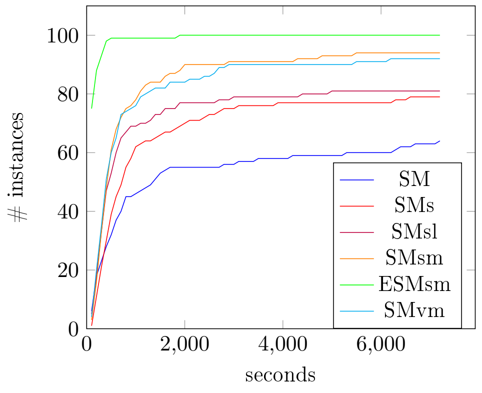
8.5.1 Performance on the solution of the root node

We present the results for the linear relaxation of (EIPR) model. Figure 8.2a shows the effect of the stabilization approach on a sample instance with 100 genotypes and 20 SNPs. As it can be easily seen, the oscillation of the dual variables is highly reduced, so that the lower bound itself does not oscillate as without stabilization. Moreover, we see that the red line stops before the blue one, meaning that the number of times we need to solve optimally the pricing problem is also reduced (and, as a consequence, the computational time decreases). In Figure 8.2b we show instead the performance improvement of solving the pricing problem using the Smart Enumeration approach: the chart reports the number of instances solved within the time showed on the x axis. As we can see, the red line is significantly higher than the blue one, ending at solving 63 instances against 35 within the time limit of two hours.

Figure 8.3 compares the performances of further improvements, such as the use of a variable stabilization parameter, the addition of multiple variables per iteration and the use of the Early-terminated Smart Enumeration. Results are divided according to the different approaches used to solve the pricing problem (LIN or SM). As we can see, for both the approaches we have that applying the stabilization technique and adding multiple variables per iteration sensibly improve the number of instances we are able to solve within the time limit, while using a variable stabilization parameter Δ does not lead to significant improvements. Moreover, using the Smart Enumeration approach to solve the pricing problem (sPP) is more efficient than the LIN approach, as we are able to solve all the instances within the time limit when the Early-terminated approach is used. Note that the shape of the colored lines in Figure 8.3b suggests that a large



(a) LIN



(b) SM

Figure 8.3: Performance profile graphs on different solving approaches.

	All			20 SNPs			30 SNPs		
	%solved	time	#frac	%solved	time	#frac	%solved	time	#frac
LIN	35.00	1756.47	49.2	62.00	1501.07	50.94	8.00	3735.82	35.75
LINs	51.00	2230.50	48.25	82.00	1776.85	51.54	20.00	4090.48	34.80
LINsl	59.00	2199.62	38.19	92.00	1827.81	40.91	26.00	3515.28	28.77
LINsm	68.00	2284.10	12.04	96.00	1722.49	12.06	40.00	3631.97	7.64
LINvm	68.00	2225.82	12.81	94.00	1526.97	14.64	42.00	3789.91	7.67
SM	63.00	1053.58	48.89	86.00	680.94	51.88	40.00	1854.74	42.45
SMs	79.00	911.31	48.91	98.00	696.14	54.82	60.00	1262.74	39.27
SMsl	81.00	626.11	38.40	100.00	446.90	41.46	62.00	915.15	32.93
SMsm	94.00	673.19	12.46	98.00	312.85	13.96	90.00	1065.56	10.29
SMvm	92.00	686.95	12.52	98.00	306.78	14.22	86.00	1120.17	10.13
ESMsm	100.00	102.90	12.48	100.00	40.96	13.55	100.00	164.84	11.18

Table 8.7: Computational results for the linear relaxation: average and by number of SNPs.

part of the instances are solved in a short time, while fewer are solved with a higher computational time. Instead, for the LIN approach the computational times are more evenly spread along the two hours.

We give more details on the computational results in Table 8.7, where we show the overall behaviour of all instances as well as those for instances divided according to the number of SNPs, and Table 8.8, where we compare the performances of the approaches according to the number of genotypes contained in the instances. The first column of each section records the number of instances solved within the time limit, the second column shows the average time, only regarding the percentage of solved instances, and the third shows the average number of fractional variables belonging to those instances that do not already reach an integer feasible solution simply solving the linear relaxation.

Concerning the optimality gap at the root node, we notice that the optimal value of the linear relaxation for all these instances is equal to the integer optimal value, even if the solution obtained is fractional.

