Efficient Methods of Feature Selection Based on Combinatorial Optimization Motivated by the Analysis of Large Biological Datasets

Mateus Rocha de Paula

M.Sci. (Computer Science)

B.Sci. (Computer Science)

This dissertation is submitted as a partial requirement for the Degree of **Doctor of Philosophy**

Faculty of Engineering and Built Environment

School of Electrical Engineering and Computer Science

Newcastle, NSW, Australia

August, 2012

Statement of Originality

The thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis, when deposited in the University Library[1], being made available for loan and photocopying subject to the provisions of the Copyright Act 1968.

Mateus Rocha de Paula

[1] Unless an Embargo has been approved for a determined period.

Acknowledgements

Matsuo Basho, one of the most celebrated Japanese poets, once wrote *"Every day is a journey, and the journey itself is home"*. When I think about my PhD today, that couldn't sound more true. The degree of Doctor of Philosophy not only represents the completion of this work, but also celebrates a series of meaningful events that allowed me to grow both as a person and professional; and marks the beginning of a new and exciting stage of my life. Every little step I took since I ventured this path has its own story, of immense importance, and none of which would be possible without the help great personalities that I was already fortunate to know, and others that I had the great pleasure to meet on my way. To all those, my many big thanks.

To my mum France, dad José Vicente and sister Rafaella, my eternal gratitude for all the love and support. Every time the light in the end of the tunnel seemed a little distant, it was you who made it brighter. This victory is very much yours too.

Many thanks to my supervisors Pablo and Regina, who so patiently mentored me through all the process, teaching me the skills and giving me the tools that I needed to succeed on this challenge and many yet to come. You not only helped me to be a better researcher, but also taught me how to be truly committed to science, and for that I'll always be thankful. To my friend and colleague Martín for supporting me from the very beginning of my academic career and for opening the doors to this opportunity, to Australia and of his own home to me, making it all possible. It's always been a pleasure to work with such a good friend, always with an open mind, positive attitude and a smile on his face! A big thanks to my friend Alexandre, for all the help both inside and outside the lab. This story would have had a much different ending without your advice, friendship, and many beers and fun times shared. To

my friend and colleague Lee-Anne, for showing me the ways of her people, helping me to feel at home on either continent! To my friend and colleague Carlos also a big thanks, for all the help, tips, teachings and laughter. It is really inspiring to see such a curious and enthusiastic mind working! Even brewing and drinking beer is science around you, and it is good science! And to my friends and colleagues Renato and Ahmed, who shared the path with me, with its challenges and joys.

I'd also like to thank the panel of examiners that reviewed this document. All your remarks were very helpful and constructive. They certainly helped me to improve my thesis, and for that - and also the acceptance and compliments, of course - I thank you!

Last but not least, I'd also like to dedicate this thesis to all my loyal friends, from all over the world. To my country fellows, for always being there for me, reminding me that I'll always have a safe place by their side either at home or from a distance. And to all my new ones as well, for welcoming me to their lands and hearts, giving me the strength to be a citizen of the world.

To Vó Cota, Tio Tonico and Shilly.

Wherever you are.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Intuitively, the Feature Selection problem is to choose a subset of a given a set of features that best represents the whole in a particular aspect, preserving the original semantics of the variables on the given samples and classes. In practice, the objective of finding such a subset is often to reveal a particular characteristic present in the given samples.

In 2004, a new feature selection approach was proposed. It was based on a combinatorial optimization problem called (α, β)-k-Feature Set Problem. The main advantage of using this approach over ranking methods is that the features are evaluated as groups, instead of only considering their individual performance.

The main drawback of this approach is the complexity of the combinatorial problems involved. Since some of them are NP-Complete, it is unlikely that there would exist an efficient method to solve them to optimality efficiently. To the best of the author's knowledge at the moment of this research, the available tools to deal with the (α, β)-k-Feature Set Problem approach can not solve problems of the magnitude required by many practical applications.

Given the big advantage brought by the multivariate characteristic of this method, its successful wide applicability and knowing that its only real known drawback is scalability, further research to overcome such a difficulty is appropriate. Even though the optimal solution of the problem is always desirable, it often is not strictly necessary in the case of many biological applications. Therefore, this work aims to propose fast heuristics to address the (α, β)-k-Feature Set Problem approach, and propose procedures to obtain dual bounds that do not rely on external optimization packages.

Chapter 1

Introduction

According to Luscombe et al. (2001), "Bioinformatics is conceptualizing biology in terms of macromolecules (in the sense of physical-chemistry) and then applying 'informatics' techniques (derived from disciplines such as applied maths, computer science, and statistics) to understand and organize the information associated with these molecules, on a large-scale". Further, they also state that the aims of bioinformatics are three-fold: (1) to organize data, (2) develop tools to analyse it and (3) use these tools to analyse it and interpret the results in a biologically meaningful manner.

These aims have become particularly clear and important with recent advances in technology. New tools like microarray chips allow experiments to be performed in a massively parallel fashion, which generates huge amounts of data that have to be stored and analysed. That creates the need for huge and efficient databases to store all this information, and efficient and scalable tools to process and analyse such a great amount of data.

Furthermore, because of the relatively high cost of microarrays and the scarcity of suitable tissue[1], the data provided by such methods is generally highly underdetermined, in the sense that one typically has many more variables than cases. That means that there may be many solutions that explain the same tableau. Kohane et al. (2002) illustrated that well, providing the following analogy: *"To solve a linear equation of one variable (e.g., $4x = 5$) we only need one equation to find the value of the variable. To solve a linear equation of two variables*

[1]See Kohane et al. (2002), chapter 2 for a discussion of which tissues are appropriate for particular experiments.

(e.g., $y = 4x + b$), two equations are required. If we have tens of thousands of variables, but only hundreds of equations, then there will be thousands of potentially valid solutions. This is the essence of what constitutes an underdetermined system". For these systems, biostatistical methods do not work well (see Kohane et al., 2002, Chapter 1.2.2), so novel techniques are necessary.

For a brief introduction on Bioinformatics, refer to Cohen (2004); Karp (2002); Luscombe et al. (2001). For further information, refer to Baldi and Brunak (2001); Bergeron (2002); Kohane et al. (2002); Lesk (2002). For a more recent review and discussion about new challenges in bioinformatics, in particular genome-wide association study challenges that will require computational methods, see Moore et al. (2010).

1.1 Bioinformatics and Learning Algorithms

One of the most important problems in Bioinformatics is to distinguish between existing classes based on characteristics (*features*) of samples. For example, observing expression patterns one can distinguish between healthy and diseased samples.

To perform this task automatically, *Learning Algorithms*, as studied by Machine Learning, are used. According to Hall (1999), *"Machine learning is the study of algorithms that automatically improve their performance with experience. An algorithm that – when presented with data that exemplifies a task – improves its ability to predict key elements of the task can be said to have learned."*

Supervised learning algorithms, or *classifiers*, attempt to create class descriptions from a training set of samples, labelled with class information, that will enable an independent (unlabelled) set of samples to be correctly identified. The Weka package (see Witten and Frank, 2005) implements several classifiers, among other tools. Many of these classifiers were used to address bioinformatics problems, like the early prediction of Alzheimer's Disease (see Gómez Ravetti and Moscato, 2008; Rocha de Paula et al., 2011) and Prostate Cancer Gómez Ravetti et al. (2009). Yamanishi et al. (2004) proposed a method to infer protein networks from multiple types of genomic data, casting the problem as a supervised learning problem. Tan

and Gilbert (2003) surveyed 16 supervised learning methods on different biological datasets, and provided some suggested issues when choosing an algorithm to address a dataset and comparing its effectiveness to other's.

Unsupervised learning algorithms, or *clustering algorithms*, on the other hand, attempt to decide which samples should be *clustered* together, inferring the classes itself, without the need of a labelled training set. González-Barrios and Quiroz (2003) proposed one such algorithm, based on the comparison between the Minimum Spanning Tree and the k Nearest Neighbours. Inostroza-Ponta (2008) compared MSTkNN, an algorithm inspired by it, to other methods found on the literature. He also uses this method to address genome-wide analysis of datasets in melanoma and prostate cancer, among other applications.

Inostroza-Ponta et al. (2011) dealt with the problem using a two level approach based on the Quadratic Assignment Problem. Tasoulis et al. (2006) used an extension of the k-windows clustering algorithm on a leukaemia microarray dataset. Abeel et al. (2008) used self-organizing maps to distinguish between the structural profiles of promoter sequences and other genomic sequences. Boutros and Okey (2005) discussed the biological motivations and applications of unsupervised pattern recognition to integrate gene expression data with other biological information, such as functional annotation, promoter data and proteomic data.

For further information on Data Mining and Machine Learning, refer to Witten and Frank (2005) and Baldi and Brunak (2001).

1.2 Bioinformatics and Optimization

Many problems that arise in bioinformatics can be cast as optimization problems and solved using well-known computer science techniques.

An optimization problem is the matter of finding at least one of the *best* solutions - as there might be more than one optimal solution - from all feasible ones. This is particularly interesting in scenarios where the associated systems are highly underdetermined (see Kohane et al., 2002). These systems have many solutions that can explain it, but one is often interested in at least one of the best.

An important aspect of such problems is that many of them are classified as *Non-deterministic Polynomial-time Complete* (**NP-Complete**), which are the optimization problems that are at least as hard as the hardest Non-deterministic Polynomial-time (**NP**) problems. Formally, they satisfy two properties: (1) they belong to the Non-deterministic Polynomial-time (**NP**) class, and (2) are Non-deterministic Polynomial-time Hard (**NP-Hard**). The **NP** class comprises all decision problems ("yes" or "no") where the correctness of a "yes" answer can be verified in polynomial time on the size of the instance. A problem is **NP-Hard** if it is "at least as hard as the hardest problems in NP", or in other words, if instances of every problem in **NP** can be cast as its instances. There are no known efficient algorithms - in the sense that they run in polynomial time - that can solve problems in this class, and it is widely suspected that they do not exist, although it has never been proven. In fact, if one such algorithm is found, it can be used to solve all other problems in this class. That makes the study of this class of algorithms one of the greatest existing challenges in computer science. Refer to Cormen (2001); Garey and Johnson (1979) for more information on this topic.

One observation, however, motivates the use of exact algorithms to solve **NP-Complete** problems: there exists several hard problems that (most likely) require exponential run time when complexity is measured in terms of the input size only, but are computable in polynomial time in the input size and in exponential time in a parameter k. That means that if k is fixed at a small value, these problems are still tractable in practice, in the sense that real world instances can be solved to optimality, even though they are traditionally considered as intractable. The parameterized complexity theory, introduced by Downey and Fellows (1999), studied such problems and proposes the following fixed-parameter hierarchy, in increasing order of difficulty: *Fixed-Parameter Tractable* (FPT), W[1], W[2] and W[P]. For more information on Parameterized Complexity see Downey and Fellows (1999); Flum and Grohe (2006).

Moreover, although there are no know polynomial algorithms to solve these problems to optimality, many of them are still able to solve relatively small, but interesting, practical instances within reasonable time. Many authors exploit that, and other interesting mathematical properties to deal with important combinatorial problems that arise in bioinformatics. Brinza and Zelikovsky (2008); Brinza et al. (2006), for example, develop several combinatorial search

methods to address the problem of Multi Single Nucleotide Polymorphisms (SNP) Disease Association, which is NP-Hard. Wang et al. (2008), in turn, developed an algorithm to find near optimal solutions for a Non-Unique Oligonucleotide Microarray Probe Selection Problem, which is also NP-Hard. Refer to Lancia (2008) for a survey on mathematical programming for computational biology.

That brings an important matter into consideration: the modelling. The first papers on computational biology, the field that "preceded" bioinformatics[2], were very theoretical and often included models of biological problems with proofs of its NP-Hardness. However, they were simple abstractions of the real problems, that many times relied on mathematical properties that made them simpler to work with. Thus, the obtained solutions were often infeasible or suboptimal in practice (see Lancia, 2008). Also, subjective aspects for consideration, like what makes a biomarker or gene more interesting than the other for a particular purpose, often makes it difficult to model biological problems because they cannot be mathematically characterized properly.

Therefore, obtaining the optimal solution, proving the optimality of a solution, or even estimating the quality of a given solution is not always critical or applicable, as the criteria of evaluation itself is not well defined or is a crude simplification of the real world applications. That means that, even if a so called optimal solution is obtained, it is difficult to state that it is the best solution that one can find for its designated purpose. In these cases, where a good solution would suffice as long as it is obtained within reasonable time, *Approximative Algorithms* and *Heuristics* are very interesting choices. Unlike Heuristics, which usually only find (hopefully) good solutions quickly, without any optimality or quality guarantees, Approximative Algorithms find solutions with provable performance worst case quality bounds and provable run time bounds. Please refer to Gonzalez (2007) for more information about Approximation Algorithms and Metaheuristics.

Finally, one last resource that can be used to work with large instances, is to parallelize the algorithms used to solve it. That is, distribute tasks that can be performed simultaneously

[2]The term "preceded" here is used in a very loose sense: only because it used to be used before the word "bioinformatics" became popular. There are still arguments about the distinction of the fields of *bioinformatics* and *computational biology* (see Cohen, 2004).

among many available processors, in such a way that it takes less time to finish. Parallel algorithms have been widely used in optimization to deal with the curse of dimensionality of the inherent problems. Even though it does not "solve" the issue, it often makes it possible to solve many practical instances of difficult problems within reasonable time. Since Bioinformatics is a very practical topic, in which one is often more interested in if an instance in particular is solvable within reasonable time than in whether the problem is tractable, the parallelization of tools is often a very interesting alternative.

1.3 Feature Selection

Intuitively, the Feature Selection problem is: given a set of candidate features, to choose a subset of them that best represents the whole in a particular aspect, preserving the original semantics of the variables on the given samples and classes. This definition shall be revisited formally later on in this text. Very often in practice, the objective of finding such a subset is to reveal a particular characteristic present in the given samples.

There are several reasons to perform feature selection. Among the most important are the reduction of the dimensionality of the dataset and elimination of irrelevant, redundant, or controversial features. These criteria are interesting because they ease the analysis, visualization and interpretation of high-dimensional datasets.

The size reduction is particularly important for many modern datasets, such as genomic ones (see Kohane et al., 2002, Chapter 4.5), because its analysis and interpretation can often be very resource-demanding. See Inostroza-Ponta (2008) for an example where the visualization problem is cast as a Quadratic Assignment Problem, which is intractable for large instances. Also, the interpretation of data often includes an inspection step, which often is not fully automatic, and is significantly easier when less variables are considered. Clark et al. (2012); Inostroza-Ponta (2008); Inostroza-Ponta et al. (2011) exemplify the visualization matter well, with several examples.

The elimination of irrelevant features is also very interesting for highly underdetermined datasets. The idea is that this procedure makes the system under consideration less under-

determined. Intuitively, that means that there is a good chance that many variables do not contain useful information, contain incomplete information (e.g.: that may be relevant but not representative in the considered samples) or contain inconvenient information (e.g.: unrelated information that may mask or confuse the identification of a pursued characteristic in the considered samples).

From the classification point of view, not only it is possible that a good selection of features leads to a better accuracy when predicting classes but it is often the case that irrelevant training information adversely affects machine learning algorithms. For example, Gómez Ravetti and Moscato (2008); Gómez Ravetti et al. (2009), employed the (α, β)-k-Feature Set approach (see Chapter 3) to select features and propose novel biomarkers for the prediction of Alzheimer's Disease and Prostate Cancer, respectively. These biomarkers are shown to be superior to others found in the literature from the classification point of view, since they lead to a better accuracy when predicting classes using classifiers. As argued by Hall (1999), one of the reasons why that often works, is that noisy data with irrelevant and inconvenient features, such as that of microarray experiments, make decision making more difficult for the classifiers. Indeed, this behaviour was observed in several cases, such as decision trees (Langley and Sage, 1994) and bayesian classifiers (Langley and Sage, 1995). Aha (1992); Aha et al. (1991); Langley and Sage (1997) propose new methods to deal with that. The discussion of such matters and methods is, however, not in the scope of this text.

Generally speaking, and according to Hall (1999), "many learning algorithms can be viewed as making a (biased) estimate of the probability of the class label given a set of features. This is a complex, high dimensional distribution. Unfortunately, induction is often performed on limited data. This makes estimating the many probabilistic parameters difficult. In order to avoid overfitting the training data, many algorithms employ the Occam's Razor (Gamberger and Lavrac, 1997) bias to build a simple model that still achieves some acceptable level of performance on the training data. This bias often leads an algorithm to prefer a small number of predictive attributes over a large number of features that, if used in the proper combination, are fully predictive of the class label. If there is too much irrelevant and redundant information present or the data is noisy and unreliable, then learning during the training phase is more

difficult."

1.4 Research Questions

In face of the new challenges imposed by highly underdetermined and high dimensional data, the problem of feature selection should be revisited.

Considering the fact that undetermined systems have many associated solutions that may have to be evaluated, and that this problem becomes even worse when the dataset is large, the first research question that arises is:

"Is it possible to quickly select features on highly dimensional datasets, without compromising the robustness of the solution?"

Further, assuming that such a method can be found, in order to prove its usefulness, it would also be interesting to show that it indeed proposes a good compromise between speed and quality of solutions. Thus, another very important research question also takes place:

"Is it possible to quickly estimate how good these solutions would be?"

In this thesis, answers to the both these research questions are sought by developing combinatorial methods to address the feature selection problem and estimate the quality of the proposed solutions.

1.5 Structure of This Thesis

After the introductory notes given in this chapter, Chapter 2 surveys and contextualises the existing feature selection methods. Even though this chapter aims to be a brief survey on modern feature selection approaches, it does not include specific method-related literature. To ease the reading of this text, these are provided on the beginning of each Chapter or Section, where they are more relevant. The particular method chosen to be studied and extended is then formally stated and detailed in Chapter 3.

Chapter 4 discusses the instances of interest, and defines the testbed that will be used throughout this text.

Next, Chapter 5 discusses, details and proposes efficient methods to obtain feasible solutions for such problems. Such methods include greedy heuristics (Section 5.2), local search procedures (Section 5.3) and metaheuristic approaches that combine these methods (Section 5.4). Section 8.1 summarizes the conclusions obtained after analysing the results depicted on Section 7.1.

Chapter 6 discusses the quality of such solutions and proposes new methods to obtain these quality estimates. These methods include simpler and faster procedures, as detailed on Section 6.1; and more elaborated and expensive methods, as detailed on Section 6.2 and Section 6.3. Each procedure is tested on Section 7.2 and the conclusions summarized on Section 8.2.

Finally, Section 8.3 concludes this text summarizing the key conclusions made on Chapter 4, Chapter 5 and Chapter 6.

Chapter 2

Existing Feature Selection Approaches

In this chapter, the most important and recent works on feature selection are briefly discussed as pertinent to the thesis aims and discussion. According to Hall (1999), the feature selection algorithms found in the literature, in the context of classification, can be organized in two categories: *Filter* and *Wrapper* algorithms. While the former includes algorithms that use only general characteristic of the data to evaluate the worth of the features (or set of features), the latter uses a learning algorithm that will ultimately be applied to the data to evaluate the worth of the features. In other words, *Filter* algorithms tackle the Feature Selection problem independently of the classification step, and *Wrapper* algorithms uses one classifier to evaluate the importance of every feature or feature set tested (e.g.: in terms of accuracy). Thus, as a general rule, *Filter* algorithms tend to be faster, simpler and independent of the choice of classifier. However, considering the same choice of classifier to be used both on the feature set selected by a filter algorithm and to be used within a wrapper method, the accuracy of the latter tends to be better. Yvan Saeys and Larrañaga (2007) also introduced a third category, *Embedded* algorithms, which merge feature selection and classification in one algorithm in the sense that the feature selector actually interacts with the classifier (one uses the others' information to perform its task). Algorithms in this category are often still not as fast as *Filter* algorithms, but faster than *Wrappers*. Yvan Saeys and Larrañaga (2007) also distinguished between two subcategories of *Filter* algorithms: *Univariate* and *Multivariate*. The former, also known as *ranking methods*, are the fastest and simplest ones and do not model feature

dependencies, that is, each feature is evaluated (ranked) individually, and the highest ranked features are selected. The latter, in turn, model feature dependency and are more complex, but differ from *Wrapper* and *Embedded* algorithms in the sense that they are still independent of the classifier. Further, they also differentiates the *Univariate Filter* algorithms in *Parametric*, in which a distribution of the samples is assumed to ease the ranking, and *Model-Free*, in which no distribution is assumed.

2.1 Wrapper Methods

Wrapper methods use the classifier data to guide its search, in attempt to obtain more accurate classification results.

Due to the inherent complexity of the resulting problems and size of the applications' instances, most works that deal with microarray data use heuristic searches, of which the evolutionary algorithms are the most popular. In particular, Duval and Hao (2009); Duval et al. (2009) used a Memetic Algorithm (Moscato, 1989; Moscato and Cotta, 2003; Moscato et al., 2004; Neri et al., 2012) that uses information from a Support Vector Machine (SVM) to select genes for the molecular classification of cancer. More specifically, their fitness function and crossover operator are based on the rankings generated the SVM, and every solution in their population is refined with an Iterated Local Search procedure (Lourenço et al., 2003). Umpai and Aitken (2005) developed a Genetic Algorithm (GA) (Davis et al., 1991) that uses the K Nearest Neighbours (kNN) to evaluate solutions. Li et al. (2001) also used a similar GA that employs the kNN classifier's ranking to evaluate solutions, and perform crossover as well, but focuses on the determination of convenient parameters to achieve a desirable sensitivity instead of relying on a local-search. Ooi and Tan (2003) used a similar GA to select features, but employs the Maximum Likelihood classifier to evaluate solutions instead of the kNN.

A few authors, however, still used sequential search approaches. Inza et al. (2004) employed search procedure called Sequential Forward Search (Chan et al., 1999) using the IB1, NB, C4.5 and CN2 classifiers to evaluate the solutions. They also compare their method with a filter variation for Leukemia and Colon Cancer datasets. Xiong et al. (2001) also used a SFS

algorithm, and a variation, called Sequential Forward Floating Search ((Chan et al., 1999)). To evaluate the solutions, both algorithms use either Fisher's Linear Discriminator Analysis (LDA), Logistic Regression or SVM classifiers.

Blanco et al. (2004), in turn, developed Estimation of Distribution algorithms that use the Naive Bayes classifier to evaluate feature sets to select features for cancer classification. These algorithms resemble GAs but instead of building populations using "genetic" principles, they are build by sampling from an estimated probabilistic distribution.

2.2 Embedded Methods

Some of the Embedded methods found on the literature exploit the inherent ability of the classifiers to discard features, like that of Random Forest, to infer a subset of discriminative features. Díaz-Uriarte and Andrés (2006) iteratively fit random forests, at each iteration building a new forest after discarding the least important genes. Jiang et al. (2004) developed an algorithm that follows the same concept and compare it to a variation that uses Fisher's LDA instead of Random Forests.

Other authors use the rankings of univariate methods as weights to guide their search. Guyon et al. (2002), for example, used SVM rankings to guide a search method based on Recursive Feature Elimination. Ma and Huang (2005) used Logistic Regression weights to allow a Receiver Operator Characteristic (ROC) technique to be used for large scale genomic data.

2.3 Filter Methods

Unlike wrapper and embedded methods, filter methods only general characteristic of the data to evaluate the worth of the features (or set of features).

2.3.1 Univariate Methods

Univariate filter methods rank features individually, by assigning a value to each. The simplest of them, the Threshold Number of Misclassifications (TNoM), detailed in Ben-Dor et al. (2000),

included setting a threshold on the observed fold-change differences within the features between the states under study, and the detection of the threshold point in each gene that minimizes the number of training sample misclassification. In other words, algorithms based on the TNoM basically rank all the features and decide what is the minimum associated worth necessary for a feature to be selected.

2.3.1.1 Univariate Parametric Methods

Jafari and Azuaje (2006) proposed two parametric variations of the TNoM, the sample t-test and ANOVA, that gave birth to Bayesian Frameworks that adapt the t-tests and statistical parameters to deal with the small number of samples, such as those proposed by Baldi and Long (2001); Fox and Dimmic (2006). Although the Gaussian models are the most popular, Thomas et al. (2001) and Newton et al. (2001) proposed the use of regression and Gamma models, respectively.

2.3.1.2 Univariate Model-Free Methods

When no distribution assumption can be made, authors often use more general approaches, such as the Wilcoxon Rank Sum, proposed by Thomas et al. (2001), BSS/WSS, proposed by Dudoit et al. (2003) and Rank Products, proposed by Breitling et al. (2004). Another popular and successful approach uses random permutations to estimate the reference distribution assumed by parametric approaches, leading to model-free alternatives. The works of Efron et al. (2001); Goss Tuscher et al. (2001); Pan (2003); Park et al. (2001) are good examples of such algorithms.

2.3.2 Multivariate Methods

Multivariate methods take filter methods a step further by considering groups of features instead of individual features. Therefore, these methods basically define a multivariate fitness function (a fitness function that assign a value to a certain group of features) and decide which combinations of features to evaluate, the fitness function being their main differential. Often as many combinations of features as possible are considered for evaluation, but due to

the curse of dimensionality, many methods choose to restrict or fix the sizes of the groups of features.

The simplest multivariate approaches, like that introduced by Bo and Jonassen (2002), exploit only bivariate interactions while others exploit more complex, higher order, ones such as Correlation based Feature Selection, proposed by Hall (1999) and later used by Wang et al. (2005); Yeoh et al. (2002), and variants of the Markov Blanket Filter like those proposed by Gevaert et al. (2006); Mamitsuka (2006); Xing et al. (2001). More complex correlations are also exploited in methods like the Minimum-Redundancy Maximum-Relevance (MRMR), proposed by Ding and Peng (2005), and Yeung and Bumgarner (2003)'s Uncorrelated Shrunken Centroid (USC). Finally, the (α, β)-k-Feature Set problem has also been successfully used in several practical applications to select features with different approaches (see Chapter 3). It can be seen as a natural step forward from Ben-Dor et al. (2000), making it fully multivariate by analysing groups of thresholds, instead of only individual ones. The fact that it is fully multivariate, and the ability to control the desired robustness of the solution and number of features to be selected make methods based on this problem particularly promising. However, the exponential characteristic of algorithms involved to solve it limited the applications so far. This work investigates ways to overcome this limitation using heuristics.

2.4 Further Reading

Even though the most relevant works for this studies have been described in this chapter, there are other methods that were sometimes too focused on specific applications, or that relied on assumptions that did not hold for the target problems. For a survey and comparison of feature selection methods more focused on ranking, which include most older and popular methods, refer to Molina et al. (2002). For a survey on feature selection methods focused on classification, refer to Dash and Liu (1997). Their paper details several greedy feature selection procedures, many of them also included in Molina et al. (2002). They also present an earlier attempt of categorization of feature selection algorithms. For a more recent review and categorization of feature selection methods, focused in bioinformatics, refer to Yvan Saeys

and Larrañaga (2007) and this chapter.

Chapter 3

The (α, β)-k-Feature Set Problem Approach

In 2004, Cotta et al. (2004) proposed a feature selection approach based in combinatorial optimization. The proposed problem, called (α, β)-k-Feature Set Problem, is a generalization of the k-Feature Set Problem, which is proven NP-Hard (see Cotta et al., 2004; Davies and Russell, 1994) and W[2]-Hard (see Cotta and Moscato, 2003a). Thus it is also NP-Hard and W[2]-Hard.

Optimization approaches that rely on **NP-Complete** problems with instances that can be solved within reasonable time in practice have been successfully used before to solve several problems, and have been shown to be of great importance. Cotta and Moscato (2003b), for example, proposed a hierarchical clustering method based on the Minimum Weight Hamiltonian Path Problem. Since this problem is NP-Hard, and exact methods were not able to solve practical instances, they addressed it using a Memetic Algorithm. Busygin et al. (2005), used Integer Programming to deal with the feature selection problem for biclustering purposes. Umpai and Aitken (2005) also solved an optimization problem to deal with the feature selection problem, focused on classification, but using a heuristic approach. Rizzi et al. (2010) proposed a hierarchical clustering algorithm based on the arithmetic-harmonic cut. The problem of finding such a cut is NP-Hard, but fixed parameter tractable. Therefore, they proposed a Memetic Algorithm to address it. Mellor et al. (2010) proposed an approach to find multi-

drug therapeutic combinations based on a more general version of the Hitting Set Problem. This problem is also NP-Hard but fixed-parameter tractable, which allowed them to solve practical instances for low values of the parameters defined, using safe reduction procedures.

This chapter formalises the underlying problems and details how one of the possible approaches that are based on them determines the required parameters. A summary of all notation used throughout this thesis is found in Appendix A

3.1 The Problem Formulation

The (α, β)-k-Feature Set Problem attempts to determine if there exists a set of k features that explain the dichotomy within the samples, maximizing the similarities between samples of the same class and the differences between the samples of different classes. A typical instance of this problem, with m samples and n features consists of a discrete valued matrix M and an array C. The $m \times n$ matrix M holds associated values $m_{p,f}$ of each sample p for each feature f. The array C of size m holds the class c_p of each sample p. The problem is defined with three positive integer parameters α and β and k. The value of α represents the minimum number of features that must explain the differences between any pair of samples of different classes. The value of β represents the minimum number of features that must explain the similarities between any pair of samples of the same class. Finally, as already mentioned, k represents the number of features to be selected.

Notice that the required data for this problem must be discrete and the values found on many datasets on the literature are often real numbers. For this purpose, Fayyad and Irani (1993)'s algorithm (see Appendix B) can used to discretize the information.

3.1.1 The Graph Representation

To formulate the problem mathematically it is convenient to represent this problem as a graph. In fact, since it is possible to reduce the k-Feature Set Problem to the Red-Blue Dominating Set Problem (see Cotta et al., 2004), the (α, β)-k-Feature Set Problem can also be thought as a problem in graphs in the same way, generalizing the problem.

Consider a bipartite graph $G = (\mathcal{U}, \mathcal{F}, \mathcal{E})$ where the set of vertices $\mathcal{F} = \{f_1, .., f_n\}$ contains one vertex for each feature and the set of vertices $\mathcal{U}$ contains one vertex $u_{p,q}$ for each pair of samples p and q, $p \neq q$, such that $\mathcal{U}$ can be partitioned in two disjoint subsets $\mathcal{A}$ and $\mathcal{B}$: if $c_p \neq c_q$, then $u_{p,q} \in \mathcal{A}$ and $u_{p,q} \in \mathcal{B}$ otherwise. In this graph, the set of all edges $\mathcal{E}$ contains an edge $e = (u_{p,q}, s_f)$ connecting vertices $u_{p,q} \in \mathcal{U}$ and $s_f \in \mathcal{F}$ if $u_{p,q} \in \mathcal{A}$ and $m_{p,f} \neq m_{q,f}$ or if $u_{p,q} \in \mathcal{B}$ and $m_{p,f} = m_{q,f}$.

Figure 3.1 shows an example, extracted from Gómez Ravetti et al. (2009), of a (α, β)-k-Feature Set Problem instance (3.1a) and its respective graph representation (3.1b).

Figure 3.2 formulates the (α, β)-k-Feature Set Problem as an Integer Program with binary variables.

3.2 The Feature Selection Approach

In Figure 3.2, the (α, β)-k-Feature Set Problem is presented as Integer Program with binary variables. Any feature set that satisfies constraints (3.2a)-(3.2c) is a feasible solution of the (α, β)-k-Feature Set Problem. There are several methodologies to define the required arguments. Rocha de Paula et al. (2011) set the required number of selected features based on experience of previous works, that is, values that worked well on previous studies, such as Ray et al. (2007) and Gómez Ravetti and Moscato (2008); and the rest of the parameters according to the methodology that will be described in this section. Gómez Ravetti et al. (2009) use an approach that attempts to favour the coverage of the nodes either on set $\mathcal{A}$ or $\mathcal{B}$.

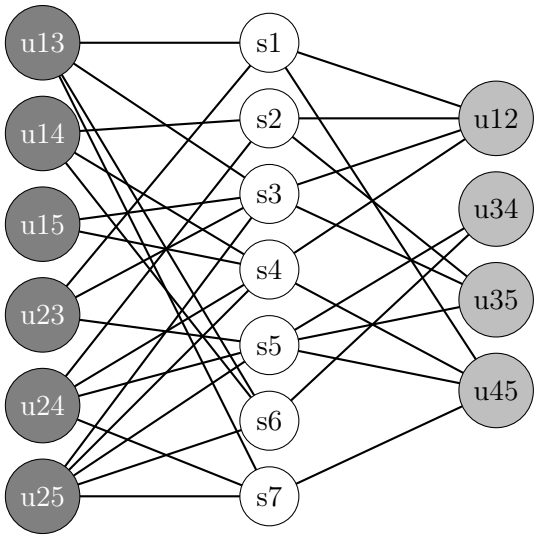
This work adopts the same methodology used in Berretta et al. (2008), where one attempts to set α as big as possible, k as small as possible and β also as big as possible, in this order of importance. This approach was designed to select a small number of features in the biological context that best represents the whole set. More specifically, the rationale behind this approach is to find a small subset of features that can be used to distinguish between two classes of samples. The features could model proteins, genes or SNPs, for example. The samples could be people who have a certain characteristic or not. Intuitively, in this case it is reasonable to say that such a subset must hold as much information as possible about the differences

Figure 3.1: An (α, β)-k-Feature Set Problem instance example, extracted from Gómez Ravetti et al. (2009), and its graph representation.

(a) An instance example of the (α, β)-k-Feature Set Problem.

Sample		1	2	3	4	5
Feature	1	0	0	1	0	0
	2	1	1	1	0	1
	3	1	1	0	1	0
	4	1	1	1	0	0
	5	0	1	0	0	0
	6	0	1	1	1	0
	7	0	1	1	0	0
Class		1	1	2	2	2

(b) Graph representation of 3.1a. The dark gray vertices are those in set $\mathcal{A}$, the light gray vertices are those in set $\mathcal{B}$ and the white vertices are those in set $\mathcal{F}$.



■

Figure 3.2: Mathematical formulation of the (α, β)-k-Feature Set Problem. Note that all input data must be defined beforehand. Different methodologies for that define different feature selection approaches. One such approach, elaborated throughout this text, is detailed on Section 3.2. Any feature set that satisfies these constraints is a feasible solution of the problem.

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \alpha \qquad \forall u \in \mathcal{A} \tag{3.2a}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \beta \qquad \forall u \in \mathcal{B} \tag{3.2b}$$

$$\sum_{f \in \mathcal{F}} x_f = k \tag{3.2c}$$

$$x_f \in \{0,1\} \forall f \in \mathcal{F} \tag{3.2d}$$

Data:
α: the number of selected features that must cover every vertex in $\mathcal{A}$.
β: the number of selected features that must cover every vertex in $\mathcal{B}$.
k: the number of features that must be selected.

Decision Variables:
x_f: 1 if feature f is selected, and 0 otherwise.

Constraints:
(3.2a): at least α selected features have to cover every vertex in $\mathcal{A}$.
(3.2b): at least β selected features have to cover every vertex in $\mathcal{B}$.
(3.2c): there must be exactly k selected features.

■

between the samples, as that is the most useful and important information. However, it would be interesting that such a subset is not to be considered too big, both because of technological reasons related to the particular applications, such as the amount of proteins that would fit in a microarray chip, for example; and to facilitate the interpretation of the results. Also, it would be interesting that such a subset had good internal consistency, that is, it should have as much evidence as possible that samples in the same group actually should be in the same group. Finally, if there are more than one solution that satisfy all that, the most interesting would be the one that holds more information. That naturally suggests the aforementioned order of importance of the problem's parameters, for which the four-stage approach depicted on Figure 3.3 was proposed. In this methodology, the variables are set sequentially, by solving a subproblem to define each parameter and fixing the obtained value before solving the next.

It is worth noticing that four optimization problems are solved, and the selected features in each of them may vary significantly. However, only the feature set obtained on the last step is used, as it is considered the most robust and useful in practice.

It is also important to notice that, with exception of the first (of determining α^*), all optimization problems solved are potentially hard, and thus may take a long time to be solved and require a significant amount of resources, depending on the instance. Indeed, since the problem of finding k^* is modelled as the unicost Min Set Multi Cover problem (see Section 3.3), it is NP-Hard. The next two problems, of finding β^* and the maximum cover, aim to find, within optimal solutions of the Min Set Multi Cover, at least one with a maximal structure, that is, that maximises also β and the total covering in that order of priority. Even though at the moment we have no formal proof of difficulty (if it is NP-Complete), solving them could be rather costly, as it would involve enumerating such solutions, and finding one such solution is often hard enough.

It is also worth noticing that, while the first and second stages of the approach model actual problem requirements: maximum distinction with minimal size, in order of importance. The two last stages, however, model desirable characteristics to be found on the solutions, also in order of importance.

Figure 3.3: The four-stage approach used to determine the (α, β)-k-Feature Set Problem parameters, α^*, β^* and k^*.