	80 genotypes			90 genotypes			100 genotypes		
	%solved	time	#frac	%solved	time	#frac	%solved	time	#frac
LIN	45.00	1633.32	21.22	40.00	682.71	21.22	30.00	2769.17	29.33
LINs	65.00	1753.25	24.54	60.00	1883.22	39.67	50.00	2730.22	46.70
LINsl	65.00	1045.46	14.54	65.00	1482.26	29.31	55.00	2362.71	14.18
LINsm	70.00	1115.36	9.29	80.00	2014.67	7.13	75.00	2665.19	10.00
LINvm	70.00	1173.76	10.71	80.00	1849.14	7.13	75.00	2613.54	9.2
SM	55.00	1120.16	24.73	60.00	335.34	37.75	65.00	1369.94	44.62
SMs	70.00	977.44	20.64	80.00	488.83	39.06	95.00	1289.06	49.11
SMsl	75.00	471.41	19.80	85.00	504.68	26.76	90.00	864.96	35.00
SMsm	95.00	845.00	8.11	95.00	436.01	6.47	95.00	408.09	8.58
SMvm	85.00	738.99	9.24	95.00	466.33	6.89	95.00	414.84	8.84
ESMsm	100.00	100.17	4.00	100.00	146.11	8.60	100.00	67.04	7.55

	110 genotypes			120 genotypes		
	%solved	time	#frac	%solved	time	#frac
LIN	30.00	2058.90	75.00	30.00	2057.75	86.50
LINs	35.00	2355.41	73.71	45.00	2730.50	75.89
LINsl	60.00	3496.61	49.25	50.00	2896.83	60.10
LINsm	70.00	3456.32	10.64	45.00	2122.53	15.89
LINvm	75.00	3698.42	12.87	40.00	1332.14	16.88
SM	70.00	781.44	65.53	65.00	1636.93	65.54
SMs	80.00	715.36	68.19	70.00	1039.25	66.14
SMsl	80.00	465.99	51.31	75.00	802.60	52.80
SMsm	95.00	517.81	12.32	90.00	1186.01	15.17
SMvm	95.00	539.74	12.32	90.00	1313.31	15.22
ESMsm	100.00	80.41	11.50	100.00	120.78	13.90

Table 8.8: Computational results for the linear relaxation: by genotype's number.

8.5.2 Performance of the branch-and-price algorithm

With respect to the branch-and-price algorithm described in Figure 7.2, in Figure 8.4 we give the profile performance graph obtained by solving the RMP with the three different algorithms for the pricing problem (sPP). As in the linear case, the most efficient one results to be the Early-terminated Smart Enumeration, that does not lose in the branching procedure the efficiency it had in the linear relaxation. In the profile graph we also include the performance of model (PIP') presented in Section 4.1, that is the state-of-the-art polynomial-sized approach to solve the HIPP problem. The computational time needed has a clear dependence on the size of the instance, thus resulting in a steps-shaped line. We can note how the algorithm embedding the Smart Enumeration outperforms (PIP'). Tables 8.9 and 8.10 summarize the results: as before, we consider also instances divided according to the number of SNPs and number of genotypes. The columns record the percentage of instances solved within the time limit, the average time and the average number of nodes of the branching tree.

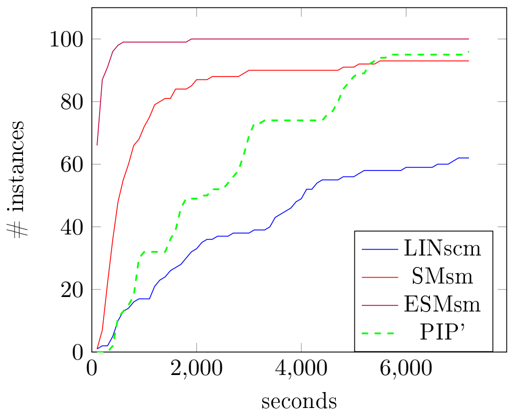


Figure 8.4: Solving to integrality. Performance profile graphs of different approaches.

	All			20 SNPs			30 SNPs		
	%solved	time	#nodes	%solved	time	#nodes	%solved	time	#nodes
PIP'	97.00	2448.53	5.24	100.00	2162.31	1.4	94.00	2753.02	9.32
LINsm	62.00	2396.06	4.85	86.00	1831.34	6.09	38.00	3674.11	2.05
SMsm	93.00	816.17	5.08	98.00	489.32	6.80	88.00	1180.15	3.16
ESMsm	100.00	119.5	4.42	100.00	59.23	5.84	100.00	179.78	3.00

Table 8.9: Solving to integrality. Computational results: average and by number of SNPs

	80 genotypes			90 genotypes			100 genotypes		
	%solved	time	#nodes	%solved	time	#nodes	%solved	time	#nodes
PIP'	95.00	1256.64	17.95	100.00	1251.02	3.40	95.00	2062.25	2.79
LINsm	65.00	1102.21	4.08	80.00	2142.57	3.06	65.00	2500.24	5.62
SMsm	95.00	1052.90	3.53	95.00	477.68	3.05	95.00	636.26	6.32
ESMsm	100.00	119.60	2.30	100.00	152.45	3.55	100.00	84.42	4.5

	110 genotypes			120 genotypes		
	%solved	time	#nodes	%solved	time	#nodes
PIP'	100.00	2862.34	1.00	95.00	4851.67	1.37
LINsm	65.00	4015.97	5.54	35.00	2176.43	7.71
SMsm	95.00	717.67	7.58	85.00	1241.04	4.88
ESMsm	100.00	99.29	6.20	100.00	141.76	5.55

Table 8.10: Solving to integrality. Computational results: by genotype's number.

CHAPTER 9

Conclusion

Mixed Integer Linear Programs for large-scale problems are difficult to solve with standard approaches, as the amount of data to be considered usually makes them impractical. The advisable approach to deal with such programs consists then in using decomposition methods. An example of such a large-scale problem is the Haplotype Inference by Pure Parsimony (HIPP) problem, that can be modeled in different ways. In particular polynomial-sized or exponential-sized formulations have been provided in the literature. With the constant improvements on the hardware and software architectures, instances of significant size can be considered. However, finding optimal solutions in an acceptable computational time remains a challenging goal.

In the literature we can find several formulations for the HIPP problem, mainly divided into two classes: with respect to the input data, there are models with an exponential number of constraints and variables, or formulations with a polynomial number of constraints and variables. The instances that are solved using these formulations within an acceptable computational time are limited in size, thus some heuristic approaches have also been investigated. In this thesis we propose a new approach for solving the HIPP problem that, starting from a formulation with an exponential number of variables and a polynomial number of constraints, aims at being competitive with state-of-the-art polynomial formulations. This approach consists in a customized branch-and-price procedure, mainly based on a specialized method to solve the linear relaxation, and an effective branching strategy.

In particular, we have studied several ways of improving the performance of the algorithm used to solve to optimality the linear relaxation by column generation, such as the design and implementation of exact and heuristic methods for solving the pricing problem, the integration of a stabilization technique, and the selection of suitable sets of variables to be added at each iteration. The most effective proposed way of solving the pricing problem exploits a partition of all its feasible solutions in order to reduce the size of the underlying integer linear program. In this way, however, the variable with minimum reduced cost is found only solving several programs per iteration of the column-generation procedure, thus the use of a combination of heuristic and exact methods has proven to be more efficient. We achieved significant improvements also by smoothing the dual variables' values and by early terminating the column-generation procedures. Both these strategies rely on the definition of a Lagrangian lower bound to the optimal solution of the linear relaxation of the HIPP problem. Additional refinements on the solution of the linear relaxation include the choice of an a-priori order on the genotypes that improves the performance of order-dependent sub-procedures.

The column-generation algorithm has been embedded in a branch-and-price framework. The proposed branching rule is able to guarantee the integrality of variables by exploiting some information derived from the constraints of the master problem, in a way that recalls the well known Ryan-Foster branching strategy for partitioning problems, and that can be effectively embedded in the pricing problem procedures, both exact and heuristics.

Moreover, in order to take further advantage from the similarity between the constraints involved in our formulation and set-partitioning constraints, we modify the original model to take a set-covering shape with improved performance in the solution of the linear relaxation. Due to the structure of the new model, an extension of the branching procedure is required to recover integer feasible solutions for HIPP. Nevertheless, the computational results show that the overall branch-and-price performance takes advantage from this set-covering-like formulation.

The proposed algorithms have been implemented in C++ using the SCIP libraries. We carried out extensive computational experiments using Cplex as linear programming solver. Numerical results for different classes of instances proved that the strategies implemented to improve the the column-generation algortihm to solve the linear relaxation are effective, since we notice a significant reduction on the computational times. Moreover, the formulation of the linear relaxation proved to be rather tight, as its optimal value is often equal to the integer optimal value, even if the solution is still fractional. Results on the implementation of the branch-and-price algorithm show how our approach is competitive with state-of-the-art polynomial formulations for instances with a large number of genotypes of limited length.