1. Determine α^*, the maximum value of α such that there exists a feasible solution for the $(\alpha, \beta) - k$-Feature Set Problem. This value can be obtained in polynomial time either by solving the model depicted on Figure 3.4 or by running Algorithm 1.

2. Determine k^*, the minimum number of features necessary to explain the dichotomy between the classes, considering that at least α^* features do so for each pair of samples. This can be done by solving the problem depicted on Figure 3.5. In this model, α is set to α^*, the value found on the previous step.

3. Determine β^*, the maximum value of β such that exactly k^* features are selected to explain the dichotomy between the classes and at least α^* features do so for each pair of samples. This can be done by solving the problem modelled on Figure 3.6, with α and k set to α^* and k^*, respectively, as obtained on the first two steps. This step maximizes the internal consistency of the samples in the same class, providing an even more robust feature selection.

4. Finally, find the k^* features that provide more explanations in total, either to the dichotomy between the classes or similarity within samples in the same class, considering the optimal parameters α^*, β^* and k^* for the $(\alpha, \beta) - k$-Feature Set Problem depicted on Figure 3.2. This is done by solving the optimization problem depicted by Figure 3.7, setting the values of α, β and k to the ones obtained on the previous steps: α^*, β^* and k^*, respectively.

■

Algorithm 1: Polynomial algorithm to solve the Max α (α, β)-k-Feature Set Problem. This problem can be trivially solved to optimality in polynomial time simply by counting the degrees of each vertex in $\mathcal{A}$. The optimum objective function value is the smallest such degree because, if any bigger value was assumed, rows (3.4b) of Figure 3.4 relative to nodes with a smaller degree would be infeasible. Similarly, if a smaller value was assumed, a better solution would be easily found simply by incrementing this value, which would not make any rows (3.4b) infeasible. This algorithm runs in $O(|\mathcal{A}|)$.

```
  /* Let M be a sufficiently large number                    */
1 min ← M;
2 foreach v ∈ A do
3     if |N(v)| < min then min ← |N(v)|;
4 return min
```

Figure 3.4: Mathematical formulation of the Max α (α, β)-k-Feature Set Problem. This problem aims to find the biggest minimum covering requirement for the pairs of samples on set $\mathcal{A}$ such that the resulting Set MultiCovering Problem solved on the next stage of the $(\alpha, \beta) - k$ Feature Set Approach still has a feasible solution.

$$\max \alpha \tag{3.4a}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f - \alpha \geq 0 \qquad \forall u \in \mathcal{A} \tag{3.4b}$$

$$x_f \in \{0,1\} \forall f \in \mathcal{F} \tag{3.4c}$$

$$\alpha \in \mathbb{Z}^+ \tag{3.4d}$$

Decision Variables:
x_f: 1 if feature f is selected, and 0 otherwise.
α: a positive integer value.

Objective Function and Constraints:
(3.4a): The objective is to maximise the value of α.
(3.4b): at least α selected features have to cover every vertex in $\mathcal{A}$.

■

Figure 3.5: Mathematical formulation of the Min k (α, β)-k-Feature Set Problem with $\beta = 0$. This problem aims to find the minimum set of features that satisfy the covering requirements of the sample pairs on set $\mathcal{A}$. It is the Set MultiCovering Problem found on the literature, with unitary costs and the same covering requirements for every row. Being a generalization of the Set Covering Problem, this problem is also proven NP-Hard.

$$\min \sum_{f \in \mathcal{F}} x_f \tag{3.5a}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \alpha \qquad \forall u \in \mathcal{A} \tag{3.5b}$$

$$x_f \in \{0,1\} \forall f \in \mathcal{F} \tag{3.5c}$$

Data:
α: the minimum number of selected features that must cover every vertex in $\mathcal{A}$. The maximum setting for this is obtained by solving the model shown on Figure 3.4.

Decision Variables:
x_f: 1 if feature f is selected, and 0 otherwise.

Objective Function and Constraints:
(3.5a): The objective is to minimise the number of selected features.
(3.5b): at least α selected features have to cover every vertex in $\mathcal{A}$.

■

Figure 3.6: Mathematical formulation of the Max β (α, β)-k-Feature Set Problem. This problem aims to find the biggest minimum covering requirement for the sample pairs in set $\mathcal{B}$ such that it is still possible to obtain a feature set of size k that covers the sample pairs in set $\mathcal{A}$ at least α times. Note that, even though this problem is very similar to the one depicted on Figure 3.4, row (3.6d) makes it significantly harder because $\mathcal{A}$ and its connected edges are not a mirror of $\mathcal{B}$ and its connected edges. That means that one might still need more than k features to satisfy rows (3.6b) and (3.6c) at the same time, which could make row (3.6d) infeasible.

$$\max \beta \tag{3.6a}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \alpha \qquad \forall u \in \mathcal{A} \tag{3.6b}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f - \beta \geq 0 \qquad \forall u \in \mathcal{B} \tag{3.6c}$$

$$\sum_{f \in \mathcal{F}} x_f = k \tag{3.6d}$$

$$x_f \in \{0,1\} \forall f \in \mathcal{F} \tag{3.6e}$$

$$\beta \in \mathbb{Z}^+ \tag{3.6f}$$

Data:
α: the minimum number of selected features that must cover every vertex in $\mathcal{A}$. The maximum setting for this is obtained by solving the model shown on Figure 3.4.
k: the number of features that must be selected. The minimum setting for this, considering the specified value of α, is obtained by solving the model shown on Figure 3.5.

Decision Variables:
x_f: 1 if feature f is selected, and 0 otherwise.
β: a positive integer value.

Objective Function and Constraints:
(3.6a): The objective is to maximise the value of β.
(3.6b): at least α selected features have to cover every vertex in $\mathcal{A}$.
(3.6c): at least β selected features have to cover every vertex in $\mathcal{B}$.
(3.6d): there must be exactly k selected features.

■

Figure 3.7: Mathematical formulation of the Max Cover (α, β)-k-Feature Set Problem. This problem aims to find the feature set that yield the biggest covering among the optimal solutions of the previous stage of the $(\alpha, \beta) - k$ Feature Set approach. That is, it attempts to find a solution that is optimal for all previous stages of the approach, that provides the best explanation for the tableau under consideration.

$$\max \sum_{f \in \mathcal{F}} w_f x_f \tag{3.7a}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \alpha \qquad \forall u \in \mathcal{A} \tag{3.7b}$$

$$\sum_{f \in \mathcal{F}: \exists e=(f,u) \in \mathcal{E}} x_f \geq \beta \qquad \forall u \in \mathcal{B} \tag{3.7c}$$

$$\sum_{f \in \mathcal{F}} x_f = k \tag{3.7d}$$

$$x_f \in \{0,1\} \forall f \in \mathcal{F} \tag{3.7e}$$

Data:
α: the minimum number of selected features that must cover every vertex in $\mathcal{A}$. The maximum setting for this is obtained by solving the model shown on Figure 3.4.
k: the number of features that must be selected. The minimum setting for this, considering the specified value of α, is obtained by solving the model shown on Figure 3.5.
β: the minimum number of selected features that must cover every vertex in $\mathcal{B}$. The maximal setting for this, considering the specified values of α and k, is obtained by solving the model shown on Figure 3.6.
w_f: the importance (weight) of feature f. Throughout this work, w_i is the degree of $f \in \mathcal{F}$.

Decision Variables:
x_f: 1 if feature f is selected, and 0 otherwise.

Objective Function and Constraints:
(3.7a): The objective is to maximise the covering of the pairs of samples.
(3.7b): at least α selected features have to cover every vertex in $\mathcal{A}$.
(3.7c): at least β selected features have to cover every vertex in $\mathcal{B}$.
(3.7d): there must be exactly k selected features.

■

3.3 Set (Multi) Cover Problems

The (α, β)-k-Feature Set problem belongs to a larger class of problems called Covering Problems, of which the most prominent examples are the Set Cover Problems. Indeed, only row (3.2c) on Figure 3.2 is not a covering constraint.

In these problems, there is a set of elements, called universe, and a set of subsets of these elements such that the union of all the subsets of elements is the universe. A covering is a subset of all available subsets of elements, whose union still contains the whole universe. The Minimum Set Cover is the minimum number of subsets necessary to compose a covering. The Minimum Set Cover Problem is NP-Hard (Garey and Johnson, 1979), and one of Karp (1972)'s 21 NP-Complete problems.

Consider a vector $\mathbf{c} \in \mathbb{Z}^+$ and a binary matrix $\mathbf{A}$. The Set Cover Problem can be formulated as the integer program defined in (3.8).

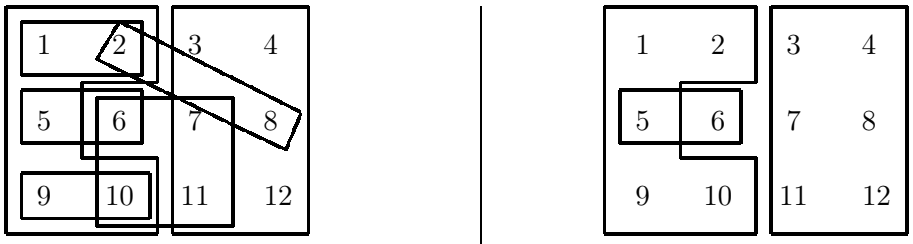
$$\min\{z = \mathbf{cx} : \mathbf{Ax} \geq \mathbf{1}, \mathbf{x} \in \{0,1\}\} \tag{3.8}$$

Figure 3.8 illustrates the problem with an example. In the representation shown in Figure 3.8a, a feasible covering requires that every element of the universe (the numbers) have to be in at least one of the selected boxes (the available subsets). In Figure 3.8b, every edge represents entry in A where the value is one. The dark nodes represent the selected subsets, and a full edge represent a specific element of the universe being covered by a specific subset. Any feasible covering would thus require every element of the universe to be connected by at least one full edge.

A generalization of this problem, the Set Multi Cover Problem, introduces redundancy, by allowing the covering requirement of the elements of the universe to be greater than one. Let $\mathbf{b} \in \mathbb{Z}^+$ be a vector containing how many times each element must be covered. The Set Multi Cover Problem can be formulated as the integer program defined in (3.9). In this problem, every element i of the universe would be required to be connected by at least $b_i \geq 1$ full edges

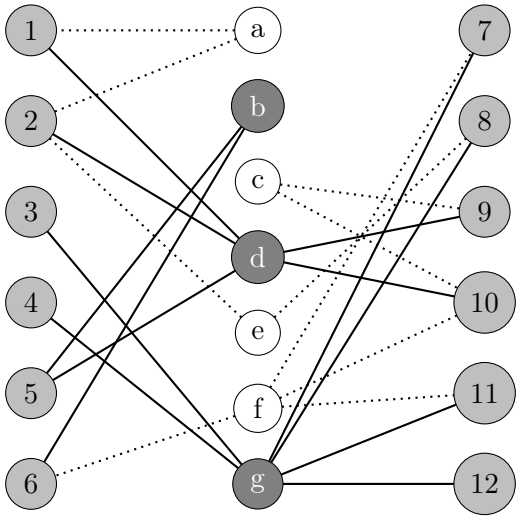
Figure 3.8: Example of a Min Set Cover. Given a universe $\mathcal{U} = \{1..12\}$ and the set of available subsets $\mathcal{F} = \{a = \{1,2\}, b = \{5,6\}, c = \{9,10\}, d = \{1,2,5,9,10\}, e = \{2,8\}, f = \{6,7,10,11\}, g = \{3,4,7,8,11,12\}\}$, a minimum set cover (of size 3) would be achieved by subsets $\mathcal{C} = \{b, d, g\}$.

(a) Set Representation



Input: Universe and a set of subsets | Output: minimum size (3) cover

(b) Graph Representation: the darkest nodes represent the minimum size (3) cover. The full edges illustrate a minimum set cover.



■

in Figure 3.8b, and at least b_i boxes in Figure 3.8a.

$$\min\{z = \mathbf{cx} : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \in \{0,1\}\} \tag{3.9}$$

It is easy to see that the Min k $(\alpha, \beta) - k$-Feature Set problem, as formulated in Figure 3.5, is modelled as a Set Multi Cover Problem, setting $\mathbf{c} = \mathbf{1}$ and $\mathbf{b} = \boldsymbol{\alpha}$. The features can be modelled as the available subsets of the elements of the universe which, in turn, can be modelled as the pairs of samples.

Since the problem depicted on Figure 3.4 is always solved to optimality, obtaining a good solution for the one formulated on Figure 3.5 is the most important step of the (α, β)-k-Feature Set approach. The next two problems, depicted by Figure 3.6 and Figure 3.7 only seek an optimal solution of the one depicted on Figure 3.5 with optimal structure: maximal beta and maximal cover.

Even though the Set Multi Cover problem only models the second stage of the $(\alpha, \beta) - k$-Feature Set approach, it can be very resourceful to gain insight into new solution methods for the whole procedure. To the best knowledge of the author at the time of this research, the only published approach to this problem is the exact approach proposed by Cotta et al. (2004). Therefore this section reviews some of the most important approaches found on the literature for the Set Cover and Multi Cover Problems.

Notice, however, that since the $(\alpha, \beta) - k$-Feature Set approach requires an optimal structure on the optimal solution, the complete approach could be - and often is - significantly harder in practice than either Set Cover or Multi Cover alone. Also, since the last two stages consider also a second, potentially large, set of elements in its universe, the full problems under consideration are also much larger than the Set Multi Cover (sub)problem solved at the second stage of the approach. Therefore, even though sample problem sizes and running times for Set (Multi) Cover problems are reported in this section, whenever available, these are merely to give the reader a better idea of the limitations of state-or-the art methods that could be adapted for the objectives pursued in this work, as these measurements are not comparable to those of any $(\alpha, \beta) - k$-Feature Set approach algorithms.

3.3.1 Existing Set Cover Approaches

One of the first methods in the field is the greedy algorithm proposed by Chvatal (1979) which, at each iteration, simply selects the subset that maximizes the ratio between its degree and its weight. This algorithm gives an approximation ratio of $\log(n)$, n being the number of features. It is interesting to notice that Lund and Yannakakis (1994) showed that the Set Cover Problem cannot be approximated by polynomial algorithms within a ratio of $\log(n/4)$ under assumptions that are considered stronger than that $P \neq NP$ (see Feige, 1998). Therefore, it is reasonable to say that it is unlikely that Chvatal's approximation ratio can be significantly improved using a greedy algorithm.

In attempt to overcome this difficulty, a non-deterministic version of Chvatal's algorithm was then developed by Feo and Resende (1989). In this setting, instead of deterministically choosing the subsets, it restricts the choice to a subset of interesting unselected features and chooses randomly between them. Also, since it has a random component, and thus different results can be obtained from different runs, it is ran several times and the best solution is considered.

Hassin and Levin (2006) developed a version of Chvatal's greedy algorithm that is able to regret its decisions. In this modified version, the algorithm is allowed to withdraw a some selected subsets from the solution, provided that the newly selected ones cover all the items that were covered by the withdrawn. Using this restricted backtracking strategy they were able to obtain a better approximation ratio than that of the original algorithm.

A natural extension of greedy and probabilistic algorithms, is the use of meta-heuristics. Lan et al. (2007) developed a procedure that uses a modified version of Chvatal's greedy heuristic, and a local search procedure within a Meta-RaPS framework. The improving heuristic randomly removes a few subsets from a solution, which makes it infeasible for not covering all the elements, and then solves the reduced size problem composed of only the uncovered elements to make it feasible again. The resulting method was able to obtain near-optimal solutions for problems with up to 1000 elements and 10.000 subsets within reasonable time.

Jacobs and Brusco (1994) proposed a Simulated Annealing-based heuristic for the problem that was also able to solve up to 1000 elements and 10.000 subsets.

Another meta-heuristic attempt to address the problem was made by Beasley and Chu (1996), with a Genetic Algorithm. This algorithm was also able to obtain near optimal solutions for instances with up to 1000 elements and 10000 subsets within reasonable time. Solar et al. (2002) proposed a parallel version of a similar algorithm that allowed several searches of the solution space to occur simultaneously. This algorithm was able to obtain near optimal solutions for instances with up to 500 elements and 5000 subsets within reasonable time.

For more information about meta-heuristics, please refer to Glover and Kochenberger (2003).

Lovasz (1975) showed that the ratio of optimal integral and fractional covers of a hypergraph does not exceed $1 + \log d$, where d is the maximum degree. Therefore, the integrality gap of the linear relaxation of the integer programming formulation of the Set Cover Problem is of $O(\log n)$, which is the approximation rate of the Chvatal (1979)'s algorithm.

Bertsimas and Vohra (1998) and Kolliopoulos and Young (2005) proposed algorithms that used both linear and non-linear functions to round the fractional coefficients of solutions obtained by solving the linear relaxation of the problem. Their focus was on theoretical results, and practical computational results were not listed.

Following the same paradigm of cutting planes and subgradient optimization once proposed by Balas and Ho (1980), Beasley (1990a) proposed a Lagrangean heuristic which, combined with other enhancements of the method proposed by Beasley (1987) (see Beasley and Jørnsten, 1992), culminated in an exact approach able to solve to optimality instances with up to 400 elements and 4000 subsets. The Lagrangean heuristic, in particular was also able to obtain near optimal solutions, better than those obtained by a number of other heuristics, for instances with up to 1000 elements and 10000 subsets. Balas and Carrera (1996) proposed a similar exact approach, using a Branch-and-Bound framework. Lorena et al. (1994) used a similar heuristic approach, using a Surrogate relaxation of the problem, which is able to solve instances also with up to 1000 elements and 12.000 subsets within reasonable time.

Ceria et al. (1998) and Caprara et al. (1999) also proposed Lagrangean heuristics able to solve instances with up to 4000 elements and 1.000.000 subsets. The latter seemed to outperform or match the former, Beasley's, Balas and Carrera and Lorena et al.'s algorithms.

Caserta (2007) used a Tabu Search procedure as a center piece for a hyper heuristic consisted of a Primal-Dual subgradient optimization based on Ceria et al. (1998)'s approach. Significantly more complex, their procedure benefited from the introduced random components to slightly improve some of the gaps and bounds for the largest railway crew scheduling instances, and matched the results on the others.

Even though not all cited papers provide time measurements, throughout this text, a "reasonable computational time" is considered to be of the order of seconds or minutes for easy instances up to a couple of hours (up to 5000 seconds) for the hardest ones.

3.3.2 Existing Set Multi Cover Approaches

Less works on the literature deal with the Set Multi Cover Problem, and they are mostly generalizations of Set Cover algorithms.

One of the first approaches, proposed by Hall and Hochbaum (1986), uses an approximative algorithm to address the problem. Their algorithm also provides a dual feasible solution that is used to show that the ratio between the optimum and the solution obtained in the worst case does not exceed the maximum row sum of the coefficient matrix A of Equation 3.9.

Generalizing and extending the ideas of Balas and Ho (1980); Beasley (1990a); Ceria et al. (1998), Rajagopalan and Vazirani (1998) followed the classic primal dual scheme to propose parallel Primal-dual RNC approximation algorithms for (Multi) Set (Multi) Cover Problems.

Berman et al. (2007) proposed a randomized approximative algorithm for the problem, deriving estimates for the approximation ratio as the covering requirement varies, and improving the standard greedy algorithm's ratio of $O(\log n)$.

To the best knowledge of the author, up to the time of this research, the fastest approach to solve the Set Multi Cover problem (to optimatility) is the one proposed by Hua et al. (2009, 2010). Their algorithms are able to solve the Set Multi Cover problem in $O((b+2)^n)$ time using dynamic programming, where b is the maximum coverage requirement and n is the number of elements in the universe.

Finally, Pessoa et al. (2010, 2011) used the ideas of Balas and Ho (1980); Beasley (1990a); Ceria et al. (1998) within a GRASP (Feo and Resende, 1995) framework, obtaining a better

tradeoff between solution quality and running times, specially for large instances.

3.4 Safe Reduction Rules

Since the (α, β)-k-Feature Set Problem is NP-Complete (see Cotta and Moscato, 2003a; Davies and Russell, 1994), large instances quickly become an issue.

Data reduction is a key technique in the study of fixed parameter algorithms. For many intractable problems, reducing the instance to a *problem kernel* in the parameterized setting (see (Downey and Fellows, 1999, see)) can make it possible to be solved within reasonable time in practice. That is done by pre-processing the instance in a way that the data size is reduced and any solution of the problem kernel is necessarily also a solution of the original problem. When such method is used in a way that no optimal solution is lost, this is referred as a *safe reduction*.

Balas and Carrera (1996); Balas and Ho (1980); Beasley and Jørnsten (1992); Beasley (1987, 1990a) proposed many safe reduction rules for the Set Cover Problems, and in general they are crucial for the success of their, and others' like Caprara et al. (1999); Ceria et al. (1998); Lorena et al. (1994), approaches for large problems. Many commercial mathematical programming packages like CPLEX[1] and GUROBI[2] are also usually very good with redundancy and dominance tests, which comprise many of the proposed safe reductions. Therefore, even simply using one of these tools to solve the Integer Programming models could provide very useful results.

Berretta et al. (2007); Cotta and Moscato (2003b); Moscato et al. (2005) extended some of the Set Cover safe reduction rules for the (α, β)-k-Feature Set Problem. These procedures, first proposed in Cotta et al. (2004), are detailed on Algorithm 2, Algorithm 3, Algorithm 4, Algorithm 5.

Let $N(u)$ be the set of neighbours of vertex $u \in \mathcal{U}$: $N(u) = s \in \mathcal{F} : \exists(u, s) \in \mathcal{E}$. Similarly, let $N(s)$ be the set of neighbours of vertex $s \in \mathcal{F}$: $N(s) = u \in \mathcal{U} : \exists(u, s) \in \mathcal{E}$. Let d_u be an integer associated with vertex $u \in \mathcal{U}$, initially set to the number of features that still have to

[1] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
[2] http://www.gurobi.com/

cover vertex u: α if $u \in \mathcal{A}$ and β if $u \in \mathcal{B}$.

Algorithm 2: (α, β)-k-Feature Set Problem Instance Reduction Rule 1: if a vertex u needs to be covered by at least d_u features, and it has only d_u neighbours, then all its neighbours must be in the solution, otherwise it will always be infeasible for not fully covering it.

```
1 foreach u ∈ U : d_u = |N(v)| do
2     foreach s ∈ N(s) do
3         add s to solution;
4     G ← G\{u};
```

Algorithm 3: (α, β)-k-Feature Set Problem Instance Reduction Rule 2: if a vertex needs no more covering, it no longer needs to be considered when choosing a feature to be selected.

```
1 foreach u ∈ U : d_u ≤ 0 do
2     G ← G\{u};
```

Algorithm 4: (α, β)-k-Feature Set Problem Instance Reduction Rule 3: if there are two vertices, one whose neighbourhood is a subset of the other's, and the one with the smaller set of neighbours has a larger current covering requirement than the other, then there is no need to consider the one with the larger set of neighbours, because sufficient features to cover it have to be selected to cover the one with the smaller set of neighbours.

```
1 foreach u, v ∈ U, u ≠ v : d_u ≥ r_v and N(u) ⊆ N(v) do
2     G ← G\{v};
```

3.5 Conclusions

Despite the NP-Completeness of the problem, this approach has been used to solve many practical instances within reasonable time. For example, Cotta et al. (2004) used this approach to select a subset of genes, in a dataset with different types of diffuse large B-cell lymphoma, that differentiate between two classes. In Moscato et al. (2005), this approach was used to predict the U.S. Presidency election results. Berretta et al. (2007) used it to select features on a dataset with different types of cancer (NCI60). Gómez Ravetti and Moscato (2008); Gómez Ravetti et al. (2009) used it to find biomarkers for Alzheimer's Disease and Prostate Cancer, respectively. Berretta et al. (2008) also used this approach to select features on an

Algorithm 5: (α, β)-k-Feature Set Problem Instance Reduction Rule 4: if there are two features, one whose set of neighbourhood is a subset of the other's, and for every neighbour of the one with the smaller set of neighbours it is not necessary to choose both to fully cover it, then it is not necessary to consider the feature with the smaller set of neighbours, because the other one already does everything it does, and possibly more.

1 **foreach** $f, g \in \mathcal{F} : N(f) \subset N(g)$ *where* $\forall u \in N(f)$ *with* $|N(u)| \geq d_u$ **do**
2 $\quad G \leftarrow G \setminus \{f\}$;

Alzheimer's disease dataset. Mendes et al. (2008) also used this method to identify molecular portraits for prostate tumours with different Gleason patterns. Rosso et al. (2009) employed the approach to select background brain electrical activity features that distinguish childhood absence epilepsy patients from controls. To uncover a network of transcription factors that potentially dysregulate several genes in Multiple Sclerosis, at least one of its disease subtypes, Riveros et al. (2010) used an approach that included both the Hitting Set and (α, β)k-Feature Set problems. Both these problems are NP-Hard, but the exact approaches available in the literature were able to solve their instances.

The main advantage of using this optimization based approach over ranking methods is that the features are evaluated as groups, instead of only considering their individual performance. That is particularly interesting in highly underdetermined systems, in which there are many solutions that explain a single scenario. It is often the case that two or more features are highly ranked but do not explain a scenario well together, either because they do not introduce new information to one another, or because they explain different and/or conflicting aspects of the scenario, for example. In such a situation, a set of features that hold complementary information might be much more useful, better characterising the tableau.

The main drawback of this approach is the complexity of the combinatorial problems involved. Since some of them are NP-Complete, it is unlikely that there exists an efficient method to solve them to optimality efficiently (in polynomial time). Even though the applications found on the literature are important and could be dealt with within reasonable time, that is not often the case, specially considering new datasets that have been emerging, such as the ones for genome wide association studies or interpretation of large microarray datasets, for example. Those may contain up to the order of millions of features and tens of thousands

of samples. To the best of the author's knowledge, at the moment of this research, the available tools to solve the (α, β)-k-Feature Set Problem approach can not solve problems of this magnitude.

Recent works, such as Rocha de Paula et al. (2011) and Arefin et al. (2011) for example, exploited feature combinations. That is, the set of features $\mathcal{F}$ is expanded in a similar manner as the universe $\mathcal{U}$ is created for case/control datasets (see Section 3.1.1). For each possible combination of two features, a "metafeature" is included. In these works, the considered values of every sample are defined as the difference between the original values. Arefin et al. (2011) used this methodology to create metafeatures on the Breast Cancer dataset found in Van De Vijver et al. (2002). In Rocha de Paula et al. (2011), we used this technique to model cell signalling imbalance in blood plasma by introducing a new metafeature for each possible difference of values in the original dataset. With the study of these metafeatures, it was possible to show that a specific pattern of cell signalling imbalance in blood plasma has valuable information to distinguish between non-diseased controls and samples with Alzheimer's disease. In both these works the dataset is enlarged with the introduction of a number of new features of the order of the number of features squared, which is a significant amount, considering that the methodology could be applied to a dataset with tens of thousands of features.

Given the big advantage brought by the multivariate characteristic of this method, its successful wide applicability and knowing that its only real known drawback is scalability, further research to overcome such a difficulty is appropriate. As discussed on Section 1.2, even though the optimal solution of the problem is always desirable, it often is not strictly necessary, as it is the case of various biological *tableaus*, for which all solutions still need further verification, analysis, interpretation and research. Therefore, this work aims to propose efficient heuristic tools to address the (α, β)-k-Feature Set Problem approach, providing good solutions quickly, even for larger, more realistic, datasets.

Chapter 4

The Target Instances

Due to its very practical nature, in the field of bioinformatics one is often interested in instances that model the most important *tableaus* to be processed or interpreted, as those are going to be found more often in practice. For the same reason, algorithms that perform best for such instances can be more interesting than the others. To design and benchmark such algorithms, one has to understand the aspects that characterise these target instances and know why they are important.

Also, it is important to bear in mind that *large* instances are not necessarily *difficult* in practice. If good solutions (and hopefully also the optimal solution) can be obtained through an efficient procedure on the instance, it can be regarded as easily solvable, even though high dimensional (i.e.: time or resource consuming). A large instance poses a different challenge, of resource availability and management, that is, storing and retrieving data from memory efficiently. Throughout this text, *difficult* instances are those which have inherent properties that make the choices needed to solve it harder in comparison to instances of the same size that lack such characteristics. Note also that this chapter distinguishes between instances of the same problem. More specifically, it is already proven under very strong assumptions (P=NP) that it is unlikely that there would exist an efficient algorithm to solve every instance of the $(\alpha, \beta) - k$-Feature Set problem to optimality. It could be the case, however, that the some methods work better (i.e.: have lower computation time and/or memory requirements) for specific classes of instances than the others.

Table 4.1: Exact results for real world instances. Every stage of the (α, β)-k-Feature Set approach was allowed one hour to run. Each line represents one real world instance, as detailed on Section 4.1. Each column shows the instance's objective function value (OF) and computational time in seconds (CT) for each of the stages of the (α, β)-k-Feature Set approach. Whenever the obtained solution is not optimal the optimality gap is presented in parenthesis.

	Max α		**Min k**		**Max β**		**Max Cover**	
Instance	**OF**	**CT (s)**	**OF**	**CT(s)**	**OF**	**CT(s)**	**OF**	**CT(s)**
DS	14	0.001	36	0.004	10	0.009	2016	0.005
ADMF	86	0.067	292	1.388	118	7.561	581608	4.446
PD1	4094	0.073	8675	1.786	3677	910.743	24935478	266.252
PD2	154	0.004	289	0.039	76	0.648	53196	0.222
PC	409	0.021	867	0.295	231	3051.378	7540529	40.122
SM	22	23.771	128	2073.598	34 (*547.06%*)	3600		

This chapter aims to detail and discuss the expected characteristics of the target instances, and attempts to propose suitable benchmark instances to be used throughout the research, defining degrees of difficulty that would help to highlight the advantages of the methods under consideration.

4.1 Real Case/Control Testbed

Real world instances come in many different sizes and with various characteristics. As mentioned before, many times the exact approach is able to solve the instance to optimality within reasonable time and sometimes it does not. Also, what is a reasonable time also depends on the application. If one needs to solve the same instance many times, or many variations of it, a computational time that could be considered reasonable but high may become impractical.

In this work, a pool of six real world instances is considered. They were selected to be reasonably different from each other, with respect to their inherent characteristics and size. The original datasets were discretized with the entropy filter proposed by Fayyad and Irani (1992, 1993) and ambiguous features were discarded.

Table 4.1 shows the running times for each of these instances. They were allowed one hour of computation time for each of the stages of the (α, β)-k-Feature Set approach. Each stage used the best values found for the previous ones within that time.

The resulting instances are referenced as follows:

DS is the dataset proposed by Lockstone et al. (2007). It contains 73 features, which are genes, and 15 samples with seven cases of Down Syndrome and eight controls. This instance is interesting for being small and easy (as the exact approach is able to solve it quickly), while characterizing biological instances well, as it has more features than samples. It should therefore be a good standard for an easy instance, in the sense used throughout this work: computational effort required to solve.

ADMF is the dataset used by Rocha de Paula et al. (2011), which was obtained by expanding the dataset proposed by Ray et al. (2007) with one new "metafeature" for each possible pairwise combination of features. This resulting dataset contains 683 features (and metafeatures), which are (pairs of) proteins, and 83 samples with 43 cases of Alzheimer's Disease and 40 controls. This instance was selected due the methodology used to create it: since pairwise combinations of features are used, this instance is expected to be very symmetrical. In other words, groups of metafeatures that share a feature have similar characteristics (i.e.: node degrees). Even though significantly bigger than DS, it is also relatively easy to solve, but required observable running times. Moreover, tt is interesting to notice the particular methodology that Rocha de Paula et al. (2011) applied on it, which required that many variations of this instance were solved. In this case, it would be very important that these instances are solved quickly.

PD1 is the dataset used by Scherzer et al. (2007). It contains 17099 features, which are genes, and 105 samples with 50 cases of Parkinsons's Disease and 55 controls. PD2 is the dataset used by Lesnick et al. (2007). It contains 1674 features, which are Single Nucleotide Polymorphisms (SNPs), and 25 samples with 16 cases of Parkinsons's Disease and 9 controls. These two instances were selected for being associated with the same pathology but with very different characteristics. While PD2 can be solved quickly, PD1 is more challenging, with more appreciable running times, probably mainly due to its size.

PC is the dataset used by Chandran et al. (2007). It contains 3556 features, which are genes, and 171 samples with 90 cases of Prostate Cancer and 81 controls. This instance was chosen due to its intermediate size.

SM is the dataset used by Charlesworth et al. (2010). It contains 525 features and 1219

samples with 922 individuals who smoke and 297 who do not. This dataset was chosen for differing from typical biological datasets, having more samples than features. Even though it is not the biggest under consideration, it is the most demanding: the exact approach can only solve the first two stages of the (α, β)-k-Feature Set Approach. The maximisation of β can not be solved to optimality, and consequently the solutions for the maximisation of the Total Covering may not be realistic due to the suboptimality of the value of β used.

4.2 (α, β)-k-Feature Set Problem Instances

Most of the instances used to benchmark the algorithms cited on the previous chapter belong to the OR-Library (Beasley, 1990b). They differ from each other with respect to their *size* mainly. At the time of this research, this set of available instances was composed of 87 files: 50 from Beasley (1987) and Balas and Ho (1980), 20 from Beasley (1990a), 10 from Grossman and Wool (1997) and 7 from Ceria et al. (1998). The last ones were the largest ones available, with 1000000 columns and 5000 rows.

Observing the graph representation of (α, β)-k-Feature Set Problem instances, it is easy to see its tight relationship with Set Multi Cover instances. They differ from each other mainly due to the fact that the (α, β)-k-Feature Set approach partitions the elements of the universe in two correlated groups that are considered differently throughout the methodology. In particular, if the partition of the elements of the universe is taken as a parameter, the set of possible (α, β)-k-Feature Set problem instances are a subset of possible Set Multi Cover instances. That is because of structural properties that will be detailed throughout this section, related to the way the graph representation is obtained.

Namely, the instances under consideration are obtained using the method depicted in Section 3.1.1. Therefore, two important things should be noted.

First, $|\mathcal{U}|$ is of the order of the number of available samples of the datasets squared ($O(m^2)$, or more precisely $\binom{m}{2}$). Therefore, for a sufficiently large number of samples, the instance could have $|\mathcal{U}| \ggg |\mathcal{F}|$. That differs from most instances available in the literature for Set Multi Cover Problems, which have the opposite case. This could have a considerable impact on the

obtained bounds and solution process by methods derived from integer programming (Caserta, 2007).

Next, the set of all pairs of samples is composed by two correlated disjoint sets $\mathcal{A}$ and $\mathcal{B}$ of available pairs of samples that belong to the different or the same class, respectively. That comes from the fact that both the existence or not of the nodes in sets $\mathcal{A} \cup \mathcal{B}$ must be consistent with the classes that the samples are assigned in the dataset. Similarly, the existence of edges must be consistent with the values in matrix M of the original dataset and the classes of the samples involved. In other words, although any table with features and samples can be translated into a tripartite graph that will serve as an instance to the (α, β)-k-Feature Set problem, not every tripartite graph can be translated into a table with features and samples. Therefore, although every biological case-control instance can be converted into a Set Multi Cover Problem instance, by considering the features as sets in $\mathcal{F}$ and pairs of samples the elements in $\mathcal{U}$; not every Set Multi Cover Problem instance can be converted into an (α, β)-k-Feature Set Problem instance even defining appropriate partitions of the universe $\mathcal{U}$.

In particular, for every combination of three samples, there are also three possible combinations of pairs of samples. Without loss of generality, consider three samples o, p and q and their resulting pairwise combinations, visited in lexicographic order, $u_{o,p}$, $u_{o,q}$ and $u_{p,q}$. Once the properties (i.e.: whether they belong to set $\mathcal{A}$ or $\mathcal{B}$ and whether they have an edge connecting them to a feature $s_f \in \mathcal{F}$) of $u_{o,p}$ and $u_{o,q}$ are known, the properties of $u_{p,q}$ would have to be consistent with the properties of the previous two sample pairs. That is because, since there are only two classes (cases and controls) and two possible values for the entries in matrix M (different values or the same value), the set definitions (if the sample pair belongs to $\mathcal{A}$ or $\mathcal{B}$) for the first two pairs of samples already define classes for the three samples under consideration, and thus define the set that the third sample pair belongs as well. If this third sample pair was in another set, the graph would not be translatable to a table with features and samples because a value in C would be ambiguous. Similarly, the presence or absence of edges between the two first sample pairs and a given feature $s_f \in \mathcal{F}$ already defines entries on matrix M (i.e.: if they are different or not) for all three samples for the same feature $s_f \in \mathcal{F}$.

In this case, if the presence or absence of an edge between the third pair of samples and a feature is not consistent with the presence/absence of edges between the other two sample pairs and the same feature, and assuming that the set definitions are consistent with the sample classes; the graph would not be translatable to a table with features and samples because a value in M would be ambiguous. See Figure C.1 and Figure C.2 for illustrations correct and of ambiguous cases, respectively.