Further improvements to this approach could come from using a more efficient heuristic to provide an initial solution, investigating different orderings for the genotypes and refine the application of the Early-terminated Smart Enumeration, for example finding alternative rules to update the stability center. Moreover, it would be interesting to study the viability of different, ad-hoc branching strategies for the set-covering relaxation which are directly able to guarantee the integrality of the variables and the overall feasibilty of the solutions.

A further future line of research is the extension of the formulation studied in this thesis to a more general set-partitioning context. Specifically, we can consider haplotypes as elements that characterize the subset of genotypes associated to them. Thus, the formulation constraints require that each genotype belongs to two of these subsets. Moreover, the two subsets chosen must satisfy additional requirements on the haplotypes, as they must resolve the genotype. Thus, the approach presented in this thesis could be considered as a starting point for solving multiple set-partitioning formulations with additional constraints.

Bibliography

[1] T. Achterberg. “SCIP: Solving constraint integer programs”. In: *Mathematical Programming Computation* 1.1 (2009). `http://mpc.zib.de/index.php/MPC/article/view/4`, pp. 1–41.

[2] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. “Constraint integer programming: a new approach to integrate CP and MIP”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2008, LNCS 5015* (2008), pp. 6–20.

[3] B. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. “Haplotyping as perfect phylogeny: a direct approach”. In: *Journal of Computational Biology 10(3-4)* (2003), pp. 323–340.

[4] M. L. Balinski. “On a selection problem”. In: *Management Science 17* (1970), pp. 230–231.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. “Branch-and-price: Column generation for solving huge integer programs”. In: *Operations Research, 46(3)* (1998), pp. 316–329.

[6] P. Berman and M. Karpinski. “On some tighter inapproximability results”. In: *Technical report ECCC No. 29, Department of Computer Science, University of Trier, Germany* (1998).

[7] P. Bertolazzi, A. Godi, M. Labbé, and L. Tininini. “Solving haplotyping inference parsimony problem using a new basic polynomial formulation”. In: *Computers and Mathematics with Applications 55* (2008), pp. 900–911.

[8] M. Birattari. “The race Package for `R`: Racing methods for the selection of the best”. In: *Technical Report TR/IRIDIA/2003-037, IRIDIA, Université libre de Bruxelles, Belgium* (2003).

[9] R. G. Bland. “New finite pivoting rules for the simplex method”. In: *Mathematics of Operations Research 2(2)* (1977), pp. 103–107.

[10] P. Bonizzoni, G. Della Vedova, R. Dondi, and J. Li. “The haplotyping problem: an overview of computational models and solutions”. In: *Genome Sequencing Technology and Algorithms, 1st ed., S. Kim, H. Tang, and ER Mardis, Eds. Artech House, Inc* (2008), pp. 151–181.

[11] S. Bradley, A. Hax, and T. Magnanti. *Applied Mathematical Programming.* 1977.

[12] D. Brown and I. Harrower. "A new integer programming formulation for the pure parsimony problem in haplotype analysis". In: *Algorithms in Bioinformatics. Springer Berlin Heidelberg* (2004), pp. 254–265.

[13] D. Brown and I. Harrower. "Integer programming approaches to haplotype inference by pure parsimony". In: (2006).

[14] D. Catanzaro, A. Godi, and M. Labbé. "A class representative model for pure parsimony haplotyping". In: *INFORMS Journal on Computing 22(2)* (2010), pp. 195–209.

[15] A Clark. "Inference of haplotypes from PCR-amplified samples of diploid populations". In: *Molecular Biology and Evolution* 7 (1990), pp. 111–122.

[16] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming.* 2014.

[17] The Int'l HapMap Consortium. "Integrating ethics and science in the international HapMap project". In: *Nature Rev. Genetics 5(6)* (2004), pp. 467–475.

[18] *CPLEX Optimizer.* URL: `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[19] L. De Giovanni and M. Labbé. "A column generation approach for pure parsimony haplotyping". In: (2014). Conference on Operational Research in Computational Biology, Bioinformatics and Medicine.

[20] Z. Ding, V. Filkov, and D. Gusfield. "A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem". In: *Journal of Computational Biology 13(2)* (2006), pp. 522–553.

[21] O. Du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. "Stabilized column generation". In: *Discrete Mathematics, 194(1)* (1999), pp. 229–237.

[22] E. Eskin, E. Halperin, and R.M. Karp. "Large scale reconstruction of haplotypes from genotype data". In: *Proceedings of the seventh annual international conference on Research in computational molecular biology* (2003).