For example, consider Figure C.1e. If $u_{o,p} \in \mathcal{A}$, then c_o and c_p must hold different values (say, 1 and 2, respectively) because $\mathcal{A}$ holds the sample pairs that involve samples that belong to different classes. Next, if $u_{o,q} \in \mathcal{B}$, then $c_q = 1$ because, from the previous assignment, $c_o = 1$ and $\mathcal{B}$ holds sample pairs that involve samples that belong to the same class. Therefore, $u_{p,q} \in \mathcal{A}$ because, from the two previous assignments, $c_p = 2$ and $c_q = 1$ and $\mathcal{A}$ holds the sample pairs that involve samples that belong to different classes. Similarly, given these class values, if there exists an edge $(u_{o,p}, s_f) \in \mathcal{E}$ then, since $u_{o,p} \in \mathcal{A}$, the value of o in the value matrix is different from the value p for feature $s_f \in \mathcal{F}$, say 0 and 1, respectively. Next, if there is not an edge $(u_{o,q}, s_f) \in \mathcal{E}$, because $u_{o,q} \in \mathcal{B}$ the value of q must be same as the value of o, which is 0. Finally, since the values of p and q are both 1 for feature $s_f \in \mathcal{F}$ and $u_{p,q} \in \mathcal{A}$, there must not exist and edge $(u_{p,q}, s_f) \in \mathcal{E}$.

The main impact of these characteristics is on the distribution of nodes between $\mathcal{A}$ and $\mathcal{B}$, and the distribution of edges connecting $\mathcal{U}$ to $\mathcal{F}$ among the vertices in $\mathcal{U}$. However, recent works, such as Rocha de Paula et al. (2011) and Arefin et al. (2011), exploit feature combinations. That is, the set of features $\mathcal{F}$ is expanded in a similar manner as the universe $\mathcal{U}$ is created for case/control datasets (see Section 3.1.1): with the introduction of "artificial" (meta)features, one for each combination of two or more features. Such a methodology also impacts the size of set $\mathcal{F}$ and the distribution of edges connecting $\mathcal{U}$ to $\mathcal{F}$ among the vertices in $\mathcal{F}$ in a similar manner.

These edge distributions, both of edges connected to $\mathcal{U}$ and $\mathcal{F}$ could have a significant impact on how the available methods perform on the instance under consideration. For example, the methodology we used in Rocha de Paula et al. (2011), which relies on a graph created using the methodology detailed on Section 3.1.1 and also expands set $\mathcal{F}$ with meta-features, included

an iterative step consisted of the analysis of the consensus of the results of several classifiers with the "incumbent" training set, followed by the filtering of samples on this training set, leading to new incumbent training sets on each iterations. The classification step performed on each iteration was always preceded by a feature selection stage, which was performed using a (α, β)-k-Feature Set approach. Interestingly, the significant increase in instance size - from only 120 features to 7260 in our case - was not the only complicating factor we encountered, as the perturbed problems obtained after each iteration were often more computationally demanding than the original ones, even though they had less samples. Since both sets $\mathcal{U}$ and $\mathcal{F}$ are perturbed, it is difficult to state which is responsible for the impact on the performance of the algorithm. Also, changing the values of the parameters, α, β and/or k also had a significant impact on the computational challenge imposed by the resulting problems, often making them prohibitively demanding to be tackled in practice.

Another consequence of such a construction, regards centrality: very central features (in other words, ranked important) will create a family of important metafeatures with similar centralities, and consequently, highly symmetrical subproblems whose solutions include them. A similar effect is observed in the way Set Cover instances are constructed from case/control datasets using the methodology described on Section 3.1.1: very "important" samples create highly symmetrical subproblems whose dual solutions include them. Therefore, it is reasonable to expect that the structure of each of the three sets of vertices, $\mathcal{A}, \mathcal{B}$ and $\mathcal{F}$ affect the difficulty of the (α, β)-k-Feature Set Approach.

Finally, it is also important to notice that the first two stages of the methodology, β is allowed to be zero with no penalty. Therefore, the nodes in $\mathcal{B}$ do not affect the objective function and these stages consider only the set $\mathcal{A} \subseteq \mathcal{U}$. The last two stages of the methodology reward work done on the sample pairs on the $\mathcal{B}$ side, and therefore consider the whole universe $\mathcal{U}$. The instances available on the literature do not partition the elements of the universe. Therefore their characteristics target only what is equivalent to the first two stages of the (α, β)-k-Feature Set approach, as the second partition would be empty. Since the focus of this work is on the final outcome of the whole methodology rather than on the (partial) solution of a specific step, that also suggests the need for a novel benchmark testbed that allows to

study how changes in either partition affect the tractability of the instance.

4.3 The Practical Difficulty of Instances of Covering Problems

A typical Set Cover instance consists of a cost array with weights for each of the available subsets, and a representation of the associated bipartite graph (see Section 3.3). As previously mentioned, in the case of (α, β)-k-Feature Set problem instances, all subsets have the same cost (i.e.: unitary costs).

To better understand one of the factors that can make a Set Cover instance difficult in practice, consider the most successful approaches to tackle Set Cover problems found on the literature. They are often based on Lagrangian Relaxation, following the same scheme proposed by Beasley (1990a), which dualizes constraints $\mathbf{Ax} \geq \mathbf{1}$ obtaining the Integer Program:

$$\max L(\lambda) \tag{4.1}$$

where $\lambda \in \mathbb{R}^+$, and

$$L(\lambda) = \min(\mathbf{c} - \mathbf{A}^T\lambda)\mathbf{x} + \mathbf{1}\lambda \tag{4.2}$$

As stated by Beasley (1990a); Ceria et al. (1998), for any given λ this problem is easily solvable by inspecting the signs of the reduced costs. It is easy to see that any vector x that minimizes the reduced costs has $x_j = 1$ if $(c_j - \sum_i a_{ij}\lambda_j) < 0$, $x_j = 0$ if $(c_j - \sum_i a_{ij}\lambda_j) > 0$ and $x_j \in \{0, 1\}$ if $(c_j - \sum_i a_{ij}\lambda_j) = 0$. The key components of Beasley (1990a)'s approach is to use reduced cost fixing and generate feasible solutions at every iteration. That is, for a given λ, a x vector is defined by inspecting the signs of the reduced costs and branching whenever they are null, case in which x_j may either be zero or one. Note that, at any point of this process, x is not necessarily a feasible solution of the Set Cover problem. Therefore, more columns are set to 1 until it becomes feasible. The choice of which column is to be set depends on a rank function, many of which are also proposed in the literature by the same aforementioned authors.

For the unicost case, however, these ideas would not be very effective because, since the

costs are all the same and unitary, most of the reduced cost signs would be negative, setting most of the columns to one, which is equivalent to selecting most sets. That would lead to a solution of very poor quality for the Set Cover problem, which could usually only be improved by column redundancy checks. Similarly, if the costs are different, but reside in a very restricted range, they would also all fall in the same case (either positive, negative or null). If the reduced costs are positive in their majority, most of the columns would be set to zero, leading to a solution that is highly infeasible for the Set Cover problem. The solution quality would therefore be highly dependent on the column ranking algorithm or branching. Ranking algorithms might provide poor solutions as the ranks would typically be very similar. Branching, on the other hand, proves to be an impractical choice for large problems. Finally, if most of the reduced costs are null, the algorithm would again rely too much on column ranking and/or branching, as few columns would be fixed. Therefore it is reasonable to think that an easy Set Cover instance would have a wider range of reduced costs, conveniently distributed within (relatively few) positive and (many) negative values, and with few null values, in order to obtain good bounds and rely little on branching.

Another important aspect of the instances that should be considered is row and column correlation. If many columns or rows can be eliminated using safe reductions, the kernel problem may be more tractable in practice. Note that this problem may still be difficult, but its solution time and resource requirements would be thus comparable to problems of smaller size.

That said, it also becomes clear why most of the algorithms in the literature have a more poor performance for the unicost case (Ceria et al., 1998). It is also reasonable to characterise difficult instances of the Set Cover Problem in terms of the constant terms of the reduced costs. Two main components are constant and might contribute to the difficulty of the instance: the column costs and distribution of "1s" in the value matrix. Disregarding safe reductions, the instances would become harder as the column costs become more similar and as the "1s" are more uniformly distributed on the value matrix. These conditions would make the reduced costs provide poor information, and compromise the bounds used by the algorithms. In other words, the state of art algorithms would find very symmetrical problems

more difficult. Symmetry detection and breaking are still open problems and are widely studied in the literature. See Margot (2010) for a brief review and examples in this topic. Commercial packages, such as CPLEX and GUROBI, also have very effective ways of detecting and breaking symmetries and thus can be very effective in solving Set Cover Problems. Another approach, however involves exploiting the symmetry of the problem to speed up the search whenever possible.

It is also worth mentioning that the second term of the reduced costs have a very interesting interpretation on the graph representation: they formulate the node degrees and can be seen as weights to the Lagrangian multipliers. That imposes the same difficulty on many of the few local search improvement and greedy heuristics found on the literature, as they often use this information to obtain initial solutions, restrict their candidate lists and guide their searches.

Finally, it is possible to show that, for the Set Cover Problem, the best bound obtained via Lagrangian relaxation equals the one obtained via linear programming relaxation (Geoffrion, 1974; Umetani and Yagiura, 2007). The former approach has been preferred on the state-of-the art approach for it allows for more efficient implementations, reducing the computational effort that the later would require. Near optimal bounds were usually enough to obtain optimal or very good results after solution polishing. The exact approaches based on Integer Programming, in turn, are also heavily guided by the reduced costs and hence often show the same behaviour, with a potential overhead introduced by solving the Linear Programming relaxation of the model.

4.4 Random Instance Generation

It is important to notice that, even though the (α, β)-k-Feature Set Problem defines its instances as tables with cases and controls, after it is translated to its graph representation (using the methodology described in Section 3.1.1) the methods available on the literature do not rely on any of the structural properties detailed on Section 4.2 to guide its solution process. The mathematical formulation used to solve it, however, is strongly correlated to that of Set Multi Cover Problems, and therefore would share also their weaknesses and difficulties,

as discusses on Section 4.2. Since this thesis aims to address the performance and scalability issues observed with this approach, ideally without changing its scope, the generation of Set Multi Cover Problem instances is preferred. One of the reasons for that is because it greatly increases the number of possible instances, providing a more interesting spectrum of instances that can comprise characteristics that could compromise the performance of the tools found on the literature to tackle the (α, β)-k-Feature Set Problem. It also significantly reduces the complexity (and running times) of the generator, since it no longer has to guarantee that the proposed graph can be reverted to a case/control table.

Therefore, the testbed used throughout this text considers a number of instances generated using Algorithm 6, to generate Set Multi Cover instances with a given edge distribution on sets $\mathcal{A}, \mathcal{B}$ and $\mathcal{F}$.

Algorithm 6: Instance Generator used to create the proposed testbed. The basic idea is to insert edges between a vertex in $\mathcal{F}$ and either $\mathcal{A}$ or $\mathcal{B}$ until the desired edge density is reached. The steps where a vertex is chosen at random defines how well distributed the edges are among the vertices. In this work, "choose a random" follows the linear distribution depicted on Figure 4.2, and takes the bias to the uniform distribution as a parameter.

Input: Edge densities: $density_{\mathcal{AF}}$ and $density_{\mathcal{BF}}$, sizes of the sets: $|\mathcal{F}|, |\mathcal{A}| and |\mathcal{B}|$.
Output: A random tripartite graph with $|\mathcal{F}|$ nodes in $\mathcal{F}$, $|\mathcal{A}|$ nodes in $\mathcal{A}$ and $|\mathcal{B}|$ nodes in $\mathcal{B}$; and approximate edge densities $density_{\mathcal{AF}}$ and $density_{\mathcal{BF}}$ from sets $\mathcal{A}$ and $\mathcal{B}$ to $\mathcal{F}$, respectivelly.

$nedges \leftarrow density_{\mathcal{AF}} * |\mathcal{F}| * |\mathcal{A}|$;
while $nedges > 0$ **do**
    choose a random $s \in \mathcal{F} : |N(s)| < |\mathcal{A}|$;
    choose a random $u \in \mathcal{A} : \nexists e = (u, s) \in \mathcal{E}$;
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, s)\}$;
    nedges $\leftarrow$ nedges-1 ;
$nedges \leftarrow density_{\mathcal{BF}} * |\mathcal{F}| * |\mathcal{B}|$;
while $nedges > 0$ **do**
    choose a random $s \in \mathcal{F} : |N(s)| < |\mathcal{B}|$;
    choose a random $u \in \mathcal{B} : \nexists e = (u, s) \in \mathcal{E}$;
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, s)\}$;
    nedges $\leftarrow$ nedges-1 ;

The main idea behind this method is to sequentially select nodes to connect with new edges until a desired edge density is met. Note that, to ensure that the resulting graph is a valid tripartite graph, one end of the edge must be a sample pair while the other must be

a feature. To generate instances with more different characteristics, the argument that the distribution of edges defines the instance's characteristic found on Section 4.3 is exploited. This is implemented by replacing the (standard) random number generator, which follows an uniform distribution, for one that follows a linear distribution.

The reason the linear distribution is chosen is that it allows to define both a smooth or abrupt bias. In other words, assuming any ordering of the candidate nodes, the first of any two adjacent sample pairs may be only slightly less or much less probable to be chosen than the other, depending on the defined parameter $b \in [0, 1]$. Figure 4.2 illustrates this point.

This random number generator is implemented using Algorithm 7. It generates numbers in the $[0, 1]$ interval following the continuous linear distribution depicted in Figure 4.2. A discrete number, that represents a node to be connected with an edge, is easily obtained using Algorithm 8. Using this random number generator, and assuming a $1 \rightarrow 1$ mapping of integers to candidate nodes in $\mathcal{A}, \mathcal{B}$ and $\mathcal{F}$, it is possible to determine how much bigger the degree of nodes mapped by higher integers are like to be with respect to the nodes mapped by lower integers, simply by adjusting the parameter b.

Algorithm 7: Linear Random Number Generator. Generates a number in the $[0, 1]$ interval using the linear distribution depicted on Figure 4.2, using a standard random number generator $rnd()$ that follows the uniform distribution. As defined by Equation 4.3, parameter $b \in [0, 1]$ defines how closely the probability function should approximate the uniform distribution.

Input: $b \in [0, 1]$
Output: $x \in [0, 1]$, chosen at random following the linear distribution depicted on Figure 4.2.

```
1 rnd_1 ← rnd();
2 rnd_2 ← rnd();
3 if rnd_1 > rnd_2 + b then rnd_1 ← rnd_2;
4 return rnd_1;
```

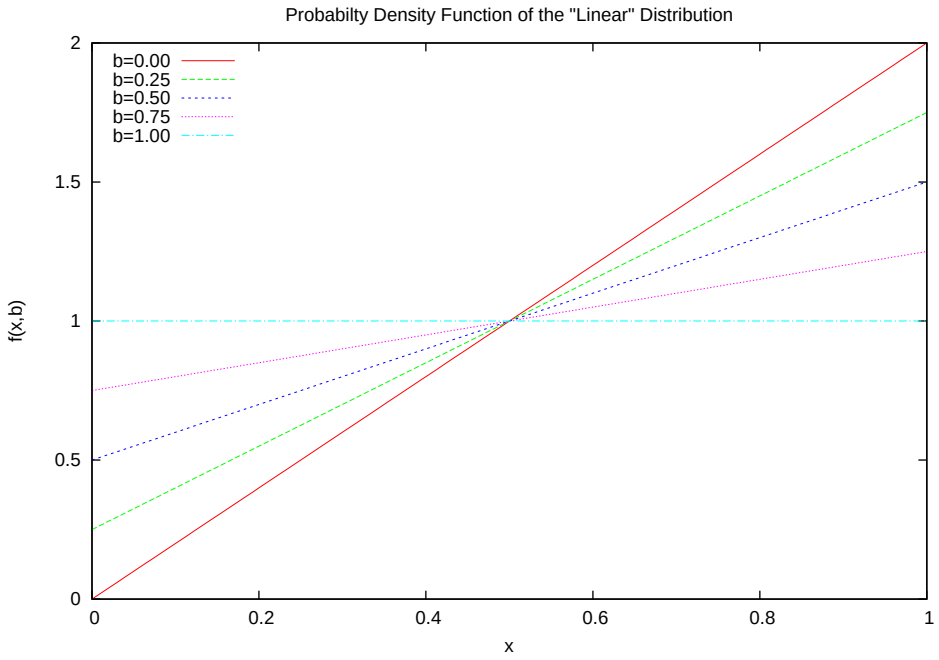
Algorithm 8: Algorithm to map integers in the interval $[x, y)$ to continuous ranges in $[0, 1]$.

Input: A continuous value $v \in [0, 1]$
Output: An integer in $[x, y)$

```
1 return ⌊(x + (y − x + 1))v⌋
```

To guarantee that there are no disconnected nodes, which could lead either to useless

Figure 4.2: Probability Density Function $f(x, b)$ of the Linear Distribution on continuous variables x and $b \in [0, 1]$, as detailed on (4.3). The variable b controls the bias on the linear distribution. As b tends to one, $f(x, b)$ approximates the uniform distribution. The probability $P(v_1 \leq x \leq v_2)$ for the random variable to fall within a particular region is given by the integral of this variable's density over the region, that is, the defined integral of the function for a specific interval $x \in [v_1, v_2]$, as detailed by (4.4).



$$f(x, b) = 2(1 - b)x + b \tag{4.3}$$

$$P(v_1 \leq x \leq v_2) = \int_{v_1}^{v_2} f(x, b)dx \tag{4.4}$$

■

features (if $s \in \mathcal{F}$ is disconnected) or an impossible instance (if $u \in \mathcal{U}$ is disconnected), Algorithm 9 is applied. That is achieved without changing the edge density by disconnecting the most dense valid (in the sense that it must be a sample pair if a feature is disconnected, or a feature otherwise) start or end node and reconnecting it to the disconnected one. The rationale behind this is to try to change the given biases as little as possible, by regretting only a choice that would be very frequent, what would not be very significant statistically.

Algorithm 9: Instance Generator: fix disconnections. This algorithm aims to connect any disconnected vertices without changing the edge densities, while minimising the impact on the edge distribution. That is achieved by disconnecting an appropriate node with very high degree and connecting the loose end to the disconnected node. The rationale behind that is only to regret a choice that would be very frequent anyway.

Input: A tripartite graph $G = (\mathcal{A} \cup \mathcal{B}, \mathcal{F}, \mathcal{E})$that may have disconnected nodes.
Output: A tripartite graph with no disconnected nodes.

while $\exists s \in \mathcal{F} : |N(s)| = 0$ **do**
  find $s\prime = max_{|N(s\prime)|}\{s\prime \in \mathcal{F} \backslash \{s\}\}$;
  find $u = max_{|N(u)|}\{u \in N(s\prime)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \backslash \{(s\prime, u)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, s)\}$;

`/* Do the same for the vertices in` $\mathcal{A}$ `*/`

while $\exists u \in \mathcal{A} : |N(u)| = 0$ **do**
  find $u\prime = max_{|N(u\prime)|}\{u\prime \in \mathcal{A} \backslash \{u\}\}$;
  find $s = max_{|N(s)|}\{s \in N(u\prime)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \backslash \{(u\prime, s)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, s)\}$;

`/* Do the same for the vertices in` $\mathcal{B}$ `*/`

while $\exists u \in \mathcal{B} : |N(u)| = 0$ **do**
  find $u\prime = max_{|N(u\prime)|}\{u\prime \in \mathcal{B} \backslash \{u\}\}$;
  find $s = max_{|N(s)|}\{s \in N(u\prime)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \backslash \{(u\prime, s)\}$;
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, s)\}$;

4.5 The Proposed Testbed

In this work, instances with 2000 features and two disjoint sets of 20000 sample pairs (totalling an universe of 40000) and an edge density of 20% are used. That would model the size of a case of a case-control dataset with 2000 features and 200 samples. The resulting instances

have 20000 pairs of samples in $\mathcal{A}$ and 20000 pairs of samples in $\mathcal{B}$, for example.

All instances were generated with the method proposed on Section 4.4, but it is not enforced that there exists a feasible case-control table of values that can be translated into it. Instead, to obtain a wide spectrum of instance characteristics that would cover more weaknesses of the existing approach, 125 classes of instances are defined, each one with three bias values on the uniform distribution (see Figure 4.2) b in $\{0, 0.25, 0.5, 0.75, 1\}$, one to control the distribution of neighbours of $\mathcal{F}$, $\mathcal{A}$ and $\mathcal{B}$ respectively.

Note that two instances generated with the same random seed and same values of the bias on the linear distribution for $\mathcal{F}$ and $\mathcal{A}$ would lead to the same instance of finding the maximum α and minimum k, but are still different since the set $\mathcal{B}$, necessary for finding the maximal β and total covering are different. That is an important remark because, even if the first two problems can be solved to optimality within reasonable time, the resulting problems solved by at the last two problems, after the introduction of set $\mathcal{B}$, might still be too demanding, not finish within reasonable time and compromise the whole (α, β)-k-Feature Set approach.

Table Table 4.2 depicts the proposed instance generation parameters $b_{\mathcal{F}}, b_{\mathcal{A}}$ and $b_{\mathcal{B}}$.

4.6 Experimental Analysis

In this section, all instances generated (see Section 4.5) are tackled by solving the models proposed by Berretta et al. (2007, 2008) using CPLEX version 12.2. One hour of computation time is allowed for each model, and the following stage, according to the methodology depicted on Section 3.2, uses the best solution found so far as input.

The difficulty of the instance is analysed considering the obtained optimality gap and running times.

Results on the first stage of the (α, β)-k-Feature Set Approach are omitted from this section because it can be solved in polynomial time using very little memory, and do not pose a serious challenge to the whole approach.

Throughout this thesis, the optimality gaps shown for the second and fourth stages of the (α, β)-k-Feature Set approach (minimisation of the number of selected features and maximisa-

Table 4.2: Proposed instances generation parameters. Each row represents one of the proposed instances. The first column associates the instance to an index that will be used to identify it throughout the text. The next three columns detail the biases on the linear distribution of edges incident to sets $\mathcal{F}, \mathcal{A}$ and $\mathcal{B}$, respectively.

Instance	$b_{\mathcal{F}}$	$b_{\mathcal{A}}$	$b_{\mathcal{B}}$	**Instance**	$b_{\mathcal{F}}$	$b_{\mathcal{A}}$	$b_{\mathcal{B}}$	**Instance**	$b_{\mathcal{F}}$	$b_{\mathcal{A}}$	$b_{\mathcal{B}}$	**Instance**	$b_{\mathcal{F}}$	$b_{\mathcal{A}}$	$b_{\mathcal{B}}$
1	0	0	0	33	0.25	0.25	0.5	65	0.5	0.5	1	97	0.75	1	0.25
2	0	0	0.25	34	0.25	0.25	0.75	66	0.5	0.75	0	98	0.75	1	0.5
3	0	0	0.5	35	0.25	0.25	1	67	0.5	0.75	0.25	99	0.75	1	0.75
4	0	0	0.75	36	0.25	0.5	0	68	0.5	0.75	0.5	100	0.75	1	1
5	0	0	1	37	0.25	0.5	0.25	69	0.5	0.75	0.75	101	1	0	0
6	0	0.25	0	38	0.25	0.5	0.5	70	0.5	0.75	1	102	1	0	0.25
7	0	0.25	0.25	39	0.25	0.5	0.75	71	0.5	1	0	103	1	0	0.5
8	0	0.25	0.5	40	0.25	0.5	1	72	0.5	1	0.25	104	1	0	0.75
9	0	0.25	0.75	41	0.25	0.75	0	73	0.5	1	0.5	105	1	0	1
10	0	0.25	1	42	0.25	0.75	0.25	74	0.5	1	0.75	106	1	0.25	0
11	0	0.5	0	43	0.25	0.75	0.5	75	0.5	1	1	107	1	0.25	0.25
12	0	0.5	0.25	44	0.25	0.75	0.75	76	0.75	0	0	108	1	0.25	0.5
13	0	0.5	0.5	45	0.25	0.75	1	77	0.75	0	0.25	109	1	0.25	0.75
14	0	0.5	0.75	46	0.25	1	0	78	0.75	0	0.5	110	1	0.25	1
15	0	0.5	1	47	0.25	1	0.25	79	0.75	0	0.75	111	1	0.5	0
16	0	0.75	0	48	0.25	1	0.5	80	0.75	0	1	112	1	0.5	0.25
17	0	0.75	0.25	49	0.25	1	0.75	81	0.75	0.25	0	113	1	0.5	0.5
18	0	0.75	0.5	50	0.25	1	1	82	0.75	0.25	0.25	114	1	0.5	0.75
19	0	0.75	0.75	51	0.5	0	0	83	0.75	0.25	0.5	115	1	0.5	1
20	0	0.75	1	52	0.5	0	0.25	84	0.75	0.25	0.75	116	1	0.75	0
21	0	1	0	53	0.5	0	0.5	85	0.75	0.25	1	117	1	0.75	0.25
22	0	1	0.25	54	0.5	0	0.75	86	0.75	0.5	0	118	1	0.75	0.5
23	0	1	0.5	55	0.5	0	1	87	0.75	0.5	0.25	119	1	0.75	0.75
24	0	1	0.75	56	0.5	0.25	0	88	0.75	0.5	0.5	120	1	0.75	1
25	0	1	1	57	0.5	0.25	0.25	89	0.75	0.5	0.75	121	1	1	0
26	0.25	0	0	58	0.5	0.25	0.5	90	0.75	0.5	1	122	1	1	0.25
27	0.25	0	0.25	59	0.5	0.25	0.75	91	0.75	0.75	0	123	1	1	0.5
28	0.25	0	0.5	60	0.5	0.25	1	92	0.75	0.75	0.25	124	1	1	0.75
29	0.25	0	0.75	61	0.5	0.5	0	93	0.75	0.75	0.5	125	1	1	1
30	0.25	0	1	62	0.5	0.5	0.25	94	0.75	0.75	0.75				
31	0.25	0.25	0	63	0.5	0.5	0.5	95	0.75	0.75	1				
32	0.25	0.25	0.25	64	0.5	0.5	0.75	96	0.75	1	0				

tion of the total covering, respectively) are calculated using Equation 4.5. For the third stage (maximisation of β), however, preliminary results have shown that the optimal objective function value can often be zero, which would result in a division by zero on Equation 4.5 that is addressed by adding a very small constant ϵ to the denominator. In these cases, the resulting gap would be exceedingly large and not describe the situation well. Therefore, the optimality gaps for this stage in particular are presented in non-percent form, and are calculated using Equation 4.6.

$$\frac{|\text{best primal bound} - \text{best dual bound}|}{\epsilon + |\text{best primal bound}|} \tag{4.5}$$

$$|\text{best primal bound} - \text{best dual bound}| \tag{4.6}$$

Figure 4.3 shows the stacked running times of each stage of the exact approach to solve the proposed instances.

It is immediately noticeable that, even though all instances have the same size, the running time required to solve them vary significantly. Next, it is also easy to see that the running time required to maximise α is insignificant with respect to the running times of the subsequent stages.

Looking at the column sizes of individual stages at the same figure, it is possible to see that with the exception of the columns associated to the maximisation of β, they are mostly of about the same size, and capped at 3600 seconds. That means that most of the problems reached the timeout limit.

In the case of the maximisation of β, most of the values seem to exceed the limit of 3600 seconds. That means that, not only these problems reached the timeout, but also often needed significantly more time to finish work in progress. It also becomes evident that this is the most demanding stage of the approach.

It is also possible to see that, excluding the maximisation of α, it is rarely the case that the exact approach reaches timeout to solve one stage, but solves any of the next stages quickly.

Table 4.3 shows the average sizes of the kernel problems, after pre-processed by CPLEX's

Figure 4.3: Stacked the running times of each stage of the exact approach to solve the proposed instances. Each column represents the stacked running times the stages of the $(\alpha, \beta)-k$ Feature Set approach for each of the proposed instances. The instances are ordered in ascending order of computational effort required. In Appendix D.1, Table D.1 detail these values and identify the instances in the order they are shown.

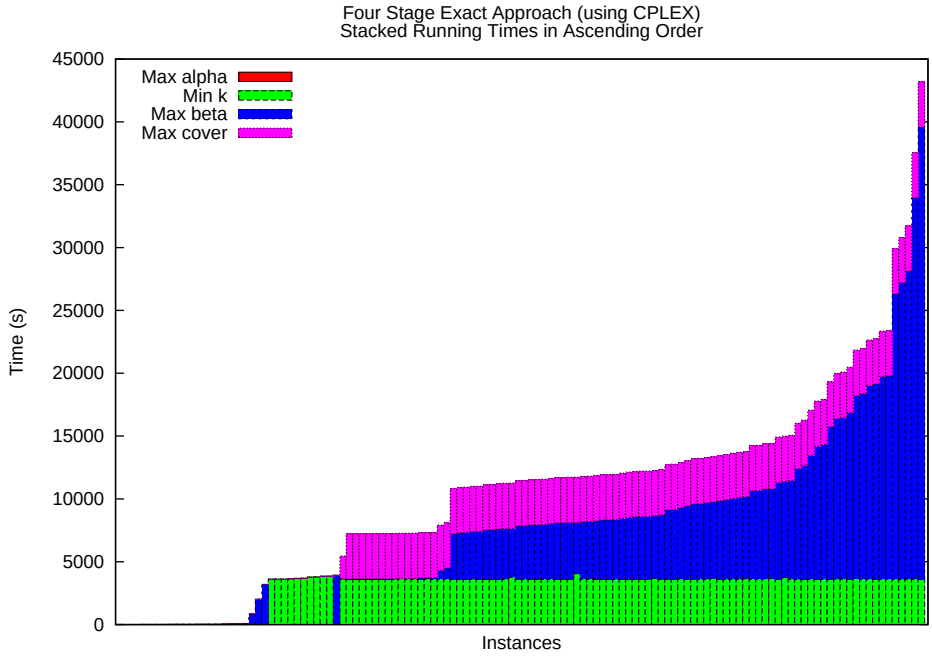
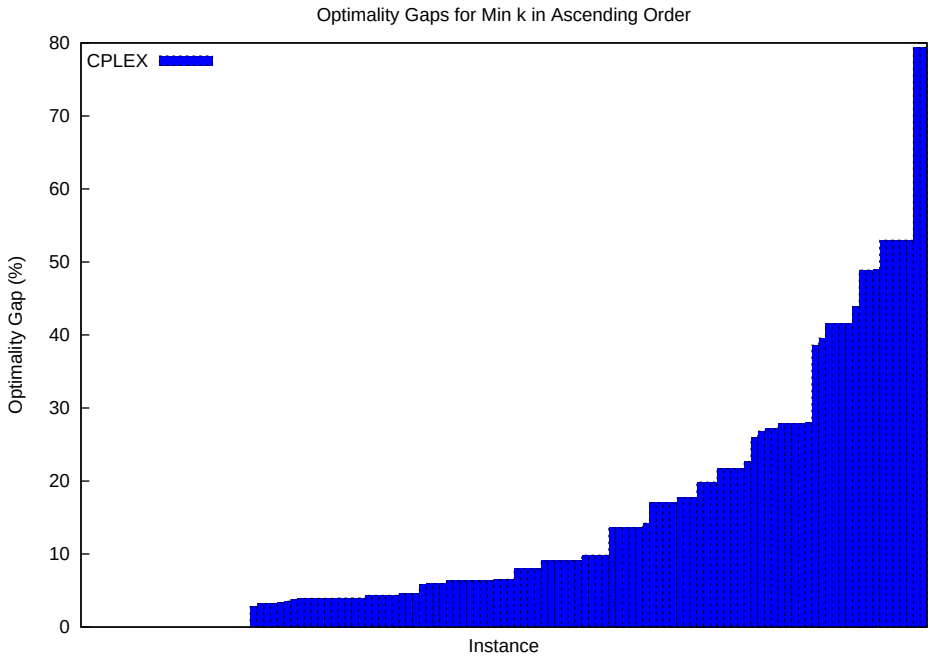


Figure 4.4: Optimality gaps, calculated with Equation 4.5, obtained with exact approach for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps, before the timeout of one hour. In Appendix D.1, Table D.2 detail these values, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree.



presolve, and average number of nodes. It is possible to see that the instances could not be reduced much for the second stage of the approach, which is modelled as the unicost Set Multi Cover problem. However, it could be reduced a more for the next two stages in average, which still did not make the solution process finish within reasonable time. It is also interesting to notice that the first and third stages of the approach explored significantly more nodes before timeout.

Since the exact method could not solve most of the proposed instances within reasonable time, it often obtained a non-zero optimality gap. Figure 4.4, Figure 4.5 and Figure 4.6 illustrate the obtained gaps for the minimisation of the number of selected features, maximisation of β and maximisation of the total covering, respectivelly.

Once more it became evident that size is not the only factor that makes the instance challenging as, even though all instances were of the same size, the obtained optimality gaps

Table 4.3: Average CPLEX results for the last three stages of the $(\alpha, \beta) - k$ Feature Set approach, and their respective standard deviations. The first column identifies the stage. The second and third columns show the sizes of sets $\mathcal{F}$ and $\mathcal{U}$, respectively, after performing safe reductions (CPLEX's presolve). The next three columns show the optimality gap, calculated using Equation 4.5, the running time to solve this stage and the number of nodes of the solution enumeration tree that CPLEX had to expand, respectively.

Stage	**Red. $\mathcal{F}$**	$\pm$	**Red $\mathcal{U}$**	$\pm$	**Gap**	$\pm$	**Time**	$\pm$	**Nodes**	$\pm$
Min k	90.03%	21.20%	80.35%	39.59%	12.35%	13.89%	2902.90	1481.25	655.40	872.63
Max β	86.34%	28.06%	68.06%	38.45%	2.91	3.10	4939.38	6308.06	8.84	57.50
Max Cover	84.67%	29.19%	55.66%	39.27%	3.00%	2.99%	2580.31	1621.13	2648.19	3400.93

Figure 4.5: Optimality gaps, calculated with Equation 4.6, obtained with exact approach for the maximisation of β. The instances are ordered in ascending order of optimality gaps, before the timeout of one hour. In Appendix D.1, Table D.3 detail these values, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree.

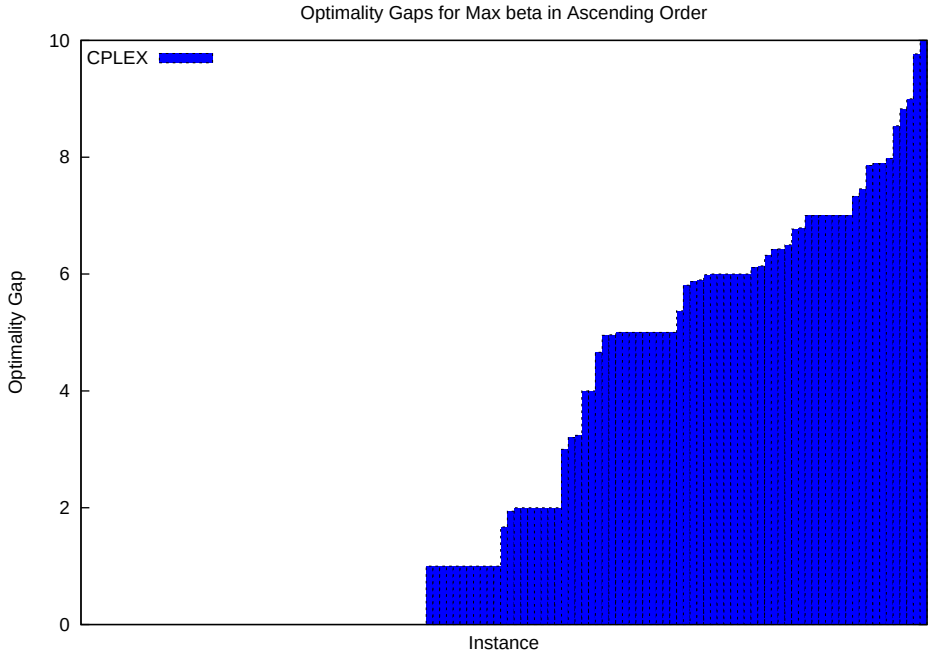
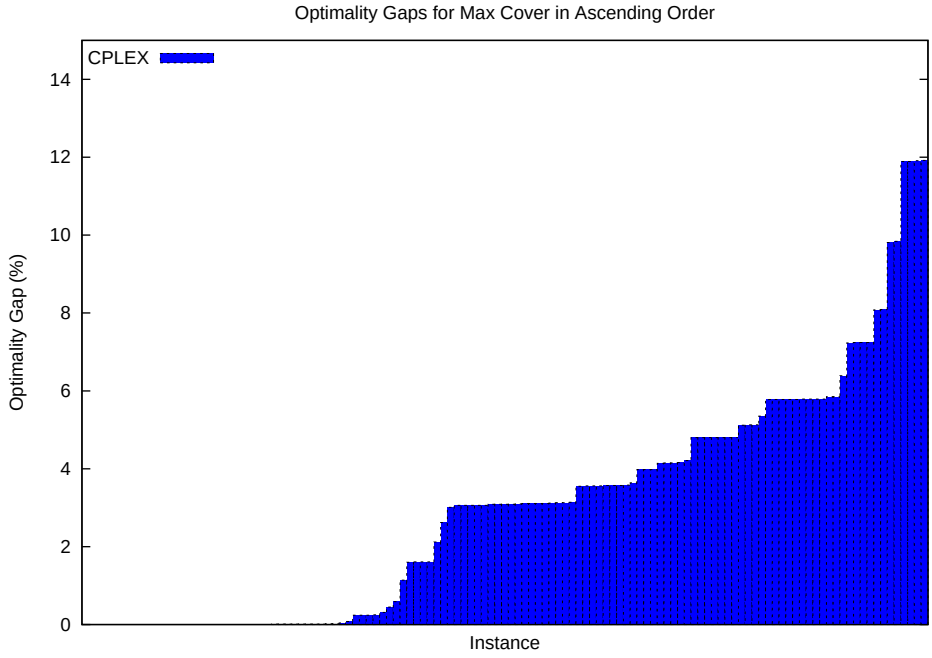


Figure 4.6: Optimality gaps, calculated with Equation 4.5, obtained with exact approach for the maximisation of the total covering. The instances are ordered in ascending order of optimality gaps, before the timeout of one hour. In Appendix D.1, Table D.4 detail these values, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree.



varied significantly.

Analysing Figure 4.4, it is possible to see that, the obtained gaps can be quite big, reaching up to 80%. However, analysing Figure 4.3 and Figure 4.5, it is interesting to notice that, even though the exact method puts a considerable effort in improving β, the difference between the obtained solution and obtained upper bound does not exceed ten features. A similar observation can be made for the percent differences for maximisation of the total covering by analysing Figure 4.3 and Figure 4.6.

4.7 Conclusions

Analysing the running times of each of the (α, β)-k-Feature Set Approach for different real world instances, it becomes clear that, even though many instances can be solved in reasonable time by the exact approach, such as instances DS, ADMF, PD1, PD2 and PC; that is not always the case and SM is a good example of that.Moreover, if it is the case that the solver has to be ran several times, larger computation times such as those observed for PD1 and PC could quickly become an issue.

Therefore, in this chapter, (α, β)-k-Feature Set Problem instances were detailed and analysed, and a suitable testbed to benchmark algorithms for the problem was proposed.

The Set (Multi) Cover instances available on the literature are not suitable for the purposes pursued in this work primarily because they do not have the characteristics that are expected to be found on practical instances of the (α, β)-k-Feature Set Problem. In particular, they have $|\mathcal{F}| \ggg |\mathcal{U}|$, as opposed to $|\mathcal{U}| \ggg |\mathcal{F}|$, which is the expected. Moreover, they do not partition $\mathcal{U}$ in two correlated and disjoint sets $\mathcal{A}$ and $\mathcal{B}$, and doing so could break the particularities of the instance, and the methodology involved would itself be a different problem.

The proposed testbed, generated with the method also proposed in this chapter, provided an instance pool suitable to the purposes pursued. As argued on the introduction of this chapter, safe-reductions could still be performed by the CPLEX presolve routine, and therefore the difficulty of most of the instances proposed in Section 4.5 would be still comparable to that of smaller instances. However, for the most demanding instances - which are most interesting

- the reducibility is either not significant with respect to the instance size or still not enough to make the problems tractable.

Observing the performance of the exact approach on the proposed testbed, it became clear that for the second and last stages of the (α, β)-k-Feature Set approach (minimisation of the number of selected features, and maximisation of β), the main difficulty is excessive enumeration. That could be due to the fact that either the primal or dual bounds used by the solver are unsatisfactory. On the other hand, given the small number of expanded nodes and the excessively large running times, which included significant effort to finish work in progress, obtaining any feasible solution at all seemed to be an issue already.

Finally, given the big computational effort involved in the four stage approach, it is reasonable to question its effectiveness. Also, since it includes a hard and well known problem, with many explored feasible solutions on the beginning, followed by a very demanding related problem, with few feasible solutions and little space for improvement; different designs could perhaps use more information obtained during the solution process on the following problems.

Chapter 5

Heuristic Design and Implementation

When choosing a solution method for any problem, one should take into consideration aspects about the characteristics of the instances being tackled.

The size of the instance affects the choices that regard resource availability, such as how much time one can afford to spend searching for solutions and how much memory is available. Exact combinatorial algorithms for NP-Hard problems usually require a lot of time and/or memory but provide an optimality guarantee. That, however, is clearly impractical for many large instances. Giving up the proof of optimality with approximative algorithms and heuristics often leads to much faster and leaner methods. Still, one has to bear in mind the gap between "tractable within reasonable time" and "efficient" (in terms of computational complexity). Even though an $O(n^2)$ algorithm would be considered efficient (and cheap), it may still be impractical, if the problem is big enough or if it often exceeds the time limit.

The characteristics of the instances affect this choice in what regards the approach. Instances with characteristics that can be exploited to speed up the solution process may be solved using exact approaches, which include Integer Programming and specialised Branch-and-Bound algorithms. In these cases, giving up the proof of optimality is often not necessary. On the other hand, some instances may be more challenging for these approaches and heuristics should be preferred in practice. If the instance not only has problematic characteristics, but is also large, even simple search procedures may become impractical, and fast greedy procedures may be more interesting in practice.

As discussed on Chapter 3, the (α, β)-k-Feature Set approach involves four stages, for which at least one is a NP-Complete problem[1]. Experience also showed that, even though the first stage can be solved efficiently and, despite its NP-Completeness, many practical instances of the second can also be tackled with exact approaches, that is not often the case for instances that have been arising in recent studies (see Chapter 4). Also, the following two stages, which only look for an optimal solution of the second with particular properties, are often even more computationally demanding than the first two. Since the problem is highly symmetric, and feasible solutions of any stage must be optimum for the previous, restarting optimization from scratch, or simply from a feasible solution, might discard interesting information that could be exploited.

Therefore, this chapter discusses heuristics and search techniques to tackle the (α, β)-k-Feature Set approach, described on Section 3.2, in a different way. Instead of partitioning the problem and solving the subproblems to optimality, the proposed heuristics treat the approach as a single problem, and attempt to optimise all stages simultaneously from start solutions that are optimal for the first stage and feasible for the second. Not only that has the immediate advantage of not discarding information collected during the solution process, but also ensures that, if computation has to be stopped, the incumbent solution has more interesting characteristics than if the following stages were not performed at all, which would be more interesting in practice.

This chapter starts by detailing how to obtain initial solutions with various algorithms shown on Section 5.2. It then proceeds to Section 5.3 with methods to improve these solutions, optimising all four settings simultaneously: α, β, k and the total covering. Next, Section 5.4 details how to avoid getting stuck on local minima, allowing the algorithms to explore other regions of the feasible solution space, and increasing the chance of finding solutions of better quality. Finally, Section 7.1 experiments with the proposed algorithms and summarises the results. Section 8.1 concludes the chapter.

[1] The second stage is a particular case of the Set Multi Cover problem, which is NP-Complete. Also, the (α, β)-k-Feature Set Problem is a generalization of the k-Feature Set problem, which is equivalent to the $(1, 0)$-k-Feature Set Problem, and also NP-Complete(Garey and Johnson, 1979).

5.1 Solutions and Movements

A solution for the (α, β)-k-Feature Set Problem is basically a set of selected features $\mathcal{F}\prime$. As mentioned on Chapter 3, given values of α, β and k, this set of features must have at exactly k vertices that cover the elements in $\mathcal{A}$ at least α times and the elements of $\mathcal{B}$ β times. In the proposed implementation, however, only the value of α (α^*) is given, as it is defined deterministically beforehand. The values of β and k are optimised simultaneously, whilst also trying to maximise the total covering.

For such, two basic movements are considered in this work: *select* and *deselect* feature.

Given a set of selected features $\mathcal{F}\prime$, the *select* feature movement includes a feature $f \in \mathcal{F}$ in it: $select(f) = \mathcal{F}\prime \leftarrow \mathcal{F}\prime \cup \{f\} \in \mathcal{F}$. After selecting a feature f, the covering requirements d_u of its neighbours $u \in N(f)$ also have to be decremented: $d_u \leftarrow d_u - 1, \forall u \in N(f)$. Also, $|N(f)|$ has to be added to the total covering and β has to be recalculated from scratch (see Section 5.1.1 for a formal definition of how to calculate the objective function values).

Similarly, the *deselect* feature movement removes a feature $f \in \mathcal{F}$ from $\mathcal{F}\prime$. $deselect(f) = \mathcal{F}\prime \leftarrow \mathcal{F}\prime \backslash \{f\} \in \mathcal{F}$. After deselecting a features f, the covering requirements d_u of its neighbours $u \in N(f)$ also have to be incremented: $d_u \leftarrow d_u + 1, \forall u \in Nf$. Also, $|N(f)|$ has to be removed from the total covering value and β has to be recalculated from scratch.

In this work we also use compound movements, such as swaps. These include at least one of the basic movements, *select* and/or *deselect*, performed in any order. Let $\mathcal{F}^1 \in \mathcal{F}\prime$ and $\mathcal{F}^2 \in \mathcal{F} \backslash \mathcal{F}\prime$ be arbitrary sets of selected and unselected features. A *swap* movement would consist of performing *deselect* movements of features in $\mathcal{F}^1$ and *select* movements of features in $\mathcal{F}^2$: $swap(\mathcal{F}^1, \mathcal{F}^2) = \mathcal{F}\prime \leftarrow \mathcal{F}\prime \backslash \mathcal{F}^1 \cup \mathcal{F}^2$.

Figure 5.1 illustrates *select* and *deselect* movements.

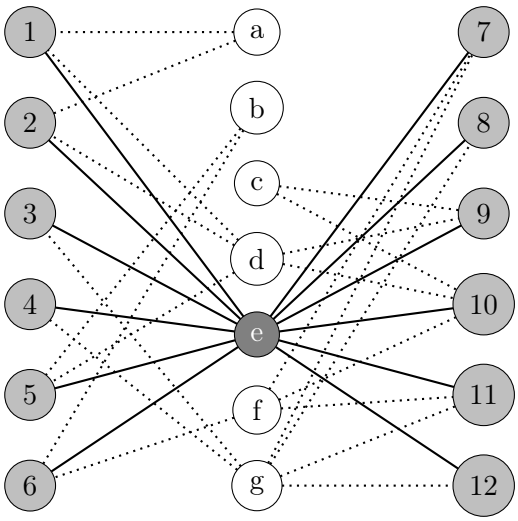
5.1.1 Feasible Solutions and Improving Movements

As previously mentioned, in this work, four objective functions are considered hierarchically: maximise α, minimise k, maximise β and maximise the total covering.

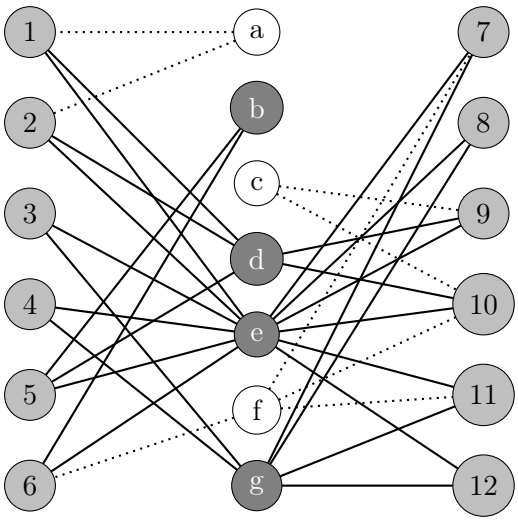
Since the first stage of the (α, β)-k-Feature Set approach can be solved to optimality efficiently, the optimum value of α is known (α^*), and does not have to be further optimised.

Figure 5.1: Given a universe $\mathcal{U} = \{1..12\}$ and the set of available subsets $\mathcal{F} = \{a = \{1,2\}, b = \{5,6\}, c = \{9,10\}, d = \{1,2,5,9,10\}, e = \{1,2,3,4,5,6,7,8,9,10,11,12\}, f = \{6,7,10,11\}, g = \{3,4,7,8,11,12\}\}$, In the figures below, the dark nodes represent the selected features and the full edges represent their coverings.. If one performs a *select* movement on a feature (say *select(b)* on (a)), k increases by 1 and the total covering increases by the number of neighbours it has (two, in this case). The values of α and β have to be recalculated by counting the minimum number of coverings on the elements of $\mathcal{A}$ and $\mathcal{B}$, respectivelly (the full edges), and wouldn't change in this case as any other edge that is not a neighbour of b would still be covered only once. A *deselect(b)* movement would have the exact opposite effect, undoing this procedure. If we select $\{b, d, g\}$, however, both α and β would be increased to 2, as all elements would be covered at least twice. Note also that, if we took $k = 1, \alpha = 1, \beta = 1$ as arguments for the problem modeled by Figure 3.2, then (a) would be feasible, at it would satisfy all the constraints. However, if we required that $\alpha = 2, \beta = 2$ it would no longer be feasible, for not covering the elements of $\mathcal{A}$ and $\mathcal{B}$ enough times. In this case, more features would have to be selected and (b) would illustrate a feasible solution, with at least two full edges connected to each element of $\mathcal{A}$ and $\mathcal{B}$.

(a) In this picture, $k = 1$ because only one feature ({e})is selected, $\alpha = 1$ because the current feature set ({e}) only covers each element of $\mathcal{A}$ once, $\beta = 1$ because the current feature set only covers the elements of $\mathcal{B}$ once and the total covering is 12, because it cover every element of the universe exactly once.



(b) In this picture, $k = 4$ because the current feature set contains four elements ({b,d,e,g}), $\alpha = 2$ because the current feature set only covers each element of $\mathcal{A}$ at least twice, $\beta = 2$ because the current feature set covers the elements of $\mathcal{B}$ at least twice and the total covering is 25, because it covers every element of the universe exactly twice, with the exception of element 5, that is covered three times.



Therefore, throughout this work, a solution is *feasible* if all sample pairs $u \in \mathcal{A}$ have their covering requirements satisfied, that is, $d_u \leq 0, \forall u \in \mathcal{A}$.

Let $G(\mathcal{U}, \mathcal{F}\prime, \mathcal{E})$ be the subgraph obtained by removing all nodes in $\mathcal{F} \backslash \mathcal{F}\prime$ and edges connected to them[2].

A solution is then evaluated observing the values of k, β and total covering on the graph $G(\mathcal{U}, \mathcal{F}\prime, \mathcal{E})$:

- $k = |\mathcal{F}\prime|$
- $\beta = \min_{u \in \mathcal{B}}\{|N(u)|\}$
- Total Covering $= \sum_{u \in \mathcal{U}} |N(u)|$

Therefore, a movement is considered improving if the resulting solution is feasible and yields, with respect to the original solution, either (in this order):

1. A smaller value of k
2. A higher value of β, without increasing the value of k
3. A higher total covering, without decreasing the value of β or increasing the value of k

The function *isBetter*($\mathcal{F}_1\prime, \mathcal{F}_2\prime$), referenced throughout this chapter, returns true if the solution given by $\mathcal{F}_1\prime$ is better, in these terms, than $\mathcal{F}_2\prime$; and false otherwise.

This policy not only has the obvious advantage of saving the time that would be spent solving the two next stages separately but also acts in synergy with the overall search mechanism, introducing diversity and preventing cycling.

5.2 Constructive Greedy Algorithms

In this work it is required that all feasible solutions be optimum solutions of the first stage of the (α, β)-k-Feature Set approach. Therefore, the problem modelled in Figure 3.4 is solved to optimality by inspection, and α is set to the obtained optimal value α^*. Then, the problem of

[2] Figure 3.8b illustrates that. While the full graph would be $G(\mathcal{U}, \mathcal{F}, \mathcal{E})$, the graph represented by the coloured nodes and full edges would be $G(\mathcal{U}, \mathcal{F}\prime, \mathcal{E})$.

finding initial solutions for the problem under consideration is that of finding initial solutions for the Minimum Cardinality Set Multi Cover problem, with the covering requirements set to α^* for all vertices in $\mathcal{A}$ and zero for all vertices in $\mathcal{B}$. Note, however, that the algorithms detailed in this work do not assume this particularity. That is because, after any selection the covering requirements of (potentially) only a group of sample pairs is changed. Therefore the associated subproblems no longer share the same covering requirements and are no longer the Minimum Cardinality (α, β)-k-Feature Set modelled on Figure 3.5, but the unicost Set Multi Cover Problem itself.

Constructive greedy heuristics are usually designed to be a fast and reasonably simple way of finding feasible solutions for combinatorial problems. Their main characteristic is that every choice is made in a greedy fashion, based on the current state of the solution. They make the decision that seems best at the moment, never regretting or revisiting it. The remaining problem is then assessed in the same way. This procedure is repeated until a feasible solution is found.

Unless under very specific conditions where the greedy solution is optimal (see Cormen, 2001, for a discussion on such conditions), the solution can be improved by other refinement methods or backtracking. Therefore, greedy algorithms are often used to provide initial solutions and starting points for search algorithms. A good selection criterion, however, may help to obtain better solutions in general.

In this work, two constructive greedy heuristics are considered: Algorithm 10 and Algorithm 11. Note also that, rather than assuming that the proposed algorithms create a solution from an empty one, with no selected features, they assume any solution as input and simply attempt to make them feasible. That is, it selects features until all sample pairs in $\mathcal{A}$ have their covering requirements d_u satisfied. An empty or infeasible solution would then have its set of selected features increased and a feasible solution would have its set of selected features left unchanged.

Algorithm 10 selects the unselected feature that has more neighbours with unsatisfied covering requirements until the solution is feasible. It attempts to make the solution feasible using as few features as possible.

Algorithm 11 details a proposed variation that demands that the newly selected feature

Algorithm 10: Greedy Algorithm 1: always choose the unselected feature that has more neighbours with unsatisfied covering requirements. This method attempts to make the solution feasible using as few features as possible.

Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
1 while ∃u ∈ A : d_u > 0 do                    // The solution is not feasible
      // select the unselected feature that has more neighbours in A with
         unsatisfied covering requirements
2     select(max_{f∈F\F'}{|{u ∈ N(f) ∩ A : d_u > 0}|});
```

be a neighbour of the sample pair that needs more coverings. This modification attempts to focus on the most infeasible sample pair, by minimising the difference between covering requirements of the nodes in $\mathcal{A}$ as new features are selected.

Algorithm 11: Greedy Algorithm 2: always choose the unselected feature that has more neighbours with unsatisfied covering requirements and covers the vertex with least coverings. This method attempts to make the solution feasible using as few features as possible, focusing on the most infeasible part of the solution.

Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
1 while ∃u ∈ A : d_u > 0 do                    // The solution is not feasible
      // find sample pair that needs more coverings
2     u ← max_{u∈A}{d_u};
      // select the unselected feature that has more neighbours with
         unsatisfied covering requirements and covers the vertex with least
         coverings
3     select(max_{f∈N(u)∩(F\F')}{|{v ∈ N(f) ∩ A : d_v > 0}|});
```

5.2.1 Constructive Greedy Algorithms and Safe Reductions

Following the idea of greedy algorithms, another important technique is to always reduce the problem to a kernel before working on it. In the (α, β)-k-Feature Set Problem case, one of the steps that can be taken in that direction is to select any features that are provably in a global optimum before making any greedy choices. After any selection, the remaining problem can be seen as a (smaller) Set Multi Cover instance, as the covering requirements may no longer be the same for every sample pair in $\mathcal{A}$. Thus, safe reductions like the ones detailed on Section 3.4 may be performed.

It is important to notice that using safe reductions within the greedy algorithms does not guarantee that the global optimum, or even a better solution, will be found, but increases the chance that this may happen. Any reduction, such as a safe reduction or an arbitrary movement, prunes a family of feasible solutions that do not include it. Therefore, if safe reductions are performed before any unsafe greedy choice, it reduces the number of attainable feasible solutions, which could include local optimum solutions, without losing global optima that include the features selected, considering the choices that were already made. If feasibility is achieved only through a series of safe reductions, the obtained solution is one of the many possible global optima. However, every time an unsafe greedy choice is made, the task of reaching a global optimum might become unattainable.

Algorithm 12 extends Algorithm 11 by attempting to reduce the current subproblem using the safe reductions described on Section 3.4.

Algorithm 12: Greedy Algorithm 3: accounting for safe reductions. After reducing the current subproblem to a kernel, choose the unselected feature with most unselected neighbours that covers the vertex with least coverings. This method also attempts to make the solution feasible using as few features as possible, focusing on the most infeasible part of the solution, and always checking for safe reductions.

Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
1 while ∃u ∈ A : d_u > 0 do                    // The solution is not feasible
      // perform safe reductions
2     run(Algorithm 2);
      // find sample pair that needs more coverings
3     u ← max_{u∈A}{d_u};
      // select the unselected feature that has more neighbours with
         unsatisfied covering requirements and covers the vertex with least
         coverings
4     select(max_{f∈N(u)∩(F\F')}{|{v ∈ N(f) ∩ A : d_v > 0}|});
```

Due to the size of the instances under consideration (2000 features and 40000 sample pairs), in this work, only the reduction detailed on Algorithm 2 is considered. Fortunately, it is not only the cheapest to perform, but preliminary testing showed that it is also the most effective. As the instance size grew, Algorithm 3-Algorithm 5 did not seem to be cost-effective, as they introduced an overhead that made them prohibitive to be ran sporadically, or even only once, and did not significantly contribute to the solution quality after Algorithm 2 was performed.

A similar behaviour has already been reported for the Set Cover Problem by Beasley (1990a).

It is also interesting to notice that, by construction of the (α, β)-k-Feature Set approach, all the neighbours of at least one pair of samples have to be selected by Algorithm 2: the neighbours of the sample pair that defined α^*. That can be quite a significant reduction, when the value of α^* is relatively high, with respect to $|\mathcal{F}|$.

5.3 Neighborhoods and Local Searches

As already suggested by Caserta (2007), very often Set Cover instances derived from biological datasets, in special case-control datasets, differ from traditional ones because they have a number of elements in their universe much greater than the number of features. Considering the methodology to create the graph representation of the instance, as described in Section 3.1.1, this behaviour is easily confirmed, as it leads to a graph that has a number of elements of the order of the number of samples squared. The same argument applies to (α, β)-k-Feature Set Problem instances.

Motivated by Caserta's suggestion that such problems may be better addressed with primal intensive approaches, the methods proposed in this work consider swap based neighbourhoods to improve the solutions' quality. Since the main objective at this stage is to reduce the number of selected features, such neighbourhoods comprise movements consisted of deselecting a set of features and selecting one that is different, smaller and disjoint from the set of features being deselected. Let these neighbourhoods be defined as $S_{i,j}$ and comprise $\binom{|\mathcal{F}'|}{i}\binom{|\mathcal{F}|-|\mathcal{F}'|}{j}$ possibilities, where i selected features are deselected, and $j < i$ unselected features are selected instead. It is easy to see that the bigger the set of features to be selected and deselected, the bigger is the neighbourhood.

The simplest and smallest of these neighbourhoods would include all feasible solutions that can be obtained simply by deselecting one feature. The next possibilities, in increasing size of number of candidate movements, include all feasible solutions that can be attained by swapping two selected features for one unselected ($S_{2,1}$), three selected features for one unselected ($S_{3,1}$), three selected features for two unselected ($S_{3,2}$) and so on.

A local search attempts to find and perform improving movements in a solution's neighbourhood, "moving" then to a neighbouring better solution. When there are no improving movements in a certain neighbourhood to be performed, the incumbent solution is a local optimum with respect to that neighbourhood. The global optimum is the best of all existent local optima. Typically, a local search stops when it reaches a local minimum.

Since every neighbourhood may include many different improving movements, which lead to different incumbent solutions, with different neighbourhoods, the choice of which movement to choose is also important and typically made in two greedy fashions: first or best improvement. The former accepts the first improving solution it finds while the latter finds all improving solutions and only accepts the one that leads to the biggest improvement to the objective function.

Algorithm 13 and Algorithm 14 depicts typical swap-based first and best improvement local searches, respectively.

Algorithm 13: A swap-based first-improvement local search. It performs the first improving swap that it finds until none can be found.

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the number of features to remove SZ_1, and include SZ_2 ($SZ_1 > SZ_2$)
Output: A feasible solution with (potentially) improved values of k, β and/or total covering

```
1  improved ← TRUE;
2  while improved do
3      LABEL: marker;
4      improved ← FALSE;
       // All possible subsets of F' of size SZ_1
5      forall F^1 ⊆ F' : |F^1| = SZ_1 do
           // All possible subsets of F\F' of size SZ_2
6          forall F^2 ⊆ F\F' : |F^2| = SZ_2 do
7              if isBetter(F'\F^1 ∪ F^2, F') then
8                  improved ← TRUE;
9                  swap(F^1, F^2);
10                 GOTO marker;
```

Any movement in swap-based neighbourhoods always change the number of selected features by the same amount. Therefore, the concept of a best improvement local search would not apply directly, as all possible improving movements would yield the same advantage.

Algorithm 14: A swap-based best-improvement local search. It finds all available improving swaps, and performs the one that improves the given objective functions the most.

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the number of features to remove SZ_1, and include SZ_2 ($SZ_1 > SZ_2$)

Output: A feasible solution with (potentially) improved values of k, β and/or total covering

```
1  improved ← TRUE;
2  while improved do
3      improved ← FALSE;
       // All possible subsets of F′ of size SZ_1
4      forall F^1 ⊆ F′ : |F^1| = SZ_1 do
           // All possible subsets of F\F′ of size SZ_2
5          forall F^2 ⊆ F\F′ : |F^2| = SZ_2 do
6              if isBetter(F′\F^1 ∪ F^2, F′) then
                   // Keep track of the best improving solution found
7                  if improved=FALSE then // First improving solution found
8                      F^1* = F^1;
9                      F^2* = F^2;
10                     improved ← TRUE;
11                 else
12                     if isBetter(F′\F^1 ∪ F^2, F′\F^1* ∪ F^2*) then
13                         F^1*=F^1;
14                         F^2*=F^2;
15         if improved=TRUE then
16             swap(F^1*, F^2*);
```

However, a best improvement local search for the (α, β)-k-Feature Set approach could still be designed considering that three objective functions are optimised simultaneously, using the objective function hierarchy to resolve ties as detailed on Section 5.1.1.

5.4 Escaping Local Minima

As previously mentioned, greedy algorithms may lead to good solutions but, unless under very special circumstances, the obtained solutions can be improved by local search methods.

Local search methods guarantee that the solution will be a local optimum, but not always guarantee that it will be a global optimum. Moreover, the bigger the neighbourhood defined, the more expensive these methods are, because they can scale up to be fully enumerative. Therefore, the neighbourhood size must be carefully chosen according to the heuristic's design and application: how big are the target instances and how often is the local search expected to run. One can usually afford to run a more expensive, and therefore more effective, local search for small instances, or with heuristics that do not use it too frequently.

Since running the same local search on different solutions may lead to different local optima, which may include the global, an alternative would be to generate several different good quality solutions, attempt to improve all of them using fast local searches and pick the best one found. The methods proposed in this work introduce diversity to the pool of solutions using randomizations of the proposed constructive greedy heuristics and local searches within a Meta-Heuristic design.

5.4.1 Randomized Initial Solutions

The first randomization method, illustrated by Algorithm 15, simply deselect a few features, chosen at random, and uses a greedy heuristic to restore feasibility. This allows the algorithm to explore more of the infeasible solution space, as infeasible solutions that were not visited by the greedy algorithm before may be attained. Therefore, the quality of the solutions obtained using this method depends ultimately on the greedy algorithm adopted.

The next method, depicted by Algorithm 16, simply selects random redundant features to

Algorithm 15: A randomization method focused on the greedy algorithm: deselect a few features chosen at random, and use a greedy heuristic to make it feasible again. This allows the algorithm to explore more of the infeasible region of the solution space.

Input: A feasible incumbent solution $\mathcal{F}\prime$, the number of features to deselect SZ_1
Output: A (potentially) different feasible solution
1 Let $\mathcal{F}^1 \subset \mathcal{F}\prime : u \in \mathcal{F}\prime$ is chosen at random;
2 $deselect(\mathcal{F}^1)$, $|\mathcal{F}^1| = SZ_1$;
`// Either Algorithm 10, Algorithm 11 or Algorithm 12`
3 $greedy(\mathcal{F}\prime)$;

swap the incumbent solution for one with more improving neighbouring solutions, hoping that the local search can find a succession of improving movements that leads to a better solution. After this procedure, the current solution would clearly become "worse", as it would have more selected features than before. Therefore the quality of the target solutions depends ultimately on the ability of the local search, which is assumed to be performed afterwards, to reduce this number and on the availability of redundant or dominated features.

Algorithm 16: A randomization method focused on the local search: select redundant features on a given solution, moving the incumbent solution to one with more improving neighbours.

Input: A feasible incumbent solution $\mathcal{F}\prime$, the number of features to select SZ_2
Output: A (potentially) different feasible solution
1 Let $\mathcal{F}^2 \subset \mathcal{F} \backslash \mathcal{F}\prime : u \in \mathcal{F} \backslash \mathcal{F}\prime$ is chosen at random, $|\mathcal{F}^2| = SZ_2$;
2 $select(\mathcal{F}^2)$;
`// A local search that would attempt to reduce the number of selected features is assumed to be performed afterwards`

5.4.2 Randomized Choices

Given that all proposed greedy algorithms will most likely perform different unsafe greedy choices, their obtained solutions could also be different. Moreover, none of them can guarantee that the obtained solution will be optimal, or better than others for all cases. Therefore, another way to introduce even more diversity to the methods described on Section 5.4.1 is to choose one such greedy algorithm at random.

It is interesting to notice that, if the possibly increased running times of the most complex methods, such as those that include safe reductions, do not contribute to solution quality,

choosing the simpler ones sporadically as well would not compromise the solution quality, as the most complex ones are still used as well, and would help to improve the iteration's running time.

It is also noteworthy that, sporadically starting from a solution of relatively worse quality also contributes to the diversity of the initial solutions pool, for the same reasoning presented on Section 5.4.1 for Algorithm 16.

5.4.3 Randomized Local Searches

As already discussed on Section 5.2 and Section 5.3, the order in which movements are performed affect the solution structure, and consequently the information that will be available to guide the next decision. Therefore, the order in which the available improving movements are performed may affect the quality of the local minimum obtained.

Therefore, in addition to the first and best improvement approaches, another strategy would be to randomize the local searches by electing the next improving movement to be performed randomly. That could introduce more diversity to the refined solutions, if many solutions are revisited and the election criterion has a significant impact on the final solution quality. Algorithm 17 depicts a randomized first-improvement local search approach.

5.5 Metaheuristic Design

In this work, two main metaheuristics are used and combined on the proposed implementations to help escaping local minima: Tabu Search and VNS.

5.5.1 VNS: Basic Design

The main heuristic framework used in this work is the Variable Neighbourhood Search, as proposed by Mladenovic and Hansen (1997). It is a metaheuristic used to solve combinatorial and global optimization problems whose basic idea is the systematic change of neighbourhood within a local search.

As depicted by Algorithm 18, in its simplest form, it includes an ordered list of neighbour-

Algorithm 17: A randomized swap-based first-improvement local search. It shuffles a list of possible candidate moves and performs the first improving swap that it finds until none can be found. Let RandomShuffle() be a function that randomly reorders an ordered set.

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the number of features to remove SZ_1, and include SZ_2 ($SZ_1 > SZ_2$)

Output: A feasible solution with (potentially) improved values of k, β and/or total covering

```
1  improved ← TRUE;
2  while improved do
3      LABEL: marker;
4      improved ← FALSE;
       // Let F' be an ordered set of selected features
5      RandomShuffle(F');
       // All possible subsets of F' of size SZ_1
6      forall F^1 ⊆ F' : |F^1| = SZ_1 do
           // All possible subsets of F\F' of size SZ_2
7          forall F^2 ⊆ F\F' : |F^2| = SZ_2 do
8              if isBetter(F'\F^1 ∪ F^2, F') then
9                  improved ← TRUE;
10                 swap(F^1, F^2);
11                 GOTO: marker;
```

hoods. The algorithm starts with an initial solution and, at each iteration, it is perturbed by a shake procedure associated to the current neighbourhood, which is followed by a local search. If the resulting solution is better than the best known, the best known solution is updated and the current neighbourhood is reset to the first defined. The current neighbourhood is set to the next in line otherwise.

Algorithm 18: A typical VNS algorithm in its simplest form. An ordered list of neighbourhoods $X = \{x_0, ..., x_{|X|-1}\}$ is assumed. On improvement the current neighbourhood is set to the first of the list. Upon a number of failures, the current neighbourhood is set to the next in line.

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the initial shake size SZ
Output: A near optimal feasible solution $\mathcal{F}^*\prime$
// Initial incumbent solution: any constructive greedy heuristic, such as Algorithm 10, Algorithm 11 or Algorithm 12
1 $greedy(\mathcal{F}\prime)$;
// The best known solution
2 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;
// Initial neighbourhood
3 $x \leftarrow x_0$;
4 **while** the stopping criterion is not met **do**
  // Let $shake_x$ be a perturbation defined on neighbourhood x. That could be SZ movements chosen at random in $X = \{x_0 = S_{1,0}, x_1 = S_{2,1}, x_2 = S_{3,1}, x_3 = S_{3,2}...\}$, such that the obtained solution is still feasible.
5 $shake_x(\mathcal{F}\prime, SZ)$;
  // Any local search procedure, such as Algorithm 13, Algorithm 14 or Algorithm 17
6 $search(\mathcal{F}\prime)$;
7 **if** $isBetter(\mathcal{F}\prime, \mathcal{F}^*\prime)$ **then**
8 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;
9 $x \leftarrow x_0$;
10 **else**
  // $nextn(x)$ returns the neighbourhood immediately after x on X. E.g.: $nextn(x_i) = x_{i+1}$
11 $x \leftarrow nextn(x)$;

A VNS design for the (α, β)-k-Feature Set approach using this scheme, would be to increase the range of the swap sizes allowed in the neighbourhood, allowing the local search inspection space to grow. In other words, the algorithm would start only with the redundancy tests, that is, attempting to remove features without selecting any other. Upon failure, on the

next iterations it would attempt to swap two selected features for one unselected, then three selected features for one unselected, then three selected for two selected and so on. At the beginning of each iteration a fixed shake procedure, such as random movements performed in this neighbourhood (e.g.: remove a few features and select a few others, without losing feasibility), such that the obtained solution is still feasible, would be used.

That, however, would be impractical for large instances as preliminary testing showed that using a $S_{2,1}$ neighbourhood increased the average computation time from a fraction of second to a few minutes. That suggests that, even with an aggressive candidate list reduction such local searches would still be too demanding and other alternatives have to be devised.

The Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start metaheuristic originally proposed by Feo and Resende (1995) that iteratively obtains a start solution with a randomized greedy algorithm, and tries to improve it with a local search. If the current solution is better than the best known, it is updated. Unlike VNS, the same neighbourhood structure is always searched, but the randomization of the greedy procedure allows the algorithm to explore several parts of the solution space by sampling.

The proposed implementation uses both the idea of having multiple start solutions, obtained by a randomized greedy method, of GRASP; and VNS's idea of exploring random solutions systematically farther from the best known. In other words, the "adaptive" part of GRASP is designed in a VNS fashion.

5.5.2 VNS: Varying the Number of Newly Attainable Feasible Solutions

Instead of changing the neighbourhood itself, an alternative would be to change the impact of the perturbation done by the shake procedure, but still using a similar mechanism. Algorithm 19 illustrates this approach, using the VNS mechanism to increment the shake size upon failure to improve the best known solution, and reseting the shake size to the initial upon success.

One possibility is to use Algorithm 16, which simply selects features randomly, modifying the number of random unselected features to be selected. It is important to notice that, the bigger the set of selected features, the bigger the number of candidates for deselection, which

could happen in different order. Therefore, even though the neighbourhood size does not change, the number of available improving neighbours does. That also progressively increases the computational time needed by the local search procedure, which could have to analyse a longer candidate list, but less aggressively than increasing the neighbourhood itself to a (combinatorially) larger one.

5.5.3 VNS: Varying the Number of Newly Attainable Infeasible Solutions

It is well known that the greedy algorithm usually provides good solutions for the Set Cover problem (Caserta, 2007). Also, preliminary testing showed that most of the times the exact approach solves the problem quickly, it is because it is solved to optimality on the root node. That suggests that a guide function based on instance properties only, such as constructive greedy heuristics like Algorithm 10, Algorithm 11 or Algorithm 12, for example; may indeed lead to good solutions.

Therefore, another possibility is to explore the infeasible solution space around good known solutions. In this work, this is done by deselecting features randomly, and restoring feasibility using a constructive method. Unless a suffix of the list of selected features is discarded, in order of selection, the greedy algorithm would hardly make the same choices, as the solution structure would probably be different.

As detailed in Algorithm 19, the number of features to deselect is determined with a mechanism similar to the neighbourhood selection of the VNS method: the algorithm begins deselecting only a few random features. Upon failure to improve the best known solution, it increases the number of features to be randomly deselected. On success to improve the best known solution, the number of features to be deselected is reset to the minimum. This is often regarded as Strategic Oscillation and is commonly used as a specialized procedure to Tabu Search. Also, even though the neighbourhood size under consideration also does not change, the number of candidates for removal do, varying the number of attainable neighbouring infeasible solutions.

Algorithm 19: Modified VNS algorithm. In this version, rather than moving to a different (larger) neighbourhood, the shake size is modified using a similar mechanism, controlling the number of solutions that can be attained after it. The algorithm starts with a small perturbation on the best known solution and, upon failure, increases the impact cause by the perturbation. On improvement, the perturbation size is reset to the first used.

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the initial shake size SZ_0 and shake size increase SZ_i

Output: A near optimal feasible solution $\mathcal{F}^*\prime$