[23] L. Excoffier and M. Slatkin. "Maximum-likelyhood estimation of molecular haplotype frequencies in a diploid population". In: *Molecular Biology and Evolution, 12(5)* (1995), pp. 921–927.

[24] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T Koch, S. J. Maher, Miltenberger M., B. Müller, Pfetsch M. E., Puchert C., D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, Witt J. T., and J. Witzig. *The SCIP Optimization Suite 3.2.* eng. Tech. rep. 15-60. Takustr.7, 14195 Berlin: ZIB, 2016.

[25] D. Gusfield. "Haplotype inference by pure parsimony". In: *Springer Lecture Notes in Computer Science No.2676* (2003), pp. 144–155.

[26] D. Gusfield. "Haplotyping as perfect phylogeny: conceptual framework and efficient solutions". In: *Proc. 6th Annual Conference on Research in Computational Molecular Biology* (2002), pp. 166–175.

[27] D. Gusfield. "Inference of haplotypes from samples of diploid populations: complexity and algorithms". In: *Journal of Computational Biology* 8(3) (2001).

[28] Y. T. Huang, K. M. Chao, and T. Chen. "An approximation algorithm for haplotype inference by maximum parsimony". In: *Journal of Computational Biology, 12(10)* (2005), pp. 1261–1274.

[29] R. R. Hudson. "Generating samples under the Wright-Fisher neutral model of genetic variation". In: *Bioinformatics 18(2)* (2002), pp. 337–338.

[30] B. Korte and J. Vygen. *Combinatorial Optimization.* Springer, 2012.

[31] G. Lancia, M.C. Pinotti, and R. Rizzi. "Haplotyping populations by pure parsimony: complexity of exact and approximation algorithms". In: *INFORMS Journal on computing 16(4)* (2004), pp. 348–359.

[32] G. Lancia and R. Rizzi. "A polynomial case of the parsimony haplotyping problem". In: *Operations Research Letters, 34(3)* (2006), pp. 289–295.

[33] G. Lancia and P. Serafini. "A set-covering approach with column generation for parsimony haplotyping". In: *INFORMS Journal on Computing 21(1)* (2009), pp. 151–166.

[34] M. E. Lübbecke. "Column generation". In: *Wiley Encyclopedia of Operations Research and Management Science* (2010).

[35] M. E. Lübbecke and J. Desrosiers. "Selected topics in column generation". In: *Operations Research 53(6)* (2005), pp. 1007–1023.

[36] M. López-Ibáñez, J Dubois-Lacoste, T Stützle, and M. Birattari. "The irace package, Iterated Race for Automatic Algorithm Configuration". In: *Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, Belgium* (2011).

[37] S. Martello and P. Toth. *Knapsack problems. Algorithms and computer implementations.* John Wiley and Sons, 1990.

[38] T. Niu, Z. S. Qin, X. Xu, and J. S. Liu. "Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms". In: *The American Journal of Human Genetics* (2002), pp. 157–169.

[39] A. Pessoa, E. Uchoa, M. P. de Aragão, and R Rodrigues. "Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems". In: *Mathematical Programming Computation, 2(3-4)* (2010), pp. 259–290.

[40] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. "In-out separation and column generation stabilization by dual price smoothing". In: *International Symposium on Experimental Algorithms. Springer Berlin Heidelberg.* (2013), pp. 354–365.

[41] J.-C. Picard. "Maximal closure of a graph and application to combinatorial problems". In: *Management Science 22* (1976), pp. 1268–1272.

[42] J.-C. Picard and M. Queyranne. "A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory". In: *Networks 12* (1982).

[43] R Development Core Team. "R: a language and environment for statistical computing". In: (2008). URL: `http://www.R-project.org`.

[44] L. M. Rousseau, M. Gendreau, and D. Feillet. "Interior point stabilization for column generation". In: *Operations Research Letters, 35(5)* (2007), pp. 660–668.

[45] D. M. Ryan and B. A. Foster. "An integer programming approach to scheduling". In: *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling* (1981), pp. 269–280.

[46] L. Tininini, P. Bertolazzi, A. Godi, and G. Lancia. "CollHaps: a heuristic approach to haplotype inference by parsimony". In: *Computational Biology and Bioinformatics, 7(3)* (2010), pp. 511–523.

[47] A. Violin. "Mathematical programming approaches to pricing problems". PhD thesis. Université Libre de Bruxelles, Belgium and Università di Trieste, Italia, 2014.

[48] P. Wentges. "Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming". In: *International Transactions in Operational Research, 4(2)* (1997), pp. 151–162.