```
// Initial incumbent solution: any constructive greedy procedure, such
   as Algorithm 10, Algorithm 11 or Algorithm 12
```

1 $greedy(\mathcal{F}\prime)$;

```
// The best known solution
```

2 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;

```
// Initialize the shake size
```

3 $SZ \leftarrow SZ_0$;

4 **while** the stopping criterion is not met **do**

```
   // Any of the proposed randomization methods: either Algorithm 16 or
      Algorithm 15
```

5 $shake(\mathcal{F}\prime, SZ)$;

```
   // Any local search procedure, such as Algorithm 13, Algorithm 14 or
      Algorithm 17
```

6 $search(\mathcal{F}\prime)$;

7 **if** $isBetter(\mathcal{F}\prime, \mathcal{F}^*\prime)$ **then**

8 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;

9 $SZ \leftarrow SZ_0$;

10 $SZ \leftarrow SZ + SZ_i$;

5.5.4 Tabu Search

It is important to notice at this point that even though cycling is avoided by only accepting solutions that do not change the number of selected features but improve the other two objective functions under consideration, visiting (and rejecting) the same solutions repeatedly could also occur and compromise the algorithm's performance.

A naive approach to address this problem is to remember all visited solutions and prevent the algorithm from revisiting them. As discussed on Chapter 4, (α, β)-k-Feature Set Problem instances are often highly symmetrical, which means that too many solutions could be visited during the algorithm's runtime. Even using very compact data structures, that could mean storing a huge amount of solutions, which not only would increase significantly the algorithm's memory requirements, and thus reduce the maximum size of tractable instances; but also that the huge list of visited solutions would also have to be inspected very frequently.

Therefore, in this work, this issue is addressed by using Tabu Search (Glover, 1990; Glover and Laguna, 1998; Glover et al., 1989). According to this framework, certain movements are forbidden for some iterations. This short term memory mechanism attempts to prevent that a solution that was recently visited be attainable before all the features that were selected in it are available or selected again. Hopefully the chance that it happens would be very low, and the greedy and search methods would be forced to perform different movements instead, leading to different solutions.

By marking movements (selections or deselections) tabu, an optimal solution might become unattainable. To avoid this situation, aspiration criteria are used. Usually, aspiration criteria involve accepting a solution either by objective, that is, if the solution is better; or direction, that is, if the direction of the search (improving or non-improving) does not change. In this work the policy of accepting solutions by objective is adopted, and the proposed algorithms implement this mechanism by not restricting the candidate movements of the local searches with the tabu list, as they all necessary improve the incumbent solution. Then, if the incumbent solution improves the best known, it is obviously accepted. The constructive stages, however, consider only movements that are not tabu, and therefore lead the initial solutions to different regions of the solution space. Also, if all features are marked tabu, the

features that are in the tabu list for longest are unmarked.

Algorithm 20 details a modified version of the proposed VNS scheme (Algorithm 19) that accounts also includes tabu search, as discussed on Section 5.5.4.

This mechanism relies on modified versions of the randomization methods that mark the randomly selected features tabu. Algorithm 22 and Algorithm 21 detail these procedures.

Also, considering that Algorithm 21 uses a greedy algorithm to restore the solution's feasibility, modified versions of the greedy procedures that do not attempt to re-select the features that were only just deselected are also necessary in this case. Algorithm 23, Algorithm 24 and Algorithm 25 detail these procedures. Note that, in these cases, it could be the case that all the features that could be chosen by greedy selection criterion are marked tabu. To address this situation, a new aspiration criterion has to be defined: if all eligible features are marked tabu, the one that best satisfies the greedy selection criterion, regardless of the tabu list, is chosen.

5.6 Implementation Remarks

As with any heuristic design, implementation is often a key aspect for its success. This section details some of the techniques use that had the biggest impact on the proposed algorithm's performance.

5.6.1 Partial Delta Evaluation

The first item worth noticing is partial delta evaluation, which regards analysing only the part of a solution's structure that changes after a movement and can have an impact on the objective function. Even though obvious, that is probably the most important aspect to be considered when designing the heuristic's data structures, as they should allow fast evaluations. For the particular problem under consideration, any feasible solution that is still feasible after swapping a set of selected features for a smaller set of unselected features is improving. To determine whether the solution will be feasible or not, one has to check if, after the movement, all sample pairs still have enough coverings to satisfy their demand. Clearly, only the sample

Algorithm 20: Modified VNS algorithm with Tabu Search. In this version, rather than moving to a different (larger) neighbourhood, the shake size is modified using a similar mechanism, controlling the number of solutions that can be attained after it. The algorithm starts with a small perturbation on the best known solution and, upon failure, increases the impact cause by the perturbation. On improvement, the perturbation size is reset to the first used. This method differs from Algorithm 19 because it maintain a tabu list that is used by the randomization and greedy procedures. A tabu tenure list with positive integers is maintained and decremented after each iteration. Every element in this list maps exactly one feature in $\mathcal{F}$. If the associated tabu tenure is positive, the feature is marked tabu. Every time a feature is selected or deselected by a greedy algorithm, at the beginning of the iteration, it is assigned a tabu tenure. Even though the local search may revert the feature's status (selected or unselected) due to the objective aspiration criterion, the greedy algorithm can perform the same movement again on subsequent iterations before its tabu tenure expires (reaches zero).

Input: A bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$, $\alpha^* < |\mathcal{F}|$, an incumbent feasible solution $\mathcal{F}\prime$, the initial shake size SZ_0 and shake size increase SZ_i

Output: A near optimal feasible solution $\mathcal{F}^*\prime$

```
// Initial incumbent solution: any constructive greedy procedure, such
   as Algorithm 10, Algorithm 11 or Algorithm 12
```

1 $greedy(\mathcal{F}\prime)$;

```
// The best known solution
```

2 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;

```
// Initialize the shake size
```

3 $SZ \leftarrow SZ_0$;

```
// Initialize the tabu tenures with zero
```

4 $\mathcal{T} \leftarrow 0$;

5 **while** the stopping criterion is not met **do**

```
    // Any of the proposed randomization methods: either Algorithm 22 or
       Algorithm 21
```

6 $shake(\mathcal{F}\prime, SZ)$;

```
    // Any local search procedure, such as Algorithm 13, Algorithm 14 or
       Algorithm 17. Note that this does not consider T because all
       considered movements are improving with respect to the incumbent
       solution, and therefore are accepted due to the aspiration
       criterion.
```

7 $search(\mathcal{F}\prime)$;

8 **if** $isBetter(\mathcal{F}\prime, \mathcal{F}^*\prime)$ **then**

9 $\mathcal{F}^*\prime \leftarrow \mathcal{F}\prime$;

10 $SZ \leftarrow SZ_0$;

11 $SZ \leftarrow SZ + SZ_i$;

```
    // Tenure maintenance
```

12 LABEL: *marker*;

13 **foreach** $t_i \in \mathcal{T}$ **do**

14 $t_i \leftarrow \max\{0, t_i - 1\}$;

15 **if** $t_i > 0 \forall t \in \mathcal{T}$ **then**

16 GOTO *marker*;

Algorithm 21: A randomization method focused on the greedy algorithm: deselect a few features chosen at random, and use a greedy heuristic to make it feasible again. This allows the algorithm to explore more of the infeasible region of the solution space. This method differs from Algorithm 15 because it does not allow features marked tabu to be deselected.

Input: A feasible incumbent solution $\mathcal{F}\prime$, the number of features to deselect SZ_1, tabu tenures $\mathcal{T}$

Output: A (potentially) different feasible solution

```
// The set of features marked tabu
```

$\mathcal{T}\prime \leftarrow \{f_i \in \mathcal{F} : t_i \in \mathcal{T} > 0\}$;

Let $\mathcal{F}^1 \subset \mathcal{F}\prime : f \in \mathcal{F}\prime \backslash \mathcal{T}\prime$ is chosen at random;

$deselect(\mathcal{F}^1)$, $|\mathcal{F}^1| = SZ_1$;

foreach $f_i \in \mathcal{F}^1$ **do** `// Mark them all tabu`

```
    // Assigns a random positive tabu tenure not greater than the maximum
       allowed
```

    $t_i \in \mathcal{T} \leftarrow rand(1, SZ_t)$;

```
// Either Algorithm 23, Algorithm 24 or Algorithm 25
```

$greedy(\mathcal{F}\prime)$;

Algorithm 22: A randomization method focused on the local search: select redundant features on a given solution, moving the incumbent solution to one with more improving neighbours. Provided that the start solution is feasible, this works only with feasible solutions. This method differs from Algorithm 16 because it does not allow features marked tabu to be selected.

Input: A feasible incumbent solution $\mathcal{F}\prime$, the number of features to select SZ_2, tabu tenures $\mathcal{T}$, maximum tabu tenure SZ_t

Output: A (potentially) different feasible solution

```
// The set of features marked tabu
```

$\mathcal{T}\prime \leftarrow \{f_i \in \mathcal{F} : t_i \in \mathcal{T} > 0\}$;

Let $\mathcal{F}^2 \subset \mathcal{F} \backslash \mathcal{F}\prime : u \in \mathcal{F} \backslash \mathcal{F}\prime \backslash \mathcal{T}\prime$ is chosen at random, $|\mathcal{F}^2| = SZ_2$;

$select(\mathcal{F}^2)$;

foreach $f_i \in \mathcal{F}^2$ **do** `// Mark them all tabu`

```
    // Assigns a random positive tabu tenure not greater than the maximum
       allowed
```

    $t_i \in \mathcal{T} \leftarrow rand(1, SZ_t)$;

```
// A local search that would attempt to reduce the number of selected
   features is assumed to be performed afterwards
```

Algorithm 23: Greedy Algorithm 1 considering Tabu Search: always choose the unselected feature that has more neighbours with unsatisfied covering requirements. This method attempts to make the solution feasible using as few features as possible. It differs from Algorithm 10 because it does not allow features marked tabu to be selected, unless strictly necessary.

Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
// The set of features marked tabu
```
$\mathcal{T}\prime \leftarrow \{f_i \in \mathcal{F} : t_i \in \mathcal{T} > 0\}$;
while $\exists u \in \mathcal{A} : d_u > 0$ **do** `// The solution is not feasible`
  `// select the unselected feature that is not tabu and has more neighbours in` $\mathcal{A}$ `with unsatisfied covering requirements`
  $f \leftarrow \max_{f \in \mathcal{F} \setminus \mathcal{F}\prime \setminus \mathcal{T}\prime}\{|\{u \in N(f) \cap \mathcal{A} : d_u > 0\}|\}$ `// if no such feature exists, disconsider the tabu list`
  if $f = \emptyset$ **then**
    $f \leftarrow \max_{f \in \mathcal{F} \setminus \mathcal{F}\prime}\{|\{u \in N(f) \cap \mathcal{A} : d_u > 0\}|\}$;
  select(f);

Algorithm 24: Greedy Algorithm 2 considering Tabu Search: always choose the unselected feature that has more neighbours with unsatisfied covering requirements and covers the vertex with least coverings.This method attempts to make the solution feasible using as few features as possible, focusing on the most infeasible part of the solution. It differs from Algorithm 11 because it does not allow features marked tabu to be selected, unless strictly necessary.

Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
// The set of features marked tabu
```
$\mathcal{T}\prime \leftarrow \{f_i \in \mathcal{F} : t_i \in \mathcal{T} > 0\}$;
while $\exists u \in \mathcal{A} : d_u > 0$ **do** `// The solution is not feasible`
  `// find sample pair that needs more coverings`
  u $\leftarrow max_{u \in \mathcal{A}}\{d_u\}$;
  `// select the unselected feature, that is not tabu, that has more neighbours with unsatisfied covering requirements and covers the vertex with least coverings`
  $f \leftarrow \max_{f \in N(u) \cap (\mathcal{F} \setminus \mathcal{F}\prime) \setminus \mathcal{T}\prime}\{|\{v \in N(f) \cap \mathcal{A} : d_v > 0\}|\}$ `// if no such feature exists, disconsider the tabu list`
  if $f = \emptyset$ **then**
    $f \leftarrow \max_{f \in N(u) \cap (\mathcal{F} \setminus \mathcal{F}\prime)}\{|\{v \in N(f) \cap \mathcal{A} : d_v > 0\}|\}$;
  select(f);

pairs that are neighbours of the involved features are affected. Moreover, only the sample pairs that lose coverings can make the solution infeasible. Therefore, it is enough to verify

Algorithm 25: Greedy Algorithm 3 considering Tabu Search: accounting for safe reductions. After reducing the current subproblem to a kernel, choose the unselected feature with most unselected neighbours that covers the vertex with least coverings. This method attempts to make the solution feasible using as few features as possible, focusing on the most infeasible part of the solution, and always checking for safe reductions. It differs from Algorithm 12 because it does not allow features marked tabu to be selected, unless strictly necessary.

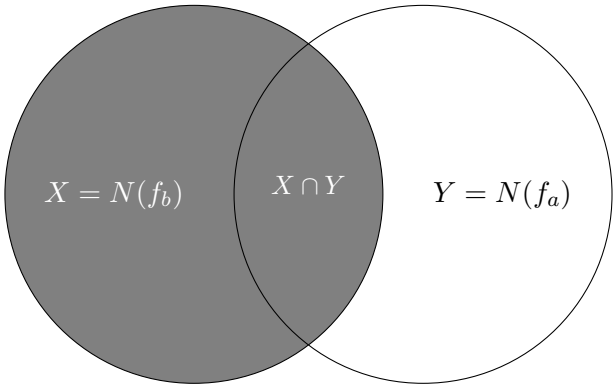
Input: The set of selected features $\mathcal{F}\prime$, a bipartite graph $G = (\mathcal{A}, \mathcal{F}, \mathcal{E})$ and $\alpha^* < |\mathcal{F}|$
Output: A feasible solution

```
// The set of features marked tabu
```
$\mathcal{T}\prime \leftarrow \{f_i \in \mathcal{F} : t_i \in \mathcal{T} > 0\}$;
while $\exists u \in \mathcal{A} : d_u > 0$ **do** `// The solution is not feasible`
  `// perform safe reductions`
  run(Algorithm 2);
  `// find sample pair that needs more coverings`
  u $\leftarrow max_{u \in \mathcal{A}}\{d_u\}$;
  `// select the unselected feature, that is not tabu, that has more neighbours with unsatisfied covering requirements and covers the vertex with least coverings`
  $f \leftarrow \max_{f \in N(u) \cap (\mathcal{F} \setminus \mathcal{F}\prime) \setminus \mathcal{T}\prime}\{|\{v \in N(f) \cap \mathcal{A} : d_v > 0\}|\}$ `// if no such feature exists, disconsider the tabu list`
  if $f = \emptyset$ **then**
    $f \leftarrow \max_{f \in N(u) \cap (\mathcal{F} \setminus \mathcal{F}\prime)}\{|\{v \in N(f) \cap \mathcal{A} : d_v > 0\}|\}$;
  select(f);

the coverings only of these sample pairs when evaluating whether a movement is improving or not.

Figure 5.2 and Figure 5.3 illustrate swapping one feature for another and two features for one respectively, and its affected sample pairs.

Figure 5.2: **Affected sample pairs after swapping 2 features** given a set of selected features, substitute one feature ($f_a \in \mathcal{F}^1$) for another ($f_b \in \mathcal{F}^2$). Sets X and Y represent the neighbours of features f_b and f_a, respectively. Only the covering of sample pairs in the white areas have to be checked when evaluating the solution for feasibility.



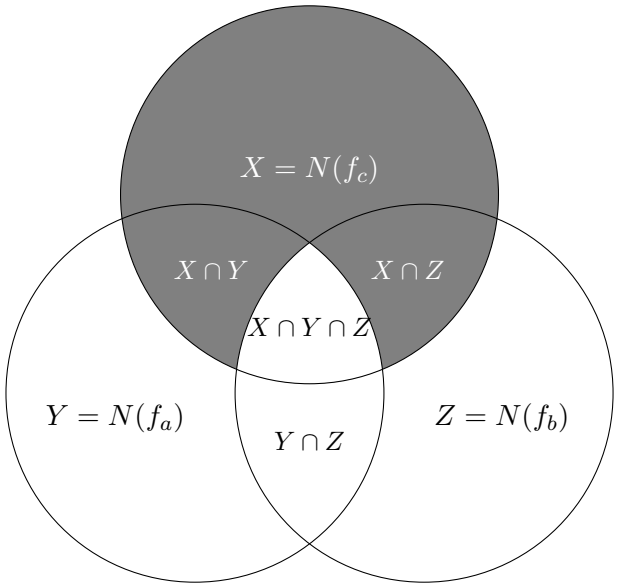
Delta evaluation remark: A solution will still be feasible after swapping one feature f_a for a feature f_b if all the neighbours of f_a that are not common to f_b have at least one extra covering. Since with the introduction of f_b the features that are neighbours both of f_b and f_a recover the covering that they lose with the removal of f_a, they do not have to be checked on evaluation.

■

5.6.2 Initial Solution

In order to obtain an initial solution for the VNS framework, any greedy algorithm such as those proposed on Section 5.2 would suffice. However, from the implementation point of view, it is advisable that such a solution be as good as possible. That is not only because it would reduce the number of iterations of the method, when the stopping criteria involves the number of consecutive unsuccessful iterations, but also because of delta evaluation. Provided one has a good quality solution, the chance that the evaluation of a bad solution is stopped early is bigger because the best known already yields stronger bounds that are more difficult to be

Figure 5.3: **Affected sample pairs after swapping 2 features for 1:** given a feature set, substitute two features ($f_a, f_b \in \mathcal{F}^1$) for one ($f_c \in$)$\mathcal{F}^2$). Sets X, Y and Z represent the neighbours of features f_c, f_a and f_b, respectively. Only the covering of sample pairs in the white areas have to be checked when evaluating the solution for feasibility.



Delta evaluation remark: A solution that is still feasible after such a movement is always improving because it yields a smaller (and therefore more interesting) selected feature set. Only neighbours of the involved features will have their covering status changed. Also, only features that lose coverings may affect the feasibility of the solution. Neighbours of the features that will no longer be selected would lose exactly one covering while neighbours of the features that will be selected after the movement would gain exactly one covering. Therefore, the number of coverings to be gained or lost can be determined for by inspecting common neighbours. If the current covering, accounting for the movement effects, of every neighbour of the features that are being deselected is enough to satisfy its requirements the resulting solution is still feasible.

■

satisfied or even improved.

Provided that the available greedy procedures obtain different solutions, and assuming that their running time is sufficiently low, one way to start with a better quality solution, which is used on the proposed implementation, is to run them all and start the VNS algorithm with the best obtained solution as best known and incumbent solution.

5.6.3 Data Structures

As already suggested on Section 3.4 and Section 5.3, most of the expensive operations used in the proposed algorithm depend on set operations. Even though a binary search tree variant would be the standard choice for such operations, the proposed implementation uses bitarrays for a better use of memory space, equivalent ease to perform set operations using logical operators and due to the fact that it is possible to exploit architecture specific instructions to perform such operations faster than if using standard methods. A collection of optimized bit operations is hosted by Sean Eron Anderson at `http://graphics.stanford.edu/~seander/bithacks.html`.

Also regarding the memory requirements, it is interesting to notice that even though Set (Multi) Cover instances derived from case-control datasets with the methodology described in Section 3.1 have an immediate memory requirement of the order of $O(n \times \binom{m}{2})$ (to fit the instance, disregarding the algorithm's data structures), they can be represented using only $O(n \times m)$ using the case-control format, if they are also (α, β)-k-Feature Set Problem instances. Therefore, instead of simply running through the sample pairs in $\mathcal{A}$ and $\mathcal{B}$, one can run through the combinations of samples and testing to determine with they belong to the same or different classes. The proposed implementation expands the case-control dataset into a Set (Multi) Cover instance to favour speed, but using the original case-control format would still be an interesting alternative, should the memory requirements be more critical than speed. That, however, would add the overhead of the recurrent class test, and compromises the implementation of optimizations for set operations, as the nodes neighbours would no longer be explicit. It also would save relatively little space, as the coverings for each pair of samples have to be kept in memory as well.

5.6.4 Caching

Caching is also an important item to be noticed, specially for problems that contain a solution substructure. However, that should be used with caution, as very large datasets might be challenging to store in memory already, without accounting for cached data. The work of Hua et al. (2009, 2010) illustrates this point well, as they managed to isolate a substructure in the Multi Set Multi Cover problem, and used it to propose dynamic programming algorithms

both for the Set Multi Cover and Multi Set Multi Cover problems. That led to an enumerative algorithm with polynomial space requirements and exponential time requirements. They claim it to be theoretically the fastest exact method available but do not provide computational results, nor a benchmark testbed. The enumerative, exponential time demanding, characteristic of their method would make it prohibitive for the target problems considered in this work, but the efficient use of caching (the dynamic programming design) to speed it up to the state-of-art highlight the importance of the technique in this scenario. In this work, feature rankings (number of unselected neighbours of the sample pair) are cached as they are very frequently needed by the greedy heuristics. These values do not change frequently and when they do, it is not necessary that all other values are updated. Since a sample pair might have many neighbours, running this search only when the it is updated sped up the greedy heuristics up to ten times.

5.6.5 Parallelization

Finally there is the matter of parallelism. In this work, it was attempted in two different ways. In the first, the focus was on the local searches, which are the most time demanding procedures of the proposed methods. They were parallelized by delegating a disjoint equal part of the search space to each available processor. However, the speed up obtained was not significant.

The second attempt regarded diversification. With this parallelization, every processor has its own incumbent solution, shares the best solution and performs the exact same job, with different random seeds. Because each thread is given different random seeds, the start solutions are also often different, allowing different threads to search different areas of the solution space, covering a larger diversity of solutions concurrently. Because of the increased diversity and amount of evaluated solutions, this parallelization of the algorithm allowed better quality final results. Even though the algorithm took about the same time to meet the stopping criterion, or even more, since it successfully found improved solutions more often; the time to reach a target solution was typically shorter.

5.7 Conclusions

In the previous chapter, it was possible to obtain a very diverse set of instances that can not be solved by the exact approach. It was also possible to see that it such difficult instances may also be found in real life (see SM, in Table 4.1, for example). One interesting observation, however, was that the computational effort required to solve each of the four stages of the (α, β)-k-Feature Set Approach could be significantly different. Also, the solution of the later stages is directly related to the solution obtained on the earlier stages.

To address the first research question of this thesis, "Is it possible to quickly select features on highly dimensional datasets, without compromising the robustness of the solution?", this chapter detailed five different methods. They combine different constructive heuristics and local search procedures in modern meta-heuristic frameworks: GRASP, VNS and Tabu Search. Since the proposed algorithms attempt to optimise all objectives simultaneously, they have the direct advantage of not requiring that any stage be solved to optimality before the other objectives are improved, quickly obtaining better overall quality suboptimal solutions whenever the exact approach does not suffice. Since the whole feature selection approach is also dealt with in one optimisation process, more information gathered for of each stage of the (α, β)-k-Feature Set Approach is shared and exploited in the solution process of all subsequent others.

The main drawback of the proposed methods, however, is that they do not provide a quality estimate by themselves, leaving the second research question "Is it possible to quickly estimate how good these solutions would be?" to be addressed in the next chapter.

Chapter 6

Solution Quality and Dual Bounds

In order to prove the optimality of a solution, two approaches are usually considered: to enumerate all feasible solutions of the problem and pick the best; and/or to obtain upper and lower bounds for the problem such that their objective function values match.

The first approach alone is clearly impractical for most NP-Complete problems. The latter depends on the possibility to obtain a decreasing sequence of upper bounds and an increasing sequence of lower bounds. Still, the best upper and lower bounds that one can obtain may not always match. Nevertheless, the distance between such bounds can be taken as a quality estimate. Most modern solvers implement a combination of both approaches to prove optimality and guide their searches.

Any feasible solution of the original problem is a Primal bound. Finding Dual Bounds, however, presents a different challenge, for which one of the most important approaches are *Relaxations*. The idea behind those is to replace a difficult problem with a relaxed one, that can be dealt with in practice, whose optimal value is at least as large, in the case of maximization problems, or as small, for minimization problems; as the original.

One of the most popular relaxations are the Linear Programming relaxations, in which the integrality constraints are dropped, allowing all the variables to assume real values. The Integer Program becomes thus a Linear Program that can be solved in polynomial time.

One usually regards upper bounds for minimization problems and lower bounds for maximization problems as *Primal Bounds*. Similarly, lower bounds for minimization problems and

upper bounds for maximisation problems are called *Dual Bounds*.

The dual bounds used to calculate the gaps shown on the Computational Results section of the previous chapter (Section 7.1) were obtained using the exact approach, with the time limit set to one hour. The exact approach relies on an Integer Program solver that uses the linear relaxation to obtain bounds, and improves them using methods that are out of the scope of this text. Even though these bounds would suffice to estimate the quality of the solutions, obtaining a good dual bound using this method could be rather expensive. Specially considering that the root relaxation, which is the only dual bound obtained fairly quickly and without branching, is usually of poor quality for the (α, β)-k-Feature Set approach problems. Also, since the methods proposed in this work do not solve the Integer Program directly, extra space would be required to store the models and data structures used by the solver, which could be quite significant, considering the target instance sizes. Therefore, this chapter details alternative procedures to obtain dual bounds for the three last steps of the (α, β)-k-Feature Set approach (as the first is solved to optimality), that do not rely on any external solvers and share data structures.

Note, however, since that the models used usually yield poor bounds for the target type of instances, with many more sample pairs than features (Caserta, 2007), the objective pursued here is not to obtain tighter gaps than the current and/or prove optimality, but to provide a quick quality estimate. As discussed on Section 1.2, for the target (biological) applications, since the problems solved are only models of the real case, and the solutions obtained still have to be validated using other techniques, they may not be good or even feasible, even if they are the optimal solution for the model under consideration. In that case, obtaining solutions and quality estimates quickly and efficiently is more important than obtaining tighter gaps that could or could not be significant. That is specially true considering the target instance sizes, for which exact and slower methods would not be able to obtain any bounds within reasonable time.

Moreover, as it was observed on Chapter 4, many times the best known gaps are already not ideal and require a significant computational effort to be obtained, even though they rely on state-of-art techniques. However, depending on the expected objective function value

range, gaps may not even be very descriptive of the solution quality. In particular, both small differences in practice or small percent differences may not be worthwhile. Therefore, in practice, a quick approach that obtains a reasonably competitive gap would be much more appealing in practice than a very demanding method that always outperforms the first. With that in mind, the main objective pursued in this chapter is to propose methods that would excel in practice due to speed and memory requirements.

Section 6.1 starts by proposing quick bounds, that can be obtained constructively, for the two last stages of the (α, β)-k-Feature Set approach. Section 6.2 then attempts to improve these bounds and obtain bounds also for the second stage using a different, classic, technique: Lagrangian Relaxation. Finally, Section 6.3 proposes dual models for all the stages that can be explored on further research either to obtain bounds for the problems or devise new methods to solve them. Section 8.2 then concludes this chapter.

6.1 Greedy Dual Bounds

The first bounds to be studied in this chapter are obtained in a greedy fashion, by solving a very easy relaxation of the associated models. They can be obtained very quickly and can be used as a rough estimate of the quality of the obtained primal solutions.

6.1.1 Lower bound for Min k

The first dual bound to be discussed is one for the minimum number of selected features. By definition, in any feasible solution of the Set Multi Cover problem all sample pairs must have their covering requirements met. Also, every feature can only cover any given sample at most once. Therefore, considering that the Set Multi Cover problem under consideration requires that each sample pair must be covered at least α times, at least α different features are necessary to do so, or else either the solution is infeasible for not covering each sample pair enough times, or some features are covering the same sample pair more than once.

That makes α a lower bound for k that is immediately available after the first stage of the (α, β)-k-Feature Set approach is completed. This problem can be modelled as shown in

Figure 3.4, and this Integer Program can be solved in polynomial time, and requires no extra memory.

6.1.2 Upper bound for Max β

Analysing the mathematical models for the maximisation of α (Figure 3.4) and β (Figure 3.6) it is interesting to notice that, disregarding the constraint that fixes the number of selected features (row (3.6d)), the problems of satisfying the covering requirements of the nodes either in set $\mathcal{A}$ or $\mathcal{B}$ (rows (3.6b) and rows (3.6c)) are completely independent. That is, if it does not matter how many features can be used, the minimum covering requirement for sample pairs in $\mathcal{A}$ also does not matter to determine the minimum covering requirement for sample pairs in $\mathcal{B}$ because one can always use extra features to do so. Therefore, if one discards row (3.6d), rows (3.6b) become redundant and can also be discarded. The resulting problem is then exactly the same one depicted in Figure 3.4, swapping set $\mathcal{A}$ for $\mathcal{B}$.

Because the relaxed problem maximises the same objective function on a larger set of feasible solutions, its optimal solution to this problem provides an upper bound for the problem of finding the Maximal β.

As already discussed, the model depicted on Figure 3.4, and therefore also this relaxation, can be solved by inspection in polynomial time, and requires no extra memory.

Even though quick, this method relies on a relaxation that discards row (3.6d), which is a very restrictive constraint of the model. That could compromise the quality the obtained bounds. Moreover, it does not consider any of the arguments of the original model, α and k, which also not only could have a negative impact on the quality of the obtained bounds, but it means that it can not be improved as k is improved.

6.1.3 Upper bound for the Max Total Covering

Consider now the problem of maximising the total covering. This problem attempts to find exactly k features that satisfy the covering requirements determined on the first stage of the (α, β)-k-Feature Set approach and yield a maximal total coverage. Clearly, the maximum total coverage that can be obtained on any given bipartite graph, using only k sample pairs from

one side and disconsidering any covering requirement, is the sum of the k highest node degrees in $\mathcal{F}$, for if any of these vertices is swapped for another, the newly selected one would have a smaller degree and the solution would therefore yield a smaller total coverage. Since no bigger total covering value can be obtained, this procedure provides an upper bound for the problem of determining the maximal total covering.

This bound can be obtained in $O(n \log n)$, since all one has to do is sort the set of available features in decreasing order of degrees. As k is updated, the described procedure can be implemented in $O(1)$ simply by subtracting the degree of the kth node on the sorted list from the current bound.

6.2 Lagrangian Relaxation

Another important method for obtaining dual bounds is the Lagrangian Relaxation, in which the idea is to drop complicating constraints and add them to the objective function with a non-negative penalty term called Lagrange Multiplier. That leads to a simpler (sub)problem because the complicating constraints do not have to be satisfied any more. Since this problem takes the Lagrange Multiplier as parameters, the Lagrangian Dual problem would then become that of finding the Lagrange Multipliers that yield the best bounds. It is possible to show that the best bound obtained by solving the Lagrangian Dual Problem is at least as good as the one obtained by solving the Linear Programming relaxation.

To guarantee that the resulting problem is indeed a relaxation, its objective function must have values at least as big (for maximization problems) or small (for minimization problems) as the original. That is easily achieved by observing the sign of the constraints being dropped and adding the penalised term accordingly. For example, consider the Integer Program $z = \max\{cx : Dx \leq d, x \in X\}$, and suppose that the constraints $Dx \leq d$ make the problem difficult. A Lagrangian Relaxation of this problem with Lagrange multipliers u would be $z(u) = max\{cx + u(d - Dx), x \in X\}$ and would yield an optimal solution $x(u)$. The best bound w_{LD} would be obtained by solving $w_{LD} = \min\{z(u) : u \geq 0\}$. The objective function value of the optimal solution of the relaxed problem is always at least as big as that of the

original problem because $d - Dx \geq 0$ and $u \geq 0$.

For further information about Duality, Bounds and Lagrangian Relaxations please refer to Fisher (1981); Guignard (1998, 2003); Held and Karp (1970, 1971); Nemhauser and Wolsey (1989); Wolsey (1998).

6.2.1 The Subgradient Method

It is possible to show that the Lagrangian relaxation $z(u)$ of a problem is a piecewise linear convex non-differentiable function. Also, it is possible to show that the penalized terms added to the original objective function to define the Lagrangian relaxation (i.e.: $d - Dx(u)$, on the previous example) define a subgradient sg of $z(u)$ (Wolsey, 1998). Therefore, the Lagrangian Dual Problem can be solved by a straightforward generalization of the Gradient Algorithm: the Subgradient Algorithm.

Algorithm 26: Subgradient algorithm. Let sg^i be a subgradient of $z(u^i)$ and μ^i be the step size elected for iteration i. At each iteration, compute and take a step on the direction of the subgradient. The main difficulty of this method is to determine the step size.

Input: An instance of the problem under consideration
Output: A (Lagrangian) dual bound w_{LD}
`/* Set all entries to zero */`
1 Init u^0;
`/* Iteration` i `*/`
2 **while** Stopping Condition is not met **do**
  `/* Solve the relaxed subproblem */`
3 Find $z(u^i)$;
  `/* Move on the direction of the subgradient */`
4 $u^i \leftarrow \max\{u^i - \mu^i sg^i), 0\}$;
5 $i \leftarrow i + 1$;

Since there is no way to prove that the obtained bound is optimal, Algorithm 26 usually stops either after a certain number of iterations, a specified time limit, number of iterations without improvement, or if the obtained lower and upper bounds match.

The main difficulty of this method is to choose step sizes for each iteration, and there are various ways to do so (see Wolsey, 1998, for a few examples). In this work, the same method used in Pessoa et al. (2010), and suggested by Wolsey (1998), is used.

Figure 6.1: Lagrangian Relaxation of the Min k (α, β)-k-Feature Set Problem.

$$z(\lambda, \alpha) = \min \sum_{s_f \in \mathcal{F}} x_{s_f} + \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} (\alpha - \sum_{s_f \in \mathcal{F}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} x_{s_f}) \tag{6.1a}$$

$$= \min \sum_{s_f \in \mathcal{F}} x_{s_f} + \alpha \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} - \sum_{u_{p,q} \in \mathcal{A}} \sum_{s_f \in \mathcal{F}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} \lambda_{u_{p,q}} \tag{6.1b}$$

$$= \min \sum_{s_f \in \mathcal{F}} (1 - \sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) x_{s_f} + \alpha \sum_{j \in A} \lambda_{u_{p,q}} \tag{6.1c}$$

$$x_{s_f} \in \{0, 1\} \tag{6.1d}$$

Considering a maximisation problem, if $\overline{w}$ is a dual upper bound on w_{LD}, $0 < \epsilon_k < 2$ and $\mu^i = \epsilon^k [z(u^k) - \overline{w}]/||sg^i||^2$; then $z(u^i) \rightarrow \overline{w}$ or the Subgradient algorithm finds u^i with $z(u^i) \geq \overline{w} \geq w_{LD}$ within a finite number of iterations. Similarly, if $\underline{w}$ is a dual lower bound on w_{LD}, $0 < \epsilon_k < 2$ and $\mu^i = \epsilon^k [\underline{w} - z(u^k)]/||sg^i||^2$; then $z(u^i) \rightarrow \underline{w}$ or the Subgradient algorithm finds u^i with $z(u^i) \leq \underline{w} \leq w_{LD}$ within a finite number of iterations for minimisation problems. The difficulty here is that dual bounds $\overline{w}$ or $\underline{w}$ are typically unknown and a primal bound ($\underline{w} \leq w_{LD}$ for maximisation problems and $\overline{w} \geq w_{LD}$ for minimisation problems) is used in its place. However, if the term $z(u^k) - \underline{w}$ on maximisation problems does not tend to zero ($\underline{w} < w_{LD}$), $\underline{w}$ has to be increased. Similarly if the term $z(u^k) - \overline{w}$ on maximisation problems does not tend to zero ($\overline{w} > w_{LD}$), $\underline{w}$ has to be decreased (Wolsey, 1998).

6.2.2 Lower Bounds for Min k

Consider the problem of minimizing the number of selected features, as modelled on Figure 3.5. As already mentioned, this problem is the Minimum Cardinality Set Multi Cover Problem, with unitary costs. Following with the same idea of Beasley and Jørnsten (1992); Beasley (1990a); Caprara et al. (1999); Caserta (2007); Ceria et al. (1998); Pessoa et al. (2011), dualizing row (3.5b), the resulting problem, depicted on Figure 6.1, is a Lagrangian Relaxation of the Min k (α, β)-k-Feature Set Problem.

Since the only differences between this problem and the aforementioned Set Cover Lagrangian relaxation are the constants, this problem can also be easily solved to optimality by

inspection in polynomial time by setting:

$$x_f = \begin{cases} 1 & \text{if}(1 - \sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) < 0 \\ 0 & \text{if}(1 - \sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) > 0 \\ \in \{0,1\} & \text{if}(1 - \sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) = 0 \end{cases}$$

For the subgradient method, $\alpha - \sum_{s_f \in \mathcal{F}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} x_{s_f}$ is a subgradient of $z(\lambda, \alpha)$.

6.2.3 Upper Bounds for Max β

Still following the idea discussed on Section 6.1.2, dualizing rows (3.6b) and (3.6c) from Figure 3.6, and keeping row (3.6d), the model depicted on Figure 6.2 is a Lagrangian Relaxation of the Max β (α, β)-k-Feature Set Problem.

These subproblems are also easily solvable in polynomial time.

Since this is a maximisation problem, the optimal value of β is defined by:

$$\beta = \begin{cases} |F| & \text{if } (1 - \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}}) > 0 \\ 0 & \text{if } (1 - \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}}) < 0 \\ \{0..|F|\} & \text{if } (1 - \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}}) = 0 \end{cases}$$

Considering that $\beta \geq 0$, if $(1 - \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}})$ positive, it only increases the objective function value and should be exploited as much as possible. Therefore β is set to its upper bound: $\beta = |F|$. If it is negative, the associated penalty should be minimised so, β is set it to its lower bound: $\beta = 0$. If it is zero, it does not add or penalise the dual bound. So it does not matter what value is chosen for β.

Regarding x, it is a simple matter of choosing the k features that contribute to the objective function value most or, in the worst case, compromise the objective function least. Therefore, to solve this part of the subproblem, it suffices to calculate $(\sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \mu_{u_{p,q}})$ for every feature, sort them in decreasing order of these values, and set the k highest ranked features to 1 and all the others to 0. The other terms of the expression are fixed, so they do not depend on the solution.

The upper bound is obtained by calculating the objective function value, using the optimal

Figure 6.2: Lagrangian Relaxation of the Max β (α, β)-k-Feature Set Problem.

$$z(\lambda, \mu, \alpha, k) = \max \beta + \sum_{u_{p,q} \in \mathcal{A}} (\sum_{s_f \in \mathcal{F}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} - \alpha) \lambda_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}} (\sum_{s_f \in \mathcal{F}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} - \beta) \mu_{u_{p,q}} \tag{6.2a}$$

$$= \max \beta + \sum_{s_f \in \mathcal{F}} \sum_{u_{p,q} \in \mathcal{A}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} \lambda_{u_{p,q}} + \sum_{s_f \in \mathcal{F}} \sum_{u_{p,q} \in \mathcal{B}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} \mu_{u_{p,q}} - \alpha \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} - \beta \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}} \tag{6.2b}$$

$$= \max (1 - \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}}) \beta + \sum_{s_f \in \mathcal{F}} (\sum_{u_{p,q} \in \mathcal{A}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} \mu_{u_{p,q}}) x_{s_f} - \alpha \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} \tag{6.2c}$$

s.t.

$$\sum_{s_f \in \mathcal{F}} x_{s_f} = k \tag{6.2d}$$

$$x_{s_f} \in \{0, 1\} \tag{6.2e}$$

$$\beta \in \{0..|F|\} \tag{6.2f}$$

■

solution of the subproblem, with Equation 6.2a.

In this case, $\sum_{s_f \in \mathcal{F}:\exists e=(s_f,u_{p,q})\in\mathcal{E}} x_{s_f} - \alpha + \sum_{s_f \in \mathcal{F}:\exists e=(s_f,u_{p,q})\in\mathcal{E}} x_{s_f} - \beta$ is a subgradient of $z(\lambda, \mu, \alpha, k)$.

6.2.4 Upper Bounds for Max Cover

Dualizing Equation 3.7b and Equation 3.7c from Figure 3.7, Figure 6.3 depicts a Lagrangian Relaxation of the Max Cover (α, β)-k-Feature Set Problem.

These subproblems are solved in the same manner as the previous ones, considering that the value of β is already fixed. Since the objective function is different, the reduced costs also have to be recalculated for every feature s_f as $(d_{s_f} + \sum_{u_{p,q}\in\mathcal{A}:\exists e=(s_f,u_{p,q})\in\mathcal{E}} \lambda_{u_{p,q}} + \sum_{u_{p,q}\in\mathcal{B}:\exists e=(s_f,u_{p,q})\in\mathcal{E}} \mu_{u_{p,q}})$.

The upper bound is obtained by calculating the objective function value, using the optimal solution of the subproblem, with Equation 6.3a.

In this case, $\sum_{s_f \in \mathcal{F}:\exists e=(s_f,u_{p,q})\in\mathcal{E})} x_{s_f} - \alpha + \sum_{s_f \in \mathcal{F}:\exists e=(s_f,u_{p,q})\in\mathcal{E}} x_{s_f} - \beta$ is a subgradient of $z(\lambda, \mu, \alpha, \beta, k)$.

6.2.5 Implementation Remarks

6.2.5.1 Integer Rounding

Note that, from the integrality of the variables and data, valid objective function values must also be integer. Since the Lagrangian multipliers are real values, their sum is also a real value. Therefore, an obvious way to (slightly) improve the obtained bounds is to consider their floor for maximisation problems and ceiling for minimization problems.

Also notice that, specially on the later iterations, as the step size taken on the subgradient method gets smaller, the bounds tend to improve slower because the change on the Lagrangian multipliers gets smaller. Therefore considering the integer rounding of the bounds forces that the step size is adjusted taking into consideration a more practical bound, avoiding exceedingly small changes.

Figure 6.3: Lagrangian Relaxation of the Max Cover (α, β)-k-Feature Set Problem.

$$z(\lambda, \mu, \alpha, \beta, k, d_i) = \max \sum_{s_f \in \mathcal{F}} d_{s_f} x_{s_f} + \sum_{u_{p,q} \in \mathcal{A}} (\sum_{s_f \in \mathcal{F}: \exists e = (s_f, u_{p,q}) \in \mathcal{E})} x_{s_f} - \alpha) \lambda_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}} (\sum_{s_f \in \mathcal{F}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} - \beta) \mu_{u_{p,q}} \tag{6.3a}$$

$$= \max \sum_{s_f \in \mathcal{F}} d_{s_f} x_{s_f} + \sum_{s_f \in \mathcal{F}} \sum_{u_{p,q} \in \mathcal{A}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} \lambda_{u_{p,q}} + \sum_{s_f \in \mathcal{F}} \sum_{u_{p,q} \in \mathcal{B}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} x_{s_f} \mu_{u_{p,q}} - \alpha \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} - \beta \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}} \tag{6.3b}$$

$$= \max \sum_{s_f \in \mathcal{F}} (d_{s_f} + \sum_{u_{p,q} \in \mathcal{A}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} \mu_{u_{p,q}}) x_{s_f} - \alpha \sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} - \beta \sum_{u_{p,q} \in \mathcal{B}} \mu_{u_{p,q}} \tag{6.3c}$$

s.t.

$$\sum_{s_f \in \mathcal{F}} x_{s_f} = k \tag{6.3d}$$

$$x_{s_f} \in \{0, 1\} \tag{6.3e}$$

■

6.2.5.2 Safe Reductions

Since safe reductions do not cut off feasible solutions, whenever a selection is performed via a safe reduction, it can be regarded as a "mandatory" selection. Therefore, since the optimal solution of the problem must have such features selected, the bounds obtained using procedures that take the value of k under consideration might be improved if they force that the features that were selected via safe reductions are among the k that it chooses.

Since such features must be present in the final solution, it does not matter in what order they are selected. Therefore, the dual bounds obtained fixing any combination of these variables are valid dual bounds for the original problem.

It is easy to see that, for the procedure detailed on Section 6.1.3, the bound obtained by using all the selected features is the best. This algorithm always chooses the best case on the unrestricted scenario to guarantee that the result is an upper bound. Therefore, making any choice that differs from that, which would be very probable considering that the safe reductions accounts for all the constraints, could include features with lower degrees which, in turn, would lead to a lower upper bound.

6.2.6 Lagrangian Heuristics and Variable Fixing

Many times, dual information can be useful to assist primal heuristics, often leading to solutions of very good quality. Pessoa et al. (2010) proposed a GRASP heuristic guided by the information gathered by a Lagrangian Relaxation for the (weighted) Minimum Cardinality Set Multi Cover problem.

Consider the primal problem formulated on Figure 3.5 and its Lagrangian relaxation depicted on Figure 6.1. After solving the Lagrangian (sub)problem to optimality, the obtained solution may not be feasible for the primal. One first (naive) approach would be to simply use any greedy heuristic to solve the remaining (sub)problem and remove redundant features. However, the results produced by this approach are of very poor quality even for the weighted case (Caserta, 2007).

Next, observe that, if $\overline{z}$ is an incumbent (primal) solution, any better solution must satisfy $\sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} + \min\{\sum_{s_f \in \mathcal{F}} (1 - \sum_{u_{p,q} \in \mathcal{A}: \exists e = (s_f, u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) x_{s_f}\} \leq \overline{z}$. Let N_0 be the set of

features that have a negative reduced cost $r_{s_f} = (c_{s_f} - \sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f,u_{p,q}) \in \mathcal{E}} \lambda_{u_{p,q}}) < 0 \forall s_f \in \mathcal{F}$ (considering the optimal Lagrangian Multipliers) and N_1 be the set of features that have a positive reduced cost $r_{s_f} > 0 \forall s_f \in \mathcal{F}$. If $s_k \in N_1$ and $\sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} + \sum_{s_f \in N_0} r_{s_f} + r_{s_k} \geq \overline{z}$, then $x_{s_k} = 0$ in any better feasible solution. Similarly, if $s_k \in N_0$ and $\sum_{u_{p,q} \in \mathcal{A}} \lambda_{u_{p,q}} + \sum_{s_f \in N_0 \ s_k} r_{s_f} \geq \overline{z}$, then $x_{s_k} = 1$ in any better feasible solution (Wolsey, 1998).

Finally, the Lagrangian Multipliers could be used to guide the searches, restricting and/or sorting the candidate lists and resolving ties.

6.3 Other Alternatives

One last attempt to obtain dual bounds would be to consider the dual problems directly. Beasley (1990a) used this idea proposing a dual ascent procedure to find a (dual) feasible solution for the dual of the Linear Programming relaxation of the Set Cover Problem to initialize the Lagrangian Multipliers of the Lagrangian Dual problem solved. The discrete version of the dual can be used instead. Since, in this case, the integrality constraints would not be relaxed, the obtained bounds could be better.

Consider the problem pair depicted on Figure 3.5 and Figure 6.5, and the feasible solutions (not necessarily optimal) $\overline{z}$ and $\underline{w}$, respectively. Consider also the optimal solutions of their Linear Programming Relaxations z^{LP} and w^{LP}, respectively.

By definition, $\overline{z} \geq z^{LP}$ and $\underline{w} \leq w^{LP}$. From the principle of duality, $w^{LP} = z^{LP}$. Therefore, $\underline{w} \leq \overline{z}$.

Similarly, feasible solutions of the problems formulated in Figure 6.6 and Figure 6.7 provide dual bounds for Figure 3.6 and Figure 3.7, respectively.

Tackling these models directly, however, proves to be a whole new challenge, as the problems are as difficult as the original ones. Moreover, strong duality not always can be proven and, therefore, the optimal objective function values may still not match. This means that, even though a tighter gap may be possible to find, it probably would not allow to prove optimality and it may still be significantly large.

Figure 6.4: Mathematical formulation of the dual problem of the Max α (α, β)-k-Feature Set Problem.

$$\min \sum_{s_f \in \mathcal{F}} v_{s_f} \tag{6.4a}$$

$$\sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} + v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.4b}$$

$$- \sum_{u_{p,q} \in \mathcal{A}} w_{u_{p,q}} \geq 1 \tag{6.4c}$$

$$w_{u_{p,q}} \leq 0 \qquad \forall u_{p,q} \in \mathcal{A} \tag{6.4d}$$

$$v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.4e}$$

$$w_{u_{p,q}}, v_{s_f} \in \mathbb{Z} \tag{6.4f}$$

■

Figure 6.5: Mathematical formulation of the dual problem of the Min k (α, β)-k-Feature Set Problem.

$$\max \alpha \sum_{u_{p,q} \in \mathcal{A}} w_{u_{p,q}} - \sum_{s_f \in \mathcal{F}} v_{s_f} \tag{6.5a}$$

$$\sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} - v_{s_f} \leq 1 \qquad \forall s_f \in \mathcal{F} \tag{6.5b}$$

$$w_{u_{p,q}} \geq 0 \qquad \forall u_{p,q} \in \mathcal{A} \tag{6.5c}$$

$$v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.5d}$$

$$w_{u_{p,q}}, v_{s_f} \in \mathbb{Z} \tag{6.5e}$$

■

Figure 6.6: Mathematical formulation of the dual problem of the Max β (α, β)-k-Feature Set Problem.

$$\min \alpha \sum_{u_{p,q} \in \mathcal{A}} w_{u_{p,q}} + ku + \sum_{s_f \in \mathcal{F}} v_{s_f} \tag{6.6a}$$

$$\sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} + u + v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.6b}$$

$$- \sum_{u_{p,q} \in \mathcal{B}} w_{u_{p,q}} \geq 0 \tag{6.6c}$$

$$w_{u_{p,q}} \leq 0 \qquad \forall u_{p,q} \in \mathcal{U} \tag{6.6d}$$

$$v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.6e}$$

$$uunrestricted \tag{6.6f}$$

$$w_{u_{p,q}}, v_{s_f}, u \in \mathbb{Z} \tag{6.6g}$$

■

Figure 6.7: Mathematical formulation of the dual problem of the Max cover (α, β)-k-Feature Set Problem.

$$\min \alpha \sum_{u_{p,q} \in \mathcal{A}} w_{u_{p,q}} + \beta \sum_{u_{p,q} \in \mathcal{B}} w_{u_{p,q}} + ku + \sum_{s_f \in \mathcal{F}} v_{s_f} \tag{6.7a}$$

$$\sum_{u_{p,q} \in \mathcal{A}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} + \sum_{u_{p,q} \in \mathcal{B}: \exists e=(s_f, u_{p,q}) \in \mathcal{E}} w_{u_{p,q}} + u + v_{s_f} \geq deg(s_f) \qquad \forall s_f \in \mathcal{F} \tag{6.7b}$$

$$w_{u_{p,q}} \leq 0 \qquad \forall u_{p,q} \in \mathcal{U} \tag{6.7c}$$

$$v_{s_f} \geq 0 \qquad \forall s_f \in \mathcal{F} \tag{6.7d}$$

$$uunrestricted \tag{6.7e}$$

$$w_{u_{p,q}}, v_{s_f}, u \in \mathbb{Z} \tag{6.7f}$$

■

6.4 Conclusions

The previous chapter detailed five different heuristic methods to deal with (α, β)-k-Feature Set Approach. Even though they have lower computational requirements, they did not provide a quality estimate to the proposed solutions. Therefore, in this chapter, the second research question of this thesis is addressed: "Is it possible to quickly estimate how good these solutions would be?".

To do so, a more practical reasoning is used: it could not always be worth spending a lot of time obtaining strong bounds, if the improvement is not of practical significance. In particular, the expected objective function values for each stage of the (α, β)-k-Feature Set Approach are significantly different, ranging from only a few features (in case of the maximisation of β, for example) to large integer values (in the case of the maximisation of the total covering, for example). This is specially important considering that the values obtained in one stage define the next problem to be solved in the next, and it is not always possible to guarantee optimal solutions for every stage, which makes dual bounds more like quality estimates rather than optimality guarantees, cases in which speed would often be more important than the best quality.

The methods to obtain dual bounds proposed in this chapter are based in worst case scenario observations and Lagrangian Relaxation. The former are very efficient but are expected to provide weaker bounds, while the latter can be more computationally demanding but provide better estimates. Also, both methods share the advantages of not depending on external optimisation packages, and of being independent of the method used to obtain primal solutions, which is quite interesting from the economic and practical points of view.

Chapter 7

Computational Results

In Chapter 5 and Chapter 6 new methods were proposed to obtain feasible (primal) solutions for instances of the (α, β)-k-Feature Set approach, to address the scalability problems observed and reported in Section 4.6 and Section 4.7.

In this chapter, the proposed methods are tested and compared to existing (exact) method. Section 7.1 details the experimental design used to test the primal methods while Section 7.2 do the same for the dual methods.

7.1 Primal Heuristics

Three experiments were performed: one to determine which greedy algorithm works best with the proposed testbed, one to test the improvement provided by the proposed local search procedures on the obtained solutions and one to test the overall performance of each proposed (combined) heuristic.

The last experiment, however, deals with methods that include mechanisms to optimise also β and the total covering. Therefore, information about all objective functions is provided and the complete testbed proposed on Chapter 4 is used.

All algorithm runs were limited to one hour. In the case of the exact approach, CPLEX version 12.2 was allowed one hour to solve each of the three models under consideration.

Throughout this section, the proposed algorithms are referred as detailed on Table 7.1.

Throughout the experiments, the algorithms are combined and referred as detailed on Table 7.2.

Regarding memory usage, for the instance size under consideration, $|\mathcal{F}| = 2000, |\mathcal{A}| = |\mathcal{B}| = 20000$, the proposed methods only used about 200MB, including data structures for eight processors, as opposed to the few GBs required by CPLEX after many minutes of computation on the harder instances, when the enumeration tree grew bigger. Given the obvious disparity, memory usage tables are omitted for simplicity.

All experiments were conducted on an Intel Xeon with eight 2.5GHz processors and 16GB of RAM. All methods, including the exact approach, were multithreaded and used all eight processors with exactly eight concurrent threads.

7.1.1 Greedy Algorithms

Table 7.3 depicts the optimality gaps obtained for the minimisation of the number of selected features for each class of instances under consideration. On average, GR3 outperforms the others for most of the instances. However, observing also the associated standard deviation, it is unclear which greedy algorithm is best, specially considering that GR3 is often outperformed by the others for the instances where the exact approach performed worst. It is also interesting to notice that such simple and efficient procedures are already able to outperform the exact approaches for some of the most difficult cases.

7.1.2 Local Searches

In this experiment, only results of redundancy checks ($S_{1,0}$) are shown because all other bigger neighbourhoods defined were impractical for the instance size under consideration. They increased the running times of the methods from the order of a fraction of seconds to the order of several seconds or minutes without significant improvement. Such a behaviour would make any algorithm that rely on them impractical for larger instances.

Table 7.4 shows the optimality gaps obtained by running a first-improvement local search based on redundancy checks on the solutions obtained by each of greedy algorithms under consideration. Similarly, Table 7.5 show the same information for a best-improvement local

Table 7.1: Summary of Algorithms. Throughout this section the proposed algorithms are referred as described in this table. The first group are the greedy algorithms under consideration. The Second, refers to the same algorithms from the previous group, followed by a local search procedure. The third group refers to randomization methods under consideration. The fourth group simply refers to the basic VNS framework under consideration. The last group refers to the combination of a greedy algorithm, a local search and randomization methods into a metaheuristic framework.

Legend	**Algorithm**	**Description**
GR1	Algorithm 10	Standard constructive algorithm
GR2	Algorithm 11	Standard constructive algorithm with focus on the most infeasible part of the solution
GR3	Algorithm 12	Modified GR2, to account for safe reductions: tries to select features that have to be in the solution (Algorithm 2) before using GR2's greedy criterion.
GR4	–	Runs all the available greedy heuristics and chooses the best obtained solution of all
LS1	Algorithm 13 on $S_{1,0}$	First-improvement redundancy check local search ($S_{1,0}$ neighbourhood: attempt to deselect one feature without selecting another)
LS2	Algorithm 14 on $S_{1,0}$	Best-improvement redundancy check local search ($S_{1,0}$ neighbourhood: attempt to deselect one feature without selecting another)
RM1	Algorithm 16	Selects a number of redundant features chosen at random
RM2	Algorithm 15	Deselects a number of features chosen at random
RM3	–	Choose an available greedy algorithm at random to be used
RM4	Algorithm 17	Randomized First-improvement redundancy check local search ($S_{1,0}$ neighbourhood: attempt to deselect one feature without selecting another)
VNS	Algorithm 19	Modified VNS algorithm with a varying shake size
TSRM1	Algorithm 22	Selects a number of features chosen at random, marking them tabu
TSRM2	Algorithm 21	Deselects a number of features chosen at random, marking them tabu
TSRM3	–	Same as RM3, but considering the tabu lists in the same fashion as TSRM1 and TSRM2.
TSGR1	Algorithm 23	Same as GR1, but excluding features marked tabu from the list of candidates for selection
TSGR2	Algorithm 24	Same as GR2, but excluding features marked tabu from the list of candidates for selection
TSGR3	Algorithm 25	Same as GR3, but excluding features marked tabu from the list of candidates for selection
VNSTS	Algorithm 20	Modified VNS algorithm with a varying shake size and tabu list maintenance

Table 7.2: Combined Algorithms. Throughout this section the proposed algorithms are combined as described in this table. The first column identifies the algorithm, while the five next columns specifies the algorithms that were combined, as identified on Table 7.1. Namely, the second column identify whether the proposed modified VNS scheme considers Tabu Search or not. The third column identifies the algorithm used to generate the initial solution. The fourth identifies the randomization method used. The fifth identifies the constructive greedy heuristic used to restore feasibility after the randomization process, if applicable. Finally, the last column identifies the local search procedure used.

Legend	Restart	Initial	Shake	Restore Feasibility	Local Search
MH1	VNS	GR4	RM1	–	RM4
MH2	VNS	GR4	RM2	RM3	RM4
MH3	VNS	GR4	RM2	GR3	RM4
MH4	VNSTS	GR4	TSRM	TSRM3	RM4
MH5	VNSTS	GR4	TSRM	TSGR3	RM4

search based also on redundancy checks.

Inspecting Table 7.4 it is possible to see a similar behaviour as that observed on the results with the greedy heuristics: running redundancy checks on solutions obtained with GR3 tended to lead to better solutions on average, but with a higher variation, making it unclear which is the best algorithm.

Comparing the results on Table 7.4 and Table 7.5, however, it is possible to see that the values are very similar, making first improvement local searches more interesting due to the fact that they analyse less possible movements and are, therefore, faster.

7.1.3 Meta-Heuristics

Since the algorithms considered in this section have a random component, each instance was solved ten times with different seeds, and the average value was considered for five different designs, detailed on Table 7.2.

The initial shake size (SZ_0 in VNS and VNSTS) was always set to 0.5% of the selected features what was increased by 0.1% at each failed iteration (SZ_i, in the same algorithms). If the size of the shake was smaller than one, it was set to one. That would help to keep the algorithm at the minimum change for a longer period of time, favouring a more intense search on the neighbourhood closer to the best known solution.

Whenever applicable, every time a feature is deselected at random, or selected to complete

Table 7.3: Greedy Heuristic Comparison. Each row represents the results obtained with each proposed greedy heuristic for the minimisation of the number of selected features, in ascending order of optimality gap obtained with the exact approach. The first column identifies the instance as described on Table 4.2. The second shows the gaps, calculated with Equation 4.5, obtained using the exact approach. The last three the gaps obtained using each of the greedy procedures proposed. The best gap amongst the three proposed heuristics is marked in boldface. The values in italics mark the cases where the proposed algorithms obtained a better gap than the exact method.

Inst	CPLEX	GR1	GR2	GR3
1	0.00%	8.70%	10.23%	**3.78%**
26	0.00%	8.53%	10.27%	**5.33%**
51	0.00%	12.63%	14.18%	**6.29%**
76	0.00%	12.84%	15.14%	**7.56%**
101	0.00%	15.33%	18.17%	**4.80%**
41	3.23%	9.97%	8.07%	**7.98%**
46	3.88%	8.68%	**7.35%**	7.89%
16	3.92%	9.76%	8.29%	**7.89%**
21	3.97%	8.80%	**7.38%**	7.69%
66	4.34%	10.39%	8.35%	**7.93%**
61	5.82%	14.36%	11.78%	**11.42%**
71	5.93%	11.66%	**9.15%**	**9.15%**
36	6.33%	13.90%	12.35%	**11.55%**
11	7.96%	15.86%	**14.18%**	14.44%
96	9.05%	13.80%	12.36%	**11.96%**
116	9.84%	18.32%	**15.31%**	15.52%
86	13.61%	19.30%	18.49%	**17.74%**
81	17.00%	**21.79%**	23.23%	22.21%
121	17.67%	*16.46%*	***15.00%***	*15.35%*
31	21.69%	**26.13%**	26.63%	26.30%
6	22.66%	**24.92%**	27.22%	25.82%
111	27.13%	29.25%	**29.13%**	**29.13%**
106	27.85%	31.54%	32.49%	**31.42%**
56	43.87%	***39.38%***	*42.99%*	*43.17%*
91	52.98%	*17.44%*	***15.20%***	***15.20%***
Average		16.79%	16.52%	**14.70%**
Standard Deviation		8.05%	9.05%	9.88%

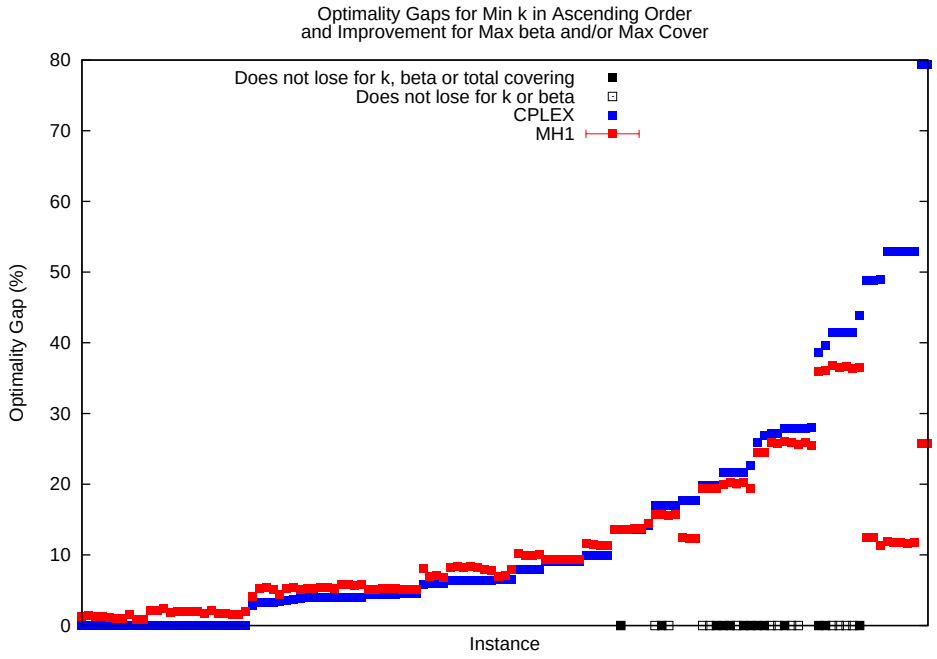
Table 7.4: Local Search Impact Comparison. Each row represents the results obtained after running first-improvement redundancy checks (LS1 on $S_{1,0}$) on the solutions obtained with each of the proposed greedy heuristics for the minimisation of the number of selected features. The first column identifies the instance as described on Table 4.2. The second shows the gaps, calculated with Equation 4.5, obtained using the exact approach. The last three the gaps obtained after running a first improvement local search based on redundancy checks on the solutions obtained each of the greedy procedures proposed. The best gap amongst the three proposed heuristics is marked in boldface. The values in italics mark the cases where the proposed algorithms obtained a better gap in average than the exact method.

Inst	**CPLEX**	**GR1+LS1**	**GR2+LS1**	**GR3+LS1**
1	0.00%	5.31%	3.95%	**3.25%**
26	0.00%	4.70%	5.18%	**3.73%**
51	0.00%	7.04%	6.59%	**4.91%**
76	0.00%	7.84%	9.48%	**6.00%**
101	0.00%	7.97%	8.37%	**4.22%**
41	3.23%	6.58%	**6.48%**	6.67%
46	3.88%	6.34%	**6.06%**	6.53%
16	3.92%	7.79%	7.19%	**6.47%**
21	3.97%	7.17%	**6.86%**	6.97%
66	4.34%	6.93%	6.57%	**6.20%**
61	5.82%	11.24%	10.06%	**9.78%**
71	5.93%	8.46%	8.15%	**8.08%**
36	6.33%	10.63%	**10.27%**	**10.27%**
11	7.96%	12.57%	12.29%	**12.02%**
96	9.05%	11.22%	10.53%	**10.32%**
116	9.84%	14.40%	**13.39%**	**13.39%**
86	13.61%	16.31%	**15.33%**	16.22%
81	17.00%	**18.39%**	19.52%	18.96%
121	17.67%	*14.14%*	***13.56%***	*13.71%*
31	21.69%	**23.53%**	24.24%	24.24%
6	22.66%	***22.27%***	23.43%	23.62%
111	27.13%	28.02%	28.64%	**27.89%**
106	27.85%	30.06%	30.19%	**29.04%**
56	43.87%	***38.77%***	*42.08%*	*42.27%*
91	52.98%	*14.20%*	***13.25%***	*13.65%*
Average		13.68%	13.67%	**13.14%**
Standard Deviation		8.76%	9.41%	9.68%

Table 7.5: Local Search Impact Comparison. Each row represents the results obtained after running best-improvement redundancy checks (LS2 on $S_{1,0}$) on the solutions obtained with each of the proposed greedy heuristics for the minimisation of the number of selected features. The first column identifies the instance as described on Table 4.2. The second shows the gaps, calculated with Equation 4.5, obtained using the exact approach. The last three the gaps obtained after running a best improvement local search based on redundancy checks on the solutions obtained each of the greedy procedures proposed. The best gap amongst the three proposed heuristics is marked in boldface. The values in italics mark the cases where the proposed algorithms obtained a better gap in average than the exact method.

Inst	**CPLEX**	**GR1+LS2**	**GR2+LS2**	**GR3+LS2**
1	0.00%	5.31%	3.95%	**3.25%**
26	0.00%	4.70%	5.18%	**3.73%**
51	0.00%	7.04%	6.89%	**4.91%**
76	0.00%	7.84%	9.48%	**6.00%**
101	0.00%	7.97%	8.37%	**4.22%**
41	3.23%	6.58%	**6.39%**	6.58%
46	3.88%	6.16%	**6.06%**	6.53%
16	3.92%	7.39%	7.19%	**6.47%**
21	3.97%	7.17%	**6.86%**	6.97%
66	4.34%	6.79%	6.79%	**6.13%**
61	5.82%	11.06%	10.15%	**9.69%**
71	5.93%	8.23%	**8.00%**	8.15%
36	6.33%	10.98%	**10.27%**	10.39%
11	7.96%	12.84%	12.29%	**12.16%**
96	9.05%	11.08%	**10.53%**	10.60%
116	9.84%	14.04%	**13.39%**	13.75%
86	13.61%	16.41%	**15.53%**	16.31%
81	17.00%	**18.39%**	19.41%	19.52%
121	17.67%	*14.00%*	***13.64%***	*13.93%*
31	21.69%	**23.53%**	24.24%	24.24%
6	22.66%	***22.27%***	23.81%	23.62%
111	27.13%	28.15%	28.64%	**27.89%**
106	27.85%	30.06%	30.31%	**28.91%**
56	43.87%	***38.77%***	*42.08%*	*42.08%*
91	52.98%	*13.96%*	***13.17%***	*13.57%*
Average		13.63%	13.70%	**13.18%**
Standard Deviation		8.80%	9.43%	9.67%

Figure 7.1: Optimality gaps, calculated with Equation 4.5, obtained with MH1, for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps obtained using the exact approach, before the timeout of one hour. The proposed approach results were at least as good as those of the exact approach considering the minimisation of the number of selected features and maximisation of β for the instances marked with a white square on the horizontal axis, and at least as good as the results considering all objective functions for the instances marked with a black square. In Appendix D.2, Table D.2 detail the results obtained with exact approach, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree. Table D.5 details the results obtained with the proposed heuristic and includes information about running times.



an infeasible solution (except for the initial solution), it is marked tabu, with a tabu tenure value in $[0, p)$ (where p equals 10% of the maximum number of unsuccessful iterations) chosen at random.

The algorithms stops after one hour of computation time or 100 unsuccessful iterations. This value was also arbitrarily set after preliminary testing with different values.

Figure 7.1, Figure 7.2, Figure 7.3, Figure 7.4 and Figure 7.5, depict the obtained average optimality gaps, and their respective standard deviation, for each proposed metaheuristic design. The instances, displayed on the horizontal axis are ordered in increasing order of the optimality gap obtained with the exact approach, and are marked with a white square on the

Figure 7.2: Optimality gaps, calculated with Equation 4.5, obtained with MH2, for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps obtained using the exact approach, before the timeout of one hour. The proposed approach results were at least as good as those of the exact approach considering the minimisation of the number of selected features and maximisation of β for the instances marked with a white square on the horizontal axis, and at least as good as the results considering all objective functions for the instances marked with a black square. In Appendix D.2, Table D.2 detail the results obtained with exact approach, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree. Table D.5 details the results obtained with the proposed heuristic and includes information about running times.

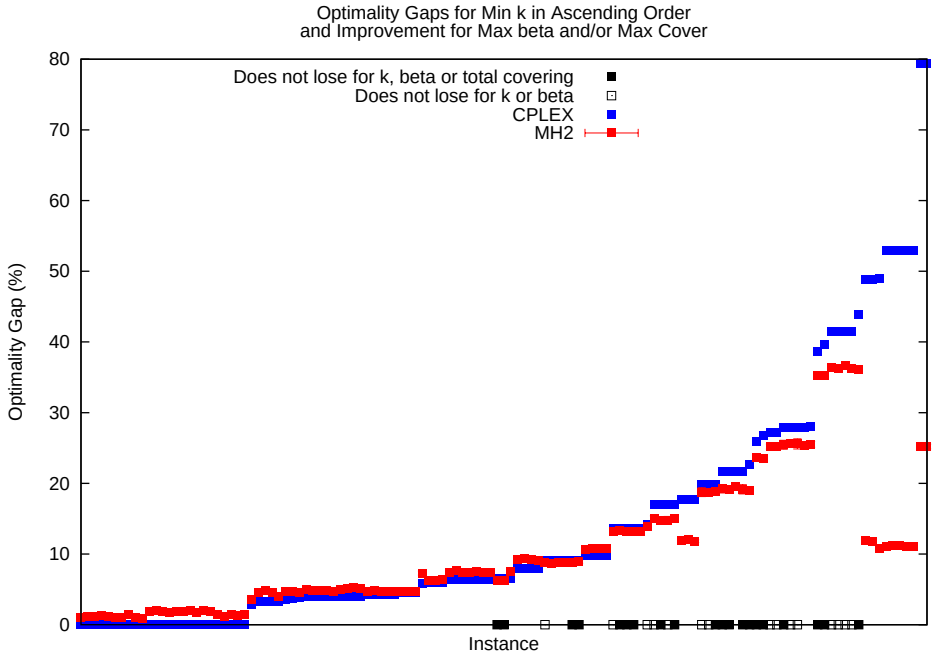


Figure 7.3: Optimality gaps, calculated with Equation 4.5, obtained with MH3, for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps obtained using the exact approach, before the timeout of one hour. The proposed approach results were at least as good as those of the exact approach considering the minimisation of the number of selected features and maximisation of β for the instances marked with a white square on the horizontal axis, and at least as good as the results considering all objective functions for the instances marked with a black square. In Appendix D.2, Table D.2 detail the results obtained with exact approach, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree. Table D.5 details the results obtained with the proposed heuristic and includes information about running times.

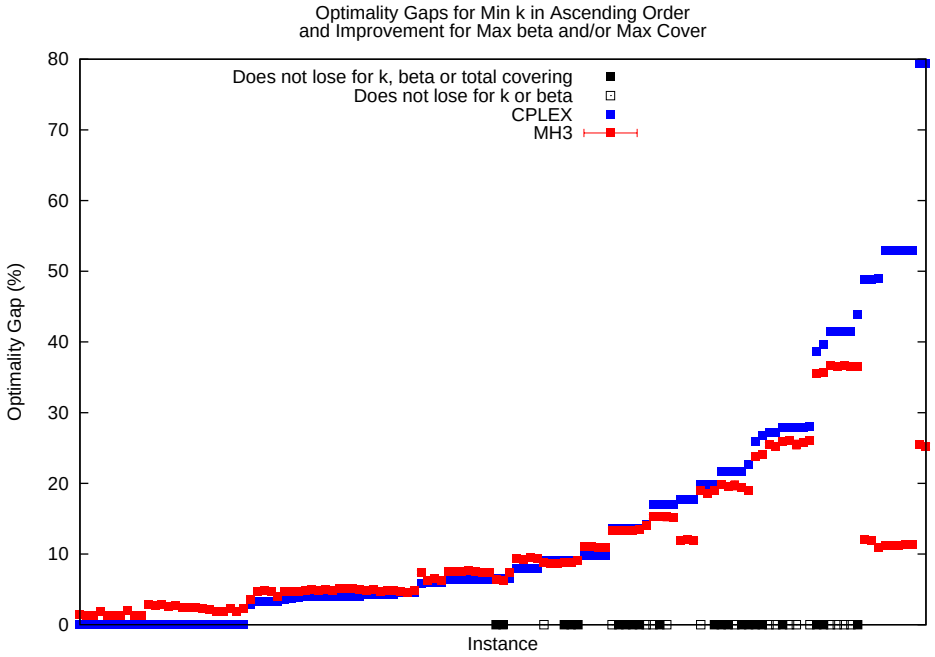


Figure 7.4: Optimality gaps, calculated with Equation 4.5, obtained with MH4, for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps obtained using the exact approach, before the timeout of one hour. The proposed approach results were at least as good as those of the exact approach considering the minimisation of the number of selected features and maximisation of β for the instances marked with a white square on the horizontal axis, and at least as good as the results considering all objective functions for the instances marked with a black square. In Appendix D.2, Table D.2 detail the results obtained with exact approach, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree. Table D.5 details the results obtained with the proposed heuristic and includes information about running times.

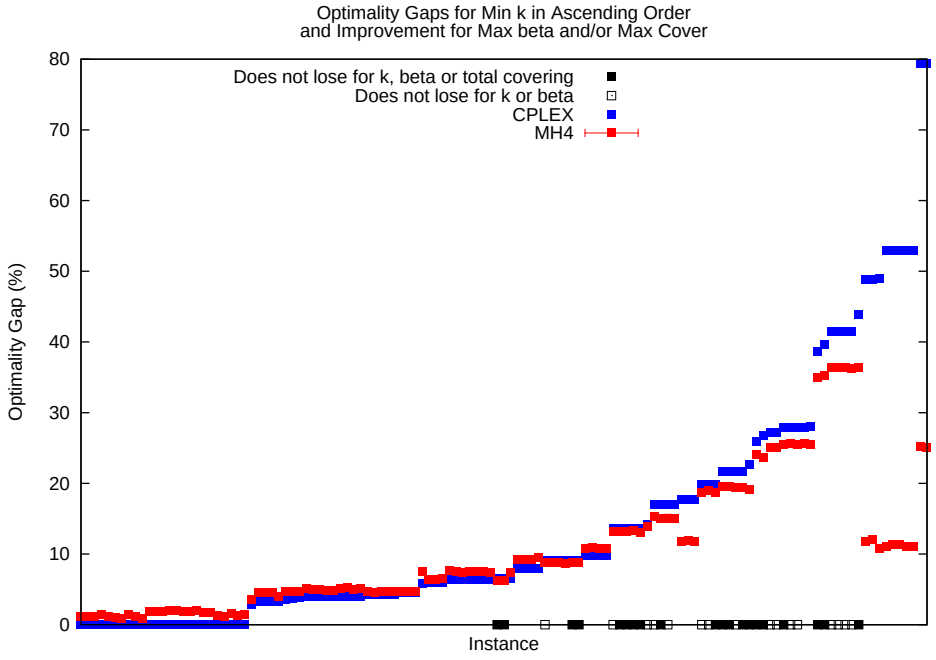
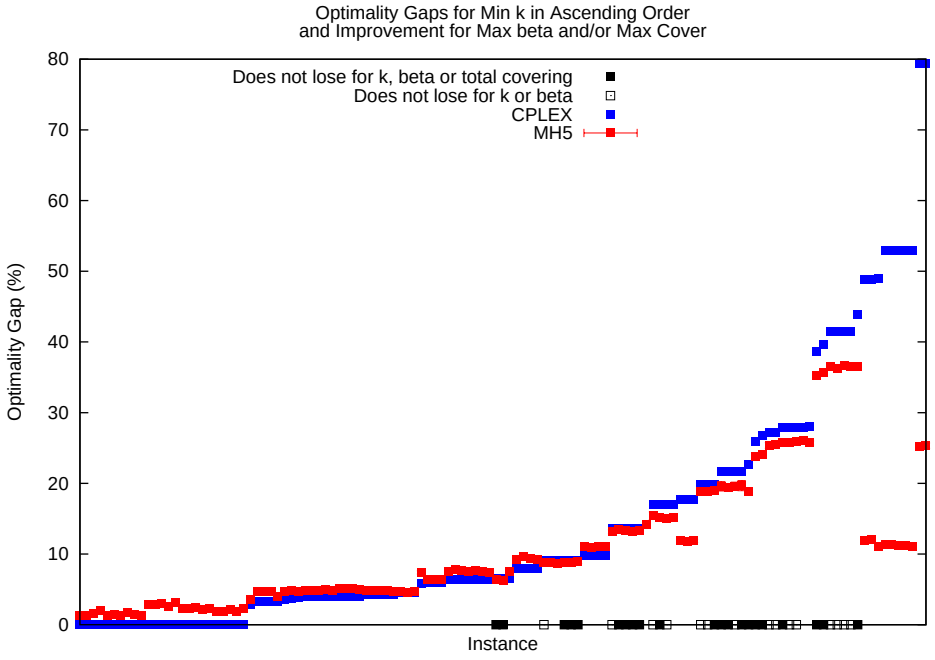


Figure 7.5: Optimality gaps, calculated with Equation 4.5, obtained with MH5, for the minimisation of the number of selected features. The instances are ordered in ascending order of optimality gaps obtained using the exact approach, before the timeout of one hour. The proposed approach results were at least as good as those of the exact approach considering the minimisation of the number of selected features and maximisation of β for the instances marked with a white square on the horizontal axis, and at least as good as the results considering all objective functions for the instances marked with a black square. In Appendix D.2, Table D.2 detail the results obtained with exact approach, identify the instances in the order they are shown and include also information about the performance of safe reductions, running time, and size of the solution enumeration tree. Table D.5 details the results obtained with the proposed heuristic and includes information about running times.



horizontal axis when the proposed approach could improve not only the number of selected features but also the value of β, with respect to the results obtained by exact approach. Similarly, a black square on the horizontal axis marks the instances where all objective functions under consideration could be improved. Since instances are not numbered in these plots to ease their readability, please refer Appendix D.2 (Table D.2-Table D.4 and Table D.5) for information on individual instances, as indexed on Table 4.2.

Analysing these plots and tables Table D.2-Table D.4 and Table D.5, it is possible to see that all algorithms had a very similar performance and they all had insignificant standard deviations. It can also be seen that the proposed algorithms could improve the objective functions rather frequently for the instances where the exact approach obtained a poorer gap. For the instances where the exact approach obtained lower gaps, or the optimal solution, the obtained solutions were also competitive, showing also small gaps, but not improving the best known results.

Observing Table 7.6, however, it is possible to notice that the proposed heuristic that chooses the constructive method at random performed slightly better in average than others for most of the instances.

Moreover, looking at Table 7.7, it is easy to see that not only the proposed heuristics can find high quality solutions, but they can also improve the state of art regarding the structural quality of the solutions. It was remarkable how they could match or obtain higher values of β and total covering even when matching or lowering the number of selected features without compromising the solution process.

Table 7.8 compares the results of all proposed heuristics and the exact method for the three last stages of the (α, β)-k-Feature Set approach for the real world instances. Table 7.9 details the respective running times.

Analysing these tables it is possible to see that the proposed results are very competitive, reaching a difference of less than 5% from the optimal solution, using only a fraction of the exact approach's time.

Table 7.6: Average primal Gaps and Running times, and their respective standard deviations, for the exact approach and each of the proposed algorithms. Excluding the exact approach, which is deterministic, these averages consider ten runs with different seeds for each instance. In the case of the exact approach, the reported value is the average of the stacked values shown in Figure 4.3. Table D.5 details all the obtained gaps for each instance individually. Table D.6 details all the running times of each proposed heuristic for each instance individually.

Optimality Gap						
	CPLEX	±	**MH1**	±	**MH2**	±
Average	14.30%	–	11.15%	0.28%	10.66%	0.29%
Standard Deviation	16.97%	–	9.34%	0.11%	9.28%	0.12%
	MH3	±	**MH4**	±	**MH5**	±
Average	10.87%	0.29%	10.67%	0.28%	10.86%	0.29%
Standard Deviation	9.27%	0.11%	9.29%	0.10%	9.25%	0.11%
Running Time (s)						
	CPLEX	±	**MH1**	±	**MH2**	±
Average	10421.75	–	6.92	1.51	11.25	2.85
Standard Deviation	8022.87	–	3.64	1.31	3.20	1.48
	MH3	±	**MH4**	±	**MH5**	±
Average	11.73	2.70	11.41	2.83	11.85	2.75
Standard Deviation	2.91	1.26	3.24	1.40	2.76	1.14

Table 7.7: Individual Stage Comparison: number of instances, out of 125, for which the proposed heuristics at least matches the exact approach's average results considering up to one hour of computation for each model.

Objective	**MH1**	**MH2**	**MH3**	**MH4**	**MH5**
min k	42	55	55	55	54
min k and max β	26	35	37	36	36
all the stages	13	20	21	20	21

Table 7.8: Results of the heuristics for real world instances. On the following table, each of the lines is an instance detailed on Section 4.1 and each column the average objective function value of ten runs with different seeds of an algorithm and, the respective standard deviation. The first column is the optimal solution obtained with the exact method, except for the maximum β and total covering of the SM instance, as the exact method could not finish in reasonable time. In this case, the best solution found after one hour of computation time for each of the stages is used. The best known value of β is also used as input for the Max Cover stage.

Instance	**CPLEX Optimum**	**Objective Function Value** MH1	MH2	MH3	MH4	MH5
			Min k			
DS	36	36	36	36	36	36
ADMF	292	301.1+-1.9	295.2+-0.4	295.1+-0.6	295.1+-0.3	294.8+-0.6
PD1	8675	8862.10+-41.71	8716.20+-3.65	8711.60+-3.27	8716.10+-3.51	8710.30+-2.91
PD2	289	290.70+-0.67	289	289.10+-0.32	289	289
PC	867	890.10+-6.14	874.90+-1.29	874.50+-0.97	874.70+-0.95	874.90+-1.10
SM	128	130.90+-0.74	129.30+-0.48	129.70+-0.48	129.50+-0.53	129.50+-0.71
			Max β			
DS	10	10	10	10	10	10
ADMF	118	119.6+-2.6	121.5+-1	119.9+-2	121.7+-1.3	120.2+-1.6
PD1	3677	3741.50+-41.14	3686.70+-9.72	3682.30+-6.77	3689.30+-11.19	3677.80+-9.08
PD2	76	76.70+-0.82	74.80+-0.42	74.30+-0.48	74.80+-0.63	74.30+-0.48
PC	231	243.20+-4.80	239.80+-2.35	239.90+-2.60	239.30+-3.71	239.80+-2.20
SM	*34*	41.20+-1.69	39.70+-1.34	39.90+-0.88	39.80+-1.14	39.70+-1.16
			Max Cover			
DS	2016	2016	2016	2016	2016	2016
ADMF	581608	599322.3+-3505.9	587433.8+-908.4	587716.7+-1365.59	587334.90+-750.71	586890.20+-1294.54
PD1	24935478	25488617.20+-143021.69	25097835.20+-14386.85	25072108.60+-13272.63	25091660.80+-11042.40	25066935.60+-10851.66
PD2	53196	53562.00+-139.15	53251.20+-61.18	53244.00+-76.58	53235.20+-69.35	53229.00+-33.10
PC	7540529	7788214.90+-71081.45	7632191.90+-22189.82	7624780.30+-9450.75	7629536.10+-16132.33	7630646.30+-14157.34
SM	*48560464*	51560465.90+-336654.36	50833151.90+-226512.42	51004935.10+-219774.00	50897664.90+-246125.44	50883364.50+-302354.75

Table 7.9: Running times of the heuristics for real world instances. On the following table, each of the lines is an instance detailed on Section 4.1 and each column the average running time of ten runs with different seeds of an algorithm, and the respective standard deviation. The first column is the total running time CPLEX used to solve all stages of the exact approach. In the case of SM, CPLEX could not finish any of the last two stages within reasonable time.

Instance	**CPLEX**	**MH1**	**MH2**	**MH3**	**MH4**	**MH5**
DS	0.02	0.03	0.020	0.020	0.02	0.02
ADMF	13.4	0.56+-0.06	0.86+-0.15	0.81+-0.14	0.9+-0.2	0.93+-0.24
PD2	0.9	0.54+-0.05	0.30+-0.03	0.27+-0.02	0.31+-0.03	0.28+-0.04
PD1	1178.8	15.63+-0.40	26.41+-4.77	22.23+-1.69	26.07+-3.19	23.18+-1.63
PC	3091.8	3.84+-0.25	4.16+-0.73	3.93+-0.72	4.48+-0.48	3.72+-0.95
SM	*timeout*	71.80+-13.35	100.34+-31.68	84.76+-15.58	81.07+-11.95	97.20+-27.64

7.2 Dual Bounds

In this section, the quality of the bounds obtained using the methods proposed in this chapter is evaluated. It is important to note at this point that the advantage pursued with these experiments is not on bound quality, but a better computation time vs quality tradeoff in practice.

While the dual gaps reported from the exact method consider all cuts and safe reductions it could find before either proving optimality or timeout, the proposed methods only account for the safe reductions done using Algorithm 2 on the original problem, fixing the features it selects. Therefore, the results obtained with exact method are expected to be at least as good as the ones obtained by the proposed methods, and are reported simply to serve as a basis of comparison for the expected bound quality, providing a better idea of how much room there would be for improvement and if it would actually be worth the effort in practice.

The algorithms were allowed up to one hour to obtain bounds for all instances proposed on Chapter 4. All gaps were calculated also using Equation 4.5 and Equation 4.6 (in the case of the maximisation of β), considering the best primal solutions obtained after one hour of computation time solving the respective model with the exact approach and the dual bounds discussed in this chapter. These primal bounds are also used as input parameters for the Lagrangian Relaxations and w_{LD} values for the Subgradient method.

7.2.1 Lower bounds of Min k

Table 7.10 shows the optimality gaps obtained after up to one hour of computation time of the exact method, the obvious lower bound on k described on Section 6.1.1, the associated standard deviations, the lower bound obtained by solving the respective Lagrangian Dual Problem described on Section 6.2.2 and their respective running times.

In this table it is easy to see that, as expected, the Lagrangian bounds were significantly better than the obvious bound, even though it took also significantly more computational time. The extra computational effort required, however, was still relatively low and the algorithms were able to converge within reasonable time. The exact method, however, outperformed all methods, but took up to 1000 times longer, reaching the time limit, even if compared to the

Table 7.10: Dual Bounds for the minimisation of the number of selected features. Each row represents the results obtained with each proposed method to obtain lower bounds for the minimisation of the number of selected features, in ascending order of optimality gap obtained with the exact approach with a timeout set to 15, 30 and 60 seconds. The first column identifies the instance as described on Table 4.2. The second shows the gaps obtained using the exact approach. The next two pairs show the optimality gaps calculated with Equation 4.5, using the value of α and Lagrangian Dual respectively, considering the best primal bound obtained with the exact method before one hour of computation time; and their computational time required to obtain these bounds.

	CPLEX 15min		*CPLEX 30min*		*CPLEX 60min*		*Greedy*		*Lagrangian*	
Instance	gap	time (s)	gap	time (s)	gap	time (s)	gap	time (s)	gap	time (s)
1	0.00%	1.45	0.00%	1.40	0.00%	1.47	99.81%	0.21	37.01%	2.72
26	0.00%	1.53	0.00%	1.41	0.00%	1.49	99.82%	0.21	39.26%	2.18
51	0.00%	2.40	0.00%	2.23	0.00%	2.32	99.83%	0.21	29.43%	3.90
76	0.00%	3.90	0.00%	4.12	0.00%	4.19	99.84%	0.60	27.00%	3.99
101	0.00%	2.01	0.00%	1.89	0.00%	1.96	99.84%	0.21	28.50%	4.92
41	3.65%	978.52	3.65%	2077.71	3.23%	3600.77	98.42%	0.45	12.64%	3.38
46	3.97%	940.06	3.97%	1839.29	3.88%	3602.80	98.28%	0.28	10.11%	3.43
16	5.44%	1688.22	4.26%	1956.22	3.92%	3600.88	98.30%	0.44	13.28%	3.54
21	4.43%	921.28	4.43%	1973.78	3.97%	3600.51	98.16%	0.27	11.74%	3.61
66	4.51%	1053.53	4.51%	1900.00	4.34%	3600.75	98.65%	0.90	10.81%	4.17
61	9.11%	1021.66	7.59%	1983.89	5.82%	3608.69	99.04%	0.54	19.72%	3.61
71	6.66%	954.77	5.93%	2021.40	5.93%	3601.15	98.53%	0.27	11.41%	4.14
36	6.33%	965.22	6.33%	1887.45	6.33%	3601.86	98.90%	0.53	24.18%	3.11
11	8.72%	981.24	8.72%	1985.35	7.96%	3601.81	98.85%	0.54	28.34%	3.12
96	42.30%	1117.65	9.05%	2557.28	9.05%	3767.36	98.66%	1.11	12.92%	4.42
116	48.34%	1025.95	10.06%	2264.67	9.84%	3603.45	98.78%	0.85	13.53%	1.54
86	13.61%	1135.91	13.61%	2216.59	13.61%	3637.28	99.16%	1.35	25.60%	2.68
81	25.88%	1340.15	26.30%	2005.14	17.00%	3603.38	99.57%	1.00	52.14%	2.60
121	48.88%	1587.99	18.27%	2080.09	17.67%	3601.56	98.79%	0.27	20.29%	3.25
31	40.84%	1084.49	40.84%	2451.36	21.69%	3601.69	99.52%	0.35	67.55%	2.48
6	43.72%	916.97	39.92%	1947.00	22.66%	3600.62	99.50%	0.35	67.67%	2.42
111	27.30%	1012.54	27.30%	2044.57	27.13%	3645.86	99.29%	0.52	38.69%	2.74
106	28.91%	1320.36	28.91%	1868.55	27.85%	3633.61	99.63%	0.83	54.83%	2.49
56	56.05%	1219.71	56.05%	1892.94	43.87%	3609.75	99.69%	0.74	73.46%	2.32
91	52.98%	1128.96	52.98%	2301.58	52.98%	3837.41	99.35%	1.21	55.55%	2.86
Average	19.27%	896.26	14.91%	1650.63	12.35%	2902.90	99.13%	0.57	31.43%	3.18
Standard Deviation	19.94%	494.78	16.99%	859.38	13.89%	1481.25	0.56%	0.34	19.89%	0.80

Table 7.11: Average Dual Gaps and Running times, and their respective standard deviations, for the maximisation of β, obtained before one hour of computation with the exact method, and using the greedy and Lagrangian procedures depicted on Section 6.1.2 and Section 6.2.3, respectively. In Appendix D.3, Table D.7 details these values for each of the proposed instances individually. Note that the gaps shown on this table are not on the percent form, but calculated using Equation 4.6.

	CPLEX		*Greedy*		*Lagrangian*	
	Gap	Time (s)	Gap	Time (s)	Gap	Time (s)
Average	2.91	4939.38	5.85	0.49	4.14	594.45
Standard Deviation	3.10	6308.06	5.15	0.18	2.80	522.91

Lagrangian Dual.

7.2.2 Upper bounds of Max β

Table 7.11 presents optimality gaps calculated with Equation 4.6 with the values obtained after up to one hour of computation time of the exact method, the obvious upper bound on β described on Section 6.1.2 and the lower bound obtained by solving the respective Lagrangian Dual Problem described on Section 6.2.3, respectively, along with their respective running times. Note that, the timeout limit of one hour simply stops the algorithms from evaluating any more solutions, and may exceed 3600 seconds due to clean up procedures and work in progress.

Interestingly, the obvious bounds were not very far from all others in average, and only used a derisive amount of computational time compared to the other methods. The Lagrangian method was about ten times faster than the exact method in average, to obtain a dual bound that differs by only two coverings.

7.2.3 Upper bounds of the Max Total Covering

Table 7.12 shows the average gaps obtained after up to one hour of computation time of the exact method, the obvious upper bound on the total covering described on Section 6.1.3, the lower bound obtained by solving the respective Lagrangian Dual Problem described on Section 6.2.4 and their respective running times.

Table 7.12: Average Dual Gaps (in percent form, calculated with Equation 4.5) and Running times, and their respective standard deviations, for the maximisation of the Total Covering, obtained before one hour of computation with the exact method, and using the greedy and Lagrangian procedures depicted on Section 6.1.3 and Section 6.2.4, respectively. In Appendix D.3, Table D.8 details these values for each of the proposed instances individually.

	CPLEX		*Greedy*		*Lagrangian*	
	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
Average	3.00	2580.31	7.82	1.15	4.96	2997.86
Standard Deviation	2.99	1621.13	5.76	0.75	3.19	1148.44

In this table it is possible to see that, even though the bounds obtained by the exact method were always the best, and the Lagrangian method outperformed the obvious, the latter require a derisive amount of time when compared to the first. It is also interesting to notice that the difference from one method to another does not exceed 5%.

7.2.4 Variable Fixing

Using the Lagrangian multipliers to fix variables one would be able to reduce the size of the problem under consideration, however, preliminary testing showed that these conditions are very rarely met, and very few variables can be fixed using this procedure. This idea was the key to the success of every heuristic evolved from Beasley (1990a)'s, and the fact the it becomes ineffective for the unicost case was already reported by Caserta (2007).

That, however, introduced an overhead to the feature comparisons that proved to be too high regardless of the comparison criterion used, even if caching values. Even though this approach led to solutions of good quality, often exploring less "intermediate" solutions, exploring the neighbourhood of more randomized initial solutions with a faster algorithm seemed to be a faster and more effective alternative. Also, exploring more "intermediate" solutions has the obvious advantage of contributing to short term memory more often, which is exploited by the Tabu Search structure.

The results of these attempts are omitted from this text because they were always significantly worse than the ones produced by the primal heuristics alone, both in terms of quality and computation time.

Chapter 8

Conclusions

In the previous chapter, all the proposed methods were tested and compared to the existing method. This chapter aims to answer the research questions presented on Section 1.4 using the information learnt with the experiments performed on **??**.

In particular, Section 8.1 discusses and answers the question *"Is it possible to quickly select features on highly dimensional datasets, without compromising the robustness of the solution?"* affirmatively.

Section 8.2, in turn, discusses and answers *"Is it possible to quickly estimate how good these solutions would be?"* also affirmatively.

8.1 Primal Heuristics

In Chapter 5, different algorithms to address the (α, β)-k-Feature Set approach were proposed and compared to the existing exact approach. The proposed greedy algorithms and local searches acted as the centrepieces of different metaheuristics that optimised all three objective functions under consideration at the same time: the minimisation of the number of selected features, and the maximisation of β and total covering.

Regarding the proposed greedy heuristics, GR3 almost always outperformed the others. However, the fact the other algorithms still performed better for a few instances suggests that considering all three approaches could also be advantageous, not only for introducing

solution diversity, but also for covering more different scenarios, where different algorithms could perform better.

The analysis of the results obtained by running a local search based on redundancy checks on the solutions obtained with each of the greedy heuristics under consideration followed the same pattern observed on the greedy results alone. That, together with the fact that the local search procedure did not improve the obtained solutions significantly, but only by less than 2%, suggests that the initial solutions seemed to be the key to find better solutions. In particular, refining solutions obtained with GR3 was most successful, but the fact that the other algorithms could still outperform it for a few instances suggest that the using all of them could still be advantageous.

The choice of the order that the movements are performed within the local search also did not have a big overall impact. Therefore, the first-improvement would be preferred, for being faster. However, since the results with first and best improvement were different, randomizing the first-improvement approach is also an interesting idea, as it can be done without significantly compromising the algorithm's performance, and would contribute to the overall solution diversity of the proposed methods.

The obtained overall results showed that all the approaches performed quite similarly, obtained competitive results for instances where the exact approach performs well, and improving the state-of-art for instances where the exact approach performs poorly. In particular, the designs that explored the infeasible region of the solution space near known good solutions were slightly more effective, specially when using all the proposed greedy heuristics, chosen at random. That supports the theory that solution diversity was the key to the success of the proposed algorithms.

As the number of selected features is reduced, the maximal values of β and total covering can be drastically limited. The bigger the number of selected features, the bigger the maximal total covering is expected to be. Similarly, the bigger the maximal total covering, the bigger β can be as well. Therefore, once the number of selected features is reduced, matching or increasing the values of β and/or total covering gets more difficult because of the lack of viable options. Also because of the hierarchical form of the objective function, gaps for the

maximum β and total covering, would not be comparable between solutions that do not share the exact same objective function values for all previous stages of the approach, as the bounds would be relative to different parameters, and thus, different problems.

Indeed, it is often the case for the most difficult instances that the exact approach can find the optimal solution for the problem, or find a high quality one, but fails to find any other feasible solution at all with a non-zero value of β. Since in this work the main target are solutions with this specific structure, finding such solutions quickly is one of its main contributions over traditional Set (Multi) Cover approaches.

Nevertheless, the proposed algorithms could also improve the best known solutions for all objective functions under consideration. Once more, the approaches that explored the infeasible solution space outperformed the one that only works on the feasible solution space. That could be due to the fact that they rely on frequent runs of randomized constructive greedy algorithms, which supports the theory that the simultaneous optimization component also plays an important role to the success of the proposed algorithms.

Finally, not only the quality of the solutions obtained by proposed algorithms were competitive with the existing (exact) approaches, but they outperformed the state-of-art algorithms in computational time and memory usage, lowering significantly the resource requirements of the approach. That would be particularly interesting not only to deal with bigger datasets but also to design more complex approaches. In particular, for the real world instances, the proposed methods could reach the optimal solution for the smaller and easier problems, and very competitive results for bigger and harder ones using only a fraction of the computational effort required by the exact approach. The main advantage of the proposed methods, however, is the ability to obtain good quality solutions for very difficult instances, that can not be effectively solved by the exact approach within reasonable time. A good example of that could be observed for the dataset proposed by Charlesworth et al. (2010), for which the exact method can not reach the optimal solution within reasonable time, and the proposed methods could quickly obtain solutions of very competitive quality to the best known.

8.2 Dual Bounds

From the experiments detailed in Section 7.2 it became clear that, in practice, not always the exact approach is the best option to obtain bounds for the problem under consideration. The quality of the obtained bounds is not always appreciable, but improving it often requires a substantial investment in computational time.

Regarding the minimisation of k, an analysis of the tradeoff between computational effort and bound quality suggests that, if the problem can not be solved using the exact method, the Lagrangian method is often the best option in practice, as it seems to converge quickly enough for most of the cases, while the exact method reaches the timeout too often. It is interesting to notice that this problem takes the value of α as a parameter, which is an obvious lower bound for the problem. When the distribution of edges is more uneven for the nodes in set $\mathcal{A}$, the instance tends to have nodes with a very small degree. That immediately leads also to small values of alpha, and poorer associated bounds, that could be improved by the exact method with the introduction of cuts and safe reductions that could be quite expensive to find and apply. As the distribution of edges connected to the nodes in set $\mathcal{A}$ gets more uniformly distributed, the instances tend to yield higher values of α and, therefore, better bounds that can be attainable quickly using simpler procedures such as the Lagrangian method.

In the case of the maximisation of β, it seemed to depend even more on the instance characteristics. A similar behaviour was expected regarding the edge density of the edges connected to the nodes on set $\mathcal{B}$, as the same reasoning can be applied. Even though the phenomena was indeed observed, the fact that the problem also takes the number of selected features as a parameter must also be taken into consideration and could be rather restrictive. When the distribution of edges connected to the nodes in set $\mathcal{B}$ is very uneven, the value of the obvious upper bound tends to be very low, and therefore good. In these cases it should be preferred, given its immediate time requirement advantage. As the distribution of edges connected to the nodes in set $\mathcal{B}$ gets more uniform, this bound tends to get poorer and the imposed number of selected features tends to be more restrictive. In this case, procedures that take such a parameter under consideration, such as the Lagrangian method, should be preferred in case the exact method can not be applied.

Finally, considering the maximisation of the Total Covering, it was easy to see that, as expected, good estimates could already be obtained using only the obvious bound obtained with the method proposed on Section 6.1.3. Considering that it is expected that only a few features are to be selected because of the second stage of the (α, β)-k-Feature Set Approach, and that for the target instances have $m \ggg n$, it is reasonable to state that the imposed number of selected features is the most restrictive constraint in this problem. Indeed, any more selected features would imply a potentially significant increase in the objective function. Fortunately, that is also the easiest constraint to satisfy and reflects positively on all the obtained bounds. When the edge distribution of the nodes in $\mathcal{U}$ is very uneven, a great disparity between node degrees is expected, and therefore, also a big range of objective functions values is attainable. Since the obvious, greedy, lower bound considers the worst possible case, these bounds were expected to be poorer. In this case, relatively more elaborate methods such as the Lagrangian method, that consider the other parameters as well, could be preferred. However, in practice the greedy bounds would be enough to satisfy the applications' needs without significantly compromising computational effort. As the edge distribution gets more uniform, the node degrees are not expected to be very different, and therefore the worst case is also not expected to be far from the best. In this case, the greedy method should be preferred, given its immediate speed advantage.

Since the proposed Lagrangian methods provided good quality bounds, but required a lot of computational effort, further research on this topic would include finding ways to speed up these processes, which were out of the scope of this work. Boyd et al. (2003) surveys a few methods to do so. Relax-and-cut schemes, as presented by Guignard (1998); Lucena (2005), often work well for these purpose. Finding ways to initialise the hotstart method with a reasonably good dual feasible solution, as done by Beasley (1987) for the Set Cover Problem, could also be an interesting option. The subgradient method is also very prone to parameter tuning, which can be further explored.

Further research on heuristic methods to solve the dual problems introduced on Section 6.3 would also be very interesting, considering the potential gain observed when less characteristics of the problems involved were relaxed.

8.3 Concluding Remarks and Future Research

In this thesis, the Feature Selection problem was revisited with focus on large biological datasets by analysing the (α, β)-k-Feature Set approach, and proposing new approaches to obtain solutions and estimate their quality quickly.

The main motivation for this project was the occurrence of instances that the current state of art exact approach could not solve, even on fairly small cases. Generally speaking, the problems were two-fold: over-abundance of solutions, and how to determine how to find the best ones.

To the best of our knowledge at the time of this research, even though different works used different methodologies to address different problems, there was only one main framework proposed to solve the problem published on the literature. It consisted of a four stage approach based on the solution of four combinatorial problems.

The first stage of the approach, which determines the maximum number of features that explain the differences between the samples, can be solved in polynomial time and, therefore, did not pose any challenge to the whole method.

The second stage of the approach, the well known unicost Set Multi Cover Problem, can many times be solved in practice. However, it was possible to generate a pool of instances which were difficult for the Integer Programming exact approach found in the literature. Unsurprisingly, these instances were also problematic for the last two stages of the approach. The third stage, which dealt with the matter of maximising the number of features that explain the similarities between samples that are in the same class, given a fixed number of selected features and number of features that are required to explain the differences between samples in different classes; was particularly demanding, as even finishing one of its subproblems often required more than 50% of the time limit available.

The case of instances with low edge density was particularly complicated for the third stage because it led to problems with a very limited objective function value range. That leads to a large set of solutions with the same objective function values, which can be a problem in case the dual bounds, used to prune families of solutions in the exact method's implicit solution enumeration tree. When the dual bounds are not good enough, too many solutions can not be

pruned and have to be analysed. This phenomenon was particularly important for the third stage of the approach, as it dealt with a rather limited objective function value range, which was often null.

Following Caserta's advice on the target type of instances, several primal intensive heuristics were devised for the problem. The key aspects that granted their success were a restricted range randomization, to overcome the difficulty introduced by large search neighbourhoods and the simultaneous optimization of all objective functions, to exploit the problem's symmetry and avoid unnecessary recalculation and the overhead introduced by the multi-stage approach.

Using this design it was possible to improve all the objective functions in many cases of the testbed, mostly amongst the most difficult - and therefore more interesting - instances, using only a small fraction of the computation time allowed. This is a rather great achievement, specially considering that the objective functions are correlated and updated hierarchically. That is specially because, once a higher priority objective function is improved, the next ones tend to obtain lower values in general, therefore making the search for better secondary objective values more difficult, if not non-existent.

Finally, the matter of obtaining dual bounds to estimate the quality of the obtained solutions was addressed using various techniques. It was possible to show that not always a great computational effort is worthwhile to obtain sufficiently good bounds in practice. In particular, efficient simple bounds proved to be quite effective to provide quality estimates for specific classes of instances. For the instances where they do not provide good estimates, the characteristics that prevent that could be isolated and more elaborated methods were proposed to deal with the problem as alternatives to the exact method. Even though the quality of the best known dual bounds was not improved, the computational effort vs estimate quality tradeoff could be significantly improved. That allows for bigger and harder problems to be solved in practice as most of the times it is not strictly required that the solutions are provably optimal, but only good enough. Also, the fact that the instances can be analysed to choose the best method to deal with them is also of great practical and scientific interest, as this information can also be used to develop new, more efficient, methods to address each case.

Therefore, since the proposed methods complements the applicability of the existing, by using only a fraction of the computational effort required whilst providing competitive solutions, it was possible to answer affirmatively both the research questions proposed: *"Is it possible to quickly select features on highly dimensional datasets, without compromising the robustness of the solution?"* and *"Is it possible to quickly estimate how good these solutions would be?"*.

Further work on the primal algorithms would include experimenting with heuristics that implement cost-effective use of dual information. On the dual algorithms, it would also be interesting to study effective ways to use primal information and other techniques to speed up the most complex dual algorithms; and to design new methods to solve the dual formulation of the models under consideration.

Appendix A

Summary of Notation

m: Number of Samples.

n: Number of Features.

M: Value matrix of a case/control dataset.

$m_{p,f}$: Value of sample p for feature f.

C: Class array of a case/control dataset.

c_p: Class of sample p.

$\mathcal{F}$: Set of available features in the $(\alpha, \beta) - k$ Feature Set of Set (Multi)Cover instance.

f_i: The subset associated to feature i.

$\mathcal{F}\prime$: Set of selected features in $\mathcal{F}$.

$\mathcal{F}^1$: Set of selected features in $\mathcal{F}$, candidates for removal.

$\mathcal{F}^2$: Set of selected features in $\mathcal{F} \backslash \mathcal{F}\prime$, candidates for selection.

$\mathcal{U}$: Universe of sample pairs in the $(\alpha, \beta) - k$ Feature Set of Set (Multi)Cover instance.

$u_{p,q}$: The pair of samples associated to the pair of samples p and q.

$\mathcal{A}$: Elements of the universe that model a pair of samples that belong to different classes.

$\mathcal{B}$: Elements of the universe that model a pair of samples that belong to the same class.

$\mathcal{E}$: Set of edges in the $(\alpha, \beta) - k$ Feature Set of Set (Multi)Cover instance.

$e = (s_f, u_{p,q})$: An edge $e \in \mathcal{E}$ that connects subset $s_f \in \mathcal{F}$ to sample pair $u_{p,q} \in \mathcal{U}$.

$N(v)$: The neighbours of vertex v.

$|S|$: The cardinality of set S.

γ^*: The optimal value of object (e.g.:variable, solution or set) γ.

d_u: number of coverings needed to meet the covering requirement of pair of samples $u \in \mathcal{U}$.

$\mathcal{T}$: Tabu tenures. Each of these values are associated with exactly one feature $f_i \in \mathcal{F}$. t_i: The tabu tenure of feature $f_i \in \mathcal{F}$

Appendix B

Discretization and Pre-Selection of Features

In order to discretize the data, and pre-select features, Fayyad and Irani's algorithm is used. Since the discretization topic is beyond the scope of this text, only an overall description of the algorithm is shown, as presented in Berretta et al. (2008). For a detailed description of the algorithm, a thorough analysis and proposed optimizations, please refer to the original paper (Fayyad and Irani, 1993).

The algorithm, applied to every feature, starts by sorting the set of samples S in non decreasing order of values. Next it identifies and calculates all possible threshold values that distinguish between two existing classes C_1 and C_2. Those are the arithmetic mean of the values of two adjacent samples with different classes in the sorted list created on the previous step.

Next, it encodes the sample values based on each threshold found, 0 for samples with values smaller than it and 1 otherwise; and separates them into two subsets S_1, for samples encoded as 0 and S_2 for samples encoded as 1. Let $P(C_i, S_j)$ be the proportion of samples in S_j that are labelled as C_i. The class information entropy values are calculated using Equation B.1

and Equation B.2. The threshold that yields minimum class entropy is chosen.

$$Ent(S_j) = -\sum_{i=1}^{2} P(C_i, S_j) \log_2[P(C_i, S_j)] \tag{B.1}$$

Then, the Minimum Description Length (MDL) Principle is applied to determine if the obtained discretization is acceptable, which happens if and only if Equation B.5, calculated by Equation B.2, Equation B.3 and Equation B.4; yields true. So, if any feature fails Equation B.5 it is excluded from further consideration, or in other words, does not pass Fayyad and Irani's entropy filter, and thus is not included in the discretized dataset. In Equation B.5, let m be the total number of samples. In Equation B.4, define as c the number of different classes present in the feature, c_1 the number of different classes present in S_1, and c_2 the number of different classes present in S_2.

$$E(S) = \frac{|S_1|}{|S|} Ent(S_1) + \frac{|S_2|}{|S|} Ent(S_2) \tag{B.2}$$

$$Gain(S) = Ent(S) - E(S) \tag{B.3}$$

$$\Delta(S) = \log_2(3^c - 2) - [cEnt(S) - c_1 Ent(S_1) - c_2 Ent(S_2)] \tag{B.4}$$

$$Gain(S) > \frac{\log_2(m-1)}{m} + \frac{\Delta(S)}{m} \tag{B.5}$$

Appendix C

Case-Control Datasets: Valid and Invalid Graph Extracts

This appendix complements Section 4.2 by illustrating how $(\alpha, \beta) - k$ Feature Set Problem instances can be mapped into Set Multi-Cover instances.

Assuming that a partitioning of the nodes in $\mathcal{A}$ and $\mathcal{B}$ is provided, and that the cardinalities of sets $\mathcal{A}$ and $\mathcal{B}$ are consistent with (C.1)-(C.2), the graph has to be translatable to a case-control table. That is, there must exist a valid labelling of the samples such that all associated subgraphs with three nodes are valid in the sense that they can be expressed in case-control format without ambiguities, according to Figure C.1. All such possible ambiguities are illustrated on Figure C.2.

Let C^1 and C^2 be the (disjoint) sets of samples that belong to the first and second classes, respectively.

$$\binom{|C|}{2} = |\mathcal{A}| + |\mathcal{B}| \tag{C.1}$$

$$|\mathcal{B}| = \binom{|C^1|}{2} + \binom{|C^2|}{2} \tag{C.2}$$

In these figures, every subfigure has three illustrations of the same scenario. From left to right, the first illustration highlights the class relationship between three samples a, b and c. The superscript (e.g.: a^1) denotes the sample's class, as seen on the third illustration, which is

an extract of the dataset. The operator between any two samples highlights the relationship modelled on the graph pattern depicted on the second illustration. While in Figure C.1 both the classes and relationship match, on Figure C.2 they are contradictory and depict a absurd scenarios, where the graph pattern can not be translated into values in a dataset's table.

Figure C.1: Valid (realistic) graph patterns for the $(\alpha, \beta) - k$ Feature Set problem, compared to the Set (multi)Cover problem. Every subfigure has three illustrations of the same scenario. From left to right, the first illustration highlights the class relationship between three samples o, p and q. The superscript (e.g.: o^1) denotes the sample's class, as seen on the third illustration, which is an extract of the dataset. The operator between any two samples highlights the relationship modelled on the graph pattern depicted on the second illustration. Note that, for the valid cases, both the classes and relationship match.

(a) Valid case with three equal values, all samples belonging to the same class, all class mirrors, rotations and inversion are obvious symmetries. The value inversion is also an obvious symmetry.

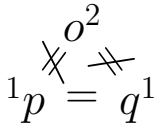
Sample	o	p	q
Value	0	0	0
Class	1	1	1

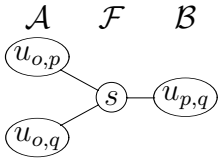
(b) Valid case with three equal values, one sample belonging to a different class. Any mirror or rotation only changes the name of the nodes, yielding the same graph topology (and hence the same value matrix), being thus a symmetry. The values and class inversion is also a symmetry.

Sample	o	p	q
Value	0	0	0
Class	2	1	1

(c) Valid case with one equal value, all samples belonging to the same class, all class mirrors, rotations and inversion are obvious symmetries. The value inversion is also an obvious symmetry.

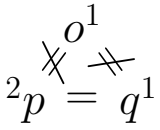
Sample	o	p	q
Value	0	1	1
Class	1	1	1

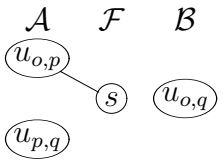




Sample	o	p	q
Value	0	1	1
Class	2	1	1

(d) Valid case with one equal value, one sample belonging to a different class. Any mirror or only changes the name of the nodes, yielding the same graph topology (and hence the same value matrix), being thus a symmetry. The values and class inversion is also a symmetry. One class rotation leads to case Figure C.1e





Sample	o	p	q
Value	0	1	1
Class	1	2	1

(e) Valid case with one equal value, one sample belonging to a different class. Any mirror or only changes the name of the nodes, yielding the same graph topology (and hence the same value matrix), being thus a symmetry. The values and class inversion is also a symmetry. One class rotation is a symmetry and two class rotations lead to case Figure C.1d

■

Figure C.2: Invalid (unrealistic) graph patterns for the $(\alpha, \beta) - k$ Feature Set problem, compared to the Set (multi)Cover problem. Every subfigure has three illustrations of the same scenario. From left to right, the first illustration highlights the class relationship between three samples o, p and q. The superscript (e.g.: o^1) denotes the sample's class, as seen on the third illustration, which is an extract of the dataset. The operator between any two samples highlights the relationship modelled on the graph pattern depicted on the second illustration. Note that, for the invalid cases, the classes and relationship are contradictory.

(a) Invalid case with two equal values, all samples belonging to the same class, all class mirrors, rotations and inversion are obvious symmetries. The value inversion is also an obvious symmetry.

Sample	o	p	q
Value	0	0	?
Class	1	1	1

(b) Invalid case with two equal values, one sample belonging to a different class.

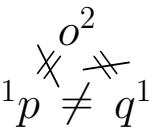
Sample	o	p	q
Value	0	0	?
Class	2	1	1

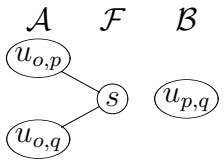
(c) Invalid case with two equal values, one sample belonging to a different class. One rotation from Figure C.2b

Sample	o	p	q
Value	0	0	?
Class	1	2	1

(d) Invalid case with no equal values, all samples belonging to the same class, all class mirrors, rotations and inversion are obvious symmetries. The value inversion is also an obvious symmetry.

Sample	o	p	q
Value	0	1	?
Class	1	1	1





Sample	o	p	q
Value	0	1	?
Class	2	1	1

(e) Invalid case with no equal values, one sample belonging to a different class.

■

Appendix D

Result Listings

This appendix contains the result listings for all data presented on the previous chapters. In particular, it includes numerical values for each instance under consideration, to complement information that was either presented graphically or summarized with average values, to ease the interpretation of the text.

D.1 Chapter 4

The following tables complement the material presented on Chapter 4

Table D.1 details the information presented graphically on Figure 4.3 and identify the instances in the order they are shown, according to the indexes presented on Table 4.2.

Table D.1: CPLEX running times, in ascending order of total running time of the approach. Each row represents one of the proposed instances. The first column identifies the instance, as described on Table 4.2. The next four columns show the running times of the exact method for each of the $(\alpha, \beta) - k$ Feature Set approach stages, in order. The last column shows the total time required by the whole approach. The exact method was allowed one hour to work for each stage. Even though the computational times are capped at 3600 seconds, this is when the exact approach is told to stop searching for solutions, and the values depicted in this graph include the time needed to finish ongoing work, which can be significant.

Inst	Max α	Min k	Max β	Max Cover	Total Time	Inst	Max α	Min k	Max β	Max Cover	Total Time
1	0.21	1.47	1.78	0.85	4.31	14	0.55	3600.69	4313.50	3600.33	11515.08
26	0.21	1.49	1.82	0.96	4.46	45	0.51	3601.10	4326.06	3607.24	11534.91
2	0.20	1.47	2.32	0.87	4.86	42	0.44	3600.50	4360.84	3600.23	11562.00
27	0.20	1.49	2.53	0.94	5.17	65	0.54	3600.12	4427.36	3600.24	11628.26
51	0.21	2.32	3.35	3.33	9.20	25	0.27	3601.43	4481.78	3600.08	11683.55
101	0.21	1.96	3.14	3.96	9.27	9	0.36	3602.04	4485.28	3602.18	11689.85
3	0.22	1.49	9.85	0.95	12.50	15	0.54	3600.43	4524.88	3600.84	11726.69
5	0.20	1.49	11.73	0.91	14.33	120	0.74	4047.72	4080.71	3607.88	11737.05
76	0.60	4.19	7.21	4.47	16.47	73	0.27	3603.15	4569.11	3600.35	11772.87
30	0.22	1.50	15.35	0.93	18.00	107	1.19	3630.12	4578.11	3600.98	11810.40
28	0.20	1.54	16.13	0.96	18.83	24	0.27	3601.47	4631.05	3600.07	11832.86
29	0.21	1.51	18.41	0.92	21.04	10	0.36	3600.51	4713.51	3600.59	11914.98
52	0.52	4.49	11.14	5.19	21.34	18	0.44	3600.86	4717.43	3600.50	11919.23
77	0.77	6.50	15.54	5.59	28.41	40	0.54	3600.20	4745.19	3600.37	11946.30
102	0.59	6.32	15.09	7.01	29.00	8	0.36	3601.71	4849.14	3600.34	12051.54
53	0.21	2.33	28.22	3.27	34.03	125	0.27	3602.17	4939.96	3600.44	12142.84
103	0.21	2.57	28.57	7.58	38.93	47	0.56	3600.43	4954.73	3600.35	12156.07
4	0.20	1.53	38.35	1.03	41.11	66	0.90	3600.75	5008.28	3600.38	12210.30
55	0.21	2.33	56.34	6.84	65.72	68	1.51	3605.45	5013.08	3605.92	12225.97
80	0.50	5.88	59.76	7.87	74.00	97	0.75	3678.08	4972.28	3601.22	12252.33
104	0.59	4.46	83.82	8.86	97.73	49	0.36	3606.40	5112.37	3602.36	12321.49
78	0.59	7.24	828.07	5.31	841.20	44	0.56	3600.93	5503.90	3600.33	12705.72
54	0.33	5.51	2015.07	18.36	2039.27	70	1.38	3600.49	5565.29	3602.75	12769.91
105	0.21	2.58	3190.20	10.61	3203.59	86	1.35	3637.28	5650.63	3600.05	12889.30
95	1.18	3602.48	1.85	1.13	3606.64	13	0.55	3601.49	5827.61	3600.08	13029.72
114	1.46	3601.21	4.46	4.26	3611.39	37	0.54	3601.05	5990.02	3600.62	13192.23
94	1.55	3604.82	3.32	2.31	3612.00	63	0.54	3600.09	6015.72	3602.58	13218.93
93	1.36	3633.40	3.51	2.36	3640.62	72	1.19	3602.08	6093.50	3600.21	13296.98
122	0.60	3667.36	3.56	2.22	3673.74	98	0.93	3666.05	6073.88	3601.85	13342.71
117	1.90	3678.63	5.43	2.36	3688.31	43	0.44	3600.47	6232.06	3600.59	13433.55
112	1.38	3765.07	5.23	2.33	3774.02	23	0.27	3600.57	6330.04	3600.63	13531.51

(*continued...*)

Inst	Max α	Min k	Max β	Max Cover	Total Time	Inst	Max α	Min k	Max β	Max Cover	Total Time
124	0.79	3775.52	2.97	2.14	3781.42	64	1.29	3605.74	6429.11	3602.39	13638.52
91	1.21	3837.41	3.39	2.66	3844.66	62	0.50	3601.78	6506.55	3603.56	13712.39
92	1.69	3844.08	2.50	2.11	3850.37	113	0.52	3646.75	6504.03	3600.44	13751.73
79	0.70	6.73	3920.06	7.57	3935.06	74	0.95	3618.17	7016.72	3600.56	14236.39
56	0.74	3609.75	43.94	1797.54	5451.96	115	0.54	3610.17	7055.09	3604.25	14270.05
22	0.28	3606.55	5.39	3600.14	7212.36	87	1.49	3634.55	7140.02	3600.38	14376.44
6	0.35	3600.62	22.78	3600.24	7223.99	119	0.73	3658.67	7115.17	3602.58	14377.15
16	0.44	3600.88	23.23	3600.05	7224.59	50	0.38	3600.64	7720.61	3605.07	14926.70
31	0.35	3601.69	22.51	3600.05	7224.59	118	1.16	3738.43	7654.72	3602.41	14996.72
21	0.27	3600.51	25.13	3600.06	7225.96	20	0.44	3600.14	7858.85	3600.41	15059.83
11	0.54	3601.81	29.01	3600.32	7231.68	19	0.44	3600.68	8807.05	3600.24	16008.41
36	0.53	3601.86	30.03	3600.07	7232.49	48	0.37	3600.65	9079.93	3600.56	16281.50
61	0.54	3608.69	27.93	3600.53	7237.68	108	1.38	3609.25	9847.98	3603.08	17061.69
111	0.52	3645.86	23.87	3600.17	7270.42	99	1.22	3602.50	10580.99	3600.13	17784.84
81	1.00	3603.38	65.47	3601.55	7271.40	32	0.37	3603.04	10710.56	3601.54	17915.51
46	0.28	3602.80	70.05	3601.47	7274.60	83	0.86	3600.44	12114.71	3600.14	19316.15
41	0.45	3600.77	106.72	3600.31	7308.25	89	1.17	3632.53	12724.39	3611.02	19969.11
106	0.83	3633.61	73.65	3601.04	7309.13	58	0.58	3623.48	12872.81	3600.34	20097.22
121	0.27	3601.56	122.89	3600.19	7324.91	75	0.27	3600.40	13277.99	3600.15	20478.81
67	1.29	3603.51	708.18	3602.48	7915.46	57	0.91	3666.87	14580.87	3600.06	21848.71
123	0.27	3632.55	867.24	3600.75	8100.81	100	0.58	3605.82	14770.70	3600.51	21977.61
82	0.89	3601.10	3616.47	3601.27	10819.73	90	1.81	3622.17	15364.48	3656.86	22645.32
38	0.54	3600.82	3698.03	3601.36	10900.75	59	0.81	3601.44	15533.25	3600.32	22735.82
71	0.27	3601.15	3741.13	3600.02	10942.57	84	1.11	3617.63	16122.19	3617.75	23358.68
12	0.28	3601.10	3805.79	3600.42	11007.58	60	0.67	3615.56	16169.38	3600.90	23386.50
7	0.36	3601.40	3814.76	3600.04	11016.56	110	1.14	3604.83	22707.50	3601.12	29914.59
17	0.44	3600.42	3921.63	3600.14	11122.63	69	1.37	3601.76	23599.10	3601.16	30803.38
34	0.36	3601.48	3928.71	3600.31	11130.86	88	1.55	3620.88	24499.53	3604.94	31726.89
35	0.36	3600.91	3989.54	3600.03	11190.83	109	0.94	3602.84	30353.52	3604.13	37561.43
39	0.54	3603.13	4040.56	3600.12	11244.35	85	0.87	3601.01	35976.86	3613.83	43192.57
96	1.11	3767.36	3883.89	3601.67	11254.02	**Avg**	0.66	2901.40	4939.38	2580.31	10421.75
116	0.85	3603.45	4239.25	3603.45	11447.00	**StdDev**	0.42	1456.03	6308.06	1621.13	8022.87
33	0.36	3601.26	4273.29	3600.26	11475.17						

Table D.2, Table D.3 and Table D.4 detail the information presented graphically on Figure 4.4, Figure 4.5 and Figure 4.6, respectively, and also indentify the instances in the order they are show, according to the indexes presented on Table 4.2.

Table D.3: CPLEX results the maximisation of β, in ascending order of optimality gaps. Each row represents one of the proposed instances. The first column identifies the instance, as described on Table 4.2. The second and third columns show the sizes of sets $\mathcal{F}$ and $\mathcal{U}$, respectively, after performing safe reductions (CPLEX's presolve). The next three columns show the optimality gap, calculated using Equation 4.6, the running time to solve this stage and the number of nodes of the solution enumeration tree that CPLEX had to expand, respectively.

Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes	Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes
1	63.25%	2.49%	0.00	1.78	0	7	99.95%	99.79%	2.00	3814.76	2
2	75.05%	3.15%	0.00	2.32	0	32	99.95%	99.87%	2.00	10710.56	3
3	82.90%	17.46%	0.00	9.85	0	37	99.65%	97.63%	2.00	5990.02	6
4	41.80%	3.07%	0.00	38.35	13	62	99.65%	98.36%	2.00	6506.55	2
5	83.15%	51.14%	0.00	11.73	0	82	98.70%	92.04%	2.00	3616.47	3
6	99.85%	52.18%	0.00	22.78	0	87	99.70%	98.97%	2.00	7140.02	2
11	99.30%	53.16%	0.00	29.01	0	107	99.85%	99.60%	2.00	4578.11	0
16	97.75%	52.07%	0.00	23.23	0	8	100.00%	99.09%	3.00	4849.14	2
21	99.15%	52.23%	0.00	25.13	0	58	100.00%	99.42%	3.20	12872.81	0
22	99.10%	73.07%	0.00	5.39	0	33	100.00%	98.48%	3.24	4273.29	0
26	71.65%	2.84%	0.00	1.82	0	98	98.40%	100.00%	4.00	6073.88	3
27	81.80%	3.65%	0.00	2.53	0	23	99.25%	100.00%	4.00	6330.04	4
28	84.80%	29.60%	0.00	16.13	0	59	100.00%	99.42%	4.66	15533.25	0
29	82.90%	46.82%	0.00	18.41	0	83	99.95%	99.97%	4.95	12114.71	0
30	82.90%	49.18%	0.00	15.35	0	108	99.85%	99.82%	4.96	9847.98	0
31	99.95%	52.56%	0.00	22.51	0	60	100.00%	99.42%	5.00	16169.38	3
36	99.65%	55.17%	0.00	30.03	0	68	99.20%	100.00%	5.00	5013.08	2
41	97.15%	53.05%	0.00	106.72	0	73	98.35%	100.00%	5.00	4569.11	3
46	99.20%	54.49%	0.00	70.05	0	38	99.65%	100.00%	5.00	3698.03	3
51	81.40%	3.60%	0.00	3.35	0	13	99.40%	100.00%	5.00	5827.61	3
52	83.45%	5.38%	0.00	11.14	0	18	97.85%	99.99%	5.00	4717.43	4
53	83.45%	32.80%	0.00	28.22	0	43	97.15%	99.92%	5.00	6232.06	4
54	83.45%	51.87%	0.00	2015.07	20	118	99.35%	100.00%	5.00	7654.72	3
55	48.45%	4.00%	0.00	56.34	111	49	99.20%	100.00%	5.00	5112.37	2
56	100.00%	52.48%	0.00	43.94	0	113	99.85%	100.00%	5.37	6504.03	0
61	97.80%	53.39%	0.00	27.93	0	9	99.95%	99.89%	5.81	4485.28	0
67	99.20%	87.90%	0.00	708.18	0	63	97.80%	99.99%	5.87	6015.72	0
76	82.10%	3.83%	0.00	7.21	0	34	99.95%	99.94%	5.90	3928.71	0
77	82.75%	5.64%	0.00	15.54	0	14	99.40%	100.00%	5.99	4313.50	0
78	82.75%	34.26%	0.00	828.07	26	19	97.85%	100.00%	6.00	8807.05	4
80	82.75%	52.21%	0.00	59.76	0	99	98.40%	100.00%	6.00	10580.99	3
81	98.70%	51.35%	0.00	65.47	0	15	99.40%	100.00%	6.00	4524.88	2
91	0.00%	0.00%	0.00	3.39	0	48	99.20%	100.00%	6.00	9079.93	3
92	0.00%	0.00%	0.00	2.50	0	20	97.85%	100.00%	6.00	7858.85	3
93	0.00%	0.00%	0.00	3.51	0	25	99.25%	100.00%	6.00	4481.78	2
94	0.00%	0.00%	0.00	3.32	0	109	99.85%	99.82%	6.11	30353.52	0
95	0.00%	0.00%	0.00	1.85	0	64	97.80%	99.99%	6.14	6429.11	2
101	79.60%	3.59%	0.00	3.14	0	10	99.95%	99.89%	6.32	4713.51	0
102	80.75%	5.12%	0.00	15.09	0	44	97.15%	99.99%	6.42	5503.90	0
103	53.15%	2.64%	0.00	28.57	7	35	99.95%	99.94%	6.43	3989.54	0
104	34.65%	2.06%	0.00	83.82	5	88	99.70%	100.00%	6.50	24499.53	0
105	81.45%	52.05%	0.00	3190.20	612	110	99.85%	99.82%	6.77	22707.50	0

Table D.2: CPLEX results the minimisation of the number of features, in ascending order of optimality gaps. Each row represents one of the proposed instances. The first column identifies the instance, as described on Table 4.2. The second and third columns show the sizes of sets $\mathcal{F}$ and $\mathcal{A}$, respectively, after performing safe reductions (CPLEX's presolve). The next three columns show the optimality gap, calculated using Equation 4.5, the running time to solve this stage and the number of nodes of the solution enumeration tree that CPLEX had to expand, respectively.

Instance	Red. $\mathcal{F}$	Red $\mathcal{A}$	Gap	Time	Nodes
1	37.15%	1.84%	0.00%	1.47	0
26	16.95%	0.88%	0.00%	1.49	0
51	71.40%	3.65%	0.00%	2.32	0
76	76.60%	4.13%	0.00%	4.19	0
101	68.05%	3.50%	0.00%	1.96	0
6	99.75%	99.78%	25.00%	3600.62	908
31	99.90%	99.88%	25.00%	3601.69	682
56	99.95%	98.84%	25.00%	3609.75	92
81	98.65%	96.86%	25.00%	3603.38	208
106	99.80%	99.63%	25.00%	3633.61	86
11	99.20%	99.99%	50.00%	3601.81	1500
36	99.60%	100.00%	50.00%	3601.86	1405
61	97.75%	99.98%	50.00%	3608.69	1204
86	99.65%	100.00%	50.00%	3637.28	79
111	99.80%	100.00%	50.00%	3645.86	151
16	97.65%	99.99%	75.00%	3600.88	2311
41	97.10%	99.98%	75.00%	3600.77	3181
66	99.15%	100.00%	75.00%	3600.75	482
91	99.35%	100.00%	75.00%	3837.41	21
116	99.30%	100.00%	75.00%	3603.45	35
21	99.05%	100.00%	100.00%	3600.51	2137
46	99.15%	100.00%	100.00%	3602.80	1072
71	98.30%	99.99%	100.00%	3601.15	741
96	98.35%	99.99%	100.00%	3767.36	34
121	99.25%	100.00%	100.00%	3601.56	56
Average	90.03%	80.35%	12.35%	2902.90	655.40
Standard Deviation	21.20%	39.59%	13.89%	1481.25	872.63

(*continued...*)

Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes	Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes
106	99.85%	53.19%	0.00	73.65	0	85	98.70%	98.43%	6.79	35976.86	0
111	99.85%	53.37%	0.00	23.87	0	39	99.65%	100.00%	7.00	4040.56	0
112	0.00%	0.00%	0.00	5.23	0	45	97.15%	99.99%	7.00	4326.06	2
114	0.00%	0.00%	0.00	4.46	0	50	99.20%	100.00%	7.00	7720.61	2
117	0.00%	0.00%	0.00	5.43	0	40	99.65%	100.00%	7.00	4745.19	3
121	99.30%	53.25%	0.00	122.89	0	84	98.70%	98.43%	7.00	16122.19	3
122	0.00%	0.00%	0.00	3.56	0	125	99.30%	100.00%	7.00	4939.96	3
123	99.30%	100.00%	0.00	867.24	0	24	99.25%	100.00%	7.00	4631.05	3
124	0.00%	0.00%	0.00	2.97	0	74	98.35%	100.00%	7.33	7016.72	0
66	99.20%	59.97%	1.00	5008.28	4	65	97.80%	99.99%	7.46	4427.36	0
71	98.35%	53.36%	1.00	3741.13	6	90	99.70%	100.00%	7.86	15364.48	0
86	99.70%	54.61%	1.00	5650.63	4	69	99.20%	100.00%	7.89	23599.10	0
96	98.40%	52.83%	1.00	3883.89	8	115	99.70%	100.00%	7.89	7055.09	0
116	99.35%	54.07%	1.00	4239.25	3	89	99.70%	100.00%	7.98	12724.39	0
79	82.75%	50.97%	1.00	3920.06	176	119	99.35%	100.00%	8.54	7115.17	0
17	97.85%	82.29%	1.00	3921.63	7	75	98.35%	100.00%	8.83	13277.99	0
42	97.15%	76.73%	1.00	4360.84	5	70	99.20%	100.00%	9.00	5565.29	2
47	99.15%	93.29%	1.00	4954.73	2	100	98.40%	100.00%	9.77	14770.70	0
72	98.35%	89.38%	1.00	6093.50	2	120	99.35%	100.00%	10.00	4080.71	0
97	98.40%	91.87%	1.00	4972.28	3	Average	86.34%	68.06%	2.91	4939.38	8.84
57	100.00%	99.42%	1.67	14580.87	0	StdDev	28.06%	38.45%	3.10	6308.06	57.50
12	99.90%	99.87%	1.94	3805.79	0						

Table D.4: CPLEX results the maximisation of the total covering, in ascending order of optimality gaps. Each row represents one of the proposed instances. The first column identifies the instance, as described on Table 4.2. The second and third columns show the sizes of sets $\mathcal{F}$ and $\mathcal{U}$, respectively, after performing safe reductions (CPLEX's presolve). The next three columns show the optimality gap, calculated using Equation 4.5, the running time to solve this stage and the number of nodes of the solution enumeration tree that CPLEX had to expand, respectively.

Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes	Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes
1	76.80%	1.36%	0.00%	0.85	0	21	99.15%	50.00%	3.10%	3600.06	10925
2	76.80%	1.36%	0.00%	0.87	0	23	99.20%	99.96%	3.11%	3600.63	3926
3	79.05%	1.59%	0.00%	0.95	0	25	99.20%	100.00%	3.11%	3600.08	3249
4	83.25%	2.31%	0.00%	1.03	0	46	99.15%	50.00%	3.11%	3601.47	5601
5	81.35%	1.94%	0.00%	0.91	0	47	99.10%	79.06%	3.11%	3600.35	2315
26	78.35%	1.61%	0.00%	0.96	0	24	99.20%	100.00%	3.12%	3600.07	3351
27	78.35%	1.61%	0.00%	0.94	0	111	99.80%	50.00%	3.13%	3600.17	4954
28	81.70%		0.00%	0.96	0	113	99.80%	50.00%	3.13%	3600.44	4965
29	82.85%	2.62%	0.00%	0.92	0	67	98.85%	85.85%	3.14%	3602.48	455
30	82.85%	2.46%	0.00%	0.93	0	41	97.10%	49.99%	3.55%	3600.31	12118
51	32.95%	1.50%	0.00%	3.33	6	16	97.75%	50.00%	3.56%	3600.05	12177
52	32.95%	1.50%	0.00%	5.19	6	19	97.80%	99.99%	3.56%	3600.24	4412
53	40.35%	1.87%	0.00%	3.27	17	42	97.10%	62.96%	3.56%	3600.23	7460
55	83.40%	2.90%	0.00%	6.84	123	17	97.80%	66.65%	3.57%	3600.14	9023
76	39.50%	1.75%	0.00%	4.47	14	18	97.80%	93.05%	3.57%	3600.50	4836
77	39.50%	1.75%	0.00%	5.59	14	20	97.80%	100.00%	3.57%	3600.41	4437
78	42.90%	1.94%	0.00%	5.31	25	43	97.10%	87.81%	3.58%	3600.59	958
91	0.00%	0.00%	0.00%	2.66	0	44	97.10%	99.92%	3.64%	3600.33	1937
92	0.00%	0.00%	0.00%	2.11	0	48	99.15%	93.84%	3.98%	3600.56	1694
93	0.00%	0.00%	0.00%	2.36	0	49	99.15%	100.00%	3.98%	3602.36	1777
94	0.00%	0.00%	0.00%	2.31	0	50	99.15%	100.00%	3.98%	3605.07	1603
95	0.00%	0.00%	0.00%	1.13	0	66	99.15%	50.00%	4.14%	3600.38	5699
105	81.40%	3.01%	0.00%	10.61	310	69	99.15%	100.00%	4.15%	3601.16	884
112	0.00%	0.00%	0.00%	2.33	0	70	99.15%	100.00%	4.15%	3602.75	917

(*continued...*)

Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes	Inst	Red. $\mathcal{F}$	Red. $\mathcal{U}$	Gap	Time	Nodes
114	0.00%	0.00%	0.00%	4.26	0	68	99.15%	98.75%	4.16%	3605.92	531
117	0.00%	0.00%	0.00%	2.36	0	45	97.10%	99.97%	4.21%	3607.24	814
122	0.00%	0.00%	0.00%	2.22	0	72	98.30%	73.27%	4.80%	3600.21	2754
124	0.00%	0.00%	0.00%	2.14	0	74	98.30%	100.00%	4.80%	3600.56	628
54	83.40%	3.10%	0.01%	18.36	142	86	99.65%	50.00%	4.80%	3600.05	1491
56	61.95%	46.22%	0.01%	1797.54	1960	87	99.65%	50.00%	4.80%	3600.38	1561
79	43.85%	1.96%	0.01%	7.57	123	88	99.65%	91.42%	4.80%	3604.94	152
80	82.70%	3.17%	0.01%	7.87	22	89	99.65%	99.96%	4.80%	3611.02	47
102	78.85%	2.00%	0.01%	7.01	99	90	99.65%	100.00%	4.80%	3656.86	2
103	81.05%	2.27%	0.01%	7.58	248	71	98.30%	50.00%	5.12%	3600.02	10096
104	80.65%	2.22%	0.01%	8.86	69	73	98.30%	99.35%	5.13%	3600.35	3776
6	99.85%	49.89%	0.02%	3600.24	9595	75	98.30%	100.00%	5.13%	3600.15	3201
125	99.25%	100.00%	0.02%	3600.44	5852	61	97.75%	49.99%	5.35%	3600.53	6750
121	82.75%	52.12%	0.04%	3600.19	11202	36	99.60%	50.00%	5.78%	3600.07	9310
33	99.95%	48.48%	0.08%	3600.26	11274	37	99.60%	50.00%	5.78%	3600.62	9271
57	99.95%	49.42%	0.24%	3600.06	1987	81	98.65%	48.43%	5.78%	3601.55	1048
59	99.95%	49.42%	0.24%	3600.32	2835	82	98.65%	48.43%	5.78%	3601.27	1563
60	99.95%	49.42%	0.24%	3600.90	2563	85	98.65%	97.38%	5.78%	3613.83	114
58	99.95%	49.42%	0.25%	3600.34	1888	38	99.60%	98.11%	5.79%	3601.36	3531
8	99.95%	49.09%	0.31%	3600.34	8592	39	99.60%	99.98%	5.79%	3600.12	2901
7	99.90%	49.89%	0.45%	3600.04	6862	40	99.60%	100.00%	5.79%	3600.37	3435
12	99.85%	50.00%	0.59%	3600.42	13290	84	98.65%	92.64%	5.79%	3617.75	62
123	98.85%	100.00%	1.14%	3600.75	3413	63	97.75%	84.51%	5.85%	3602.58	2891
119	99.30%	100.00%	1.60%	3602.58	613	65	97.75%	99.94%	5.85%	3600.24	2221
116	99.30%	50.00%	1.61%	3603.45	502	64	97.75%	99.92%	6.39%	3602.39	168
118	99.30%	99.77%	1.61%	3602.41	290	11	99.30%	50.00%	7.23%	3600.32	8283
120	99.30%	100.00%	1.61%	3607.88	241	13	99.35%	83.98%	7.24%	3600.08	3435
115	99.65%	100.00%	2.12%	3604.25	641	14	99.35%	99.96%	7.24%	3600.33	2727
22	99.00%	73.04%	2.62%	3600.14	9441	15	99.35%	99.96%	7.24%	3600.84	2058
110	99.80%	95.60%	3.01%	3601.12	557	62	99.60%	50.00%	8.08%	3603.56	1932
96	98.35%	50.00%	3.06%	3601.67	2468	83	99.90%	49.97%	8.09%	3600.14	2205
97	98.35%	76.20%	3.06%	3601.22	825	9	99.90%	96.70%	9.81%	3602.18	3849
98	98.35%	99.95%	3.06%	3601.85	464	10	99.90%	96.76%	9.84%	3600.59	2011
99	98.35%	100.00%	3.06%	3600.13	76	31	99.90%	49.94%	11.89%	3600.05	9094
100	98.35%	100.00%	3.06%	3600.51	280	32	99.90%	49.94%	11.89%	3601.54	8920
106	99.80%	49.82%	3.09%	3601.04	1049	34	99.90%	97.38%	11.91%	3600.31	3734
107	99.80%	49.82%	3.09%	3600.98	607	35	99.90%	97.29%	11.92%	3600.03	2888
108	99.80%	49.82%	3.09%	3603.08	675	Average	84.67%	55.66%	3.00%	2580.31	2648.19
109	99.80%	95.66%	3.09%	3604.13	538	StdDev	29.19%	39.27%	2.99%	1621.13	3400.93
101	78.85%	2.00%	0.01%	3.96	99						

D.2 Chapter 5

The following tables complement the material presented on Chapter 5. The algorithms are identified as detailed on Table 7.1 and Table 7.2.

Table D.5 and Table D.6 detail the information summarized with averages on Table 7.6, including the results for each instance, as indexed on Table 4.2. They also detail, instance by instance, the results summarized with total values on Table 7.7, and graphically on Figure 7.1-Figure 7.5.

Table D.5: Metaheuristics's Results. Each row represents the results obtained with each of the proposed heuristics for the minimisation of the number of selected features. The first column identifies the instance as described on Table 4.2. The second shows the gaps, calculated with Equation 4.5, obtained using the exact approach. The next three the gaps obtained by the proposed heuristic using each of the proposed randomization methods: Algorithm 16, Algorithm 16 or Algorithm 15 chosen at random and only Algorithm 15. The last two columns also show results for the two most successful approaches (the last two) combined with Tabu Search. Bold values highlight the best of the five proposed heuristics. The values in italics improve the exact methods's best effort minimizing the number of selected features. The values marked with * improve the exact method's best effort also in maximising β. The values marked with ** improve the exact method's best effort for all considered objective functions.

		Heuristic									
Instance	**CPLEX**	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
		avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
1	0.00%	1.26%	0.28%	**1.00%**	0.15%	1.46%	0.32%	1.15%	0.20%	1.36%	0.23%
2	0.00%	1.35%	0.37%	**1.11%**	0.18%	1.31%	0.22%	1.15%	0.23%	1.27%	0.22%
3	0.00%	1.29%	0.40%	**1.09%**	0.29%	1.35%	0.25%	**1.09%**	0.37%	1.53%	0.23%
4	0.00%	1.34%	0.18%	**1.31%**	0.16%	1.93%	0.25%	1.38%	0.22%	1.95%	0.19%
5	0.00%	1.20%	0.23%	**1.15%**	0.21%	1.27%	0.24%	1.16%	0.14%	1.33%	0.26%
26	0.00%	**0.99%**	0.16%	1.06%	0.14%	1.30%	0.18%	**0.99%**	0.21%	1.44%	0.22%
27	0.00%	0.98%	0.11%	1.08%	0.17%	1.35%	0.13%	**0.92%**	0.19%	1.32%	0.28%
28	0.00%	1.51%	0.23%	1.51%	0.28%	1.98%	0.28%	**1.42%**	0.12%	1.80%	0.19%
29	0.00%	**0.89%**	0.20%	0.99%	0.16%	1.34%	0.26%	1.10%	0.17%	1.42%	0.13%
30	0.00%	**0.89%**	0.14%	0.91%	0.25%	1.29%	0.22%	0.91%	0.13%	1.32%	0.26%
51	0.00%	2.14%	0.44%	1.91%	0.29%	2.83%	0.42%	**1.87%**	0.23%	2.91%	0.30%
52	0.00%	2.12%	0.41%	2.07%	0.28%	2.76%	0.35%	**1.91%**	0.31%	2.84%	0.32%
53	0.00%	2.48%	0.29%	**1.89%**	0.19%	2.86%	0.25%	1.92%	0.27%	2.97%	0.29%
54	0.00%	1.89%	0.28%	**1.76%**	0.27%	2.61%	0.48%	1.99%	0.29%	2.57%	0.34%
55	0.00%	1.97%	0.35%	**1.92%**	0.36%	2.75%	0.46%	2.01%	0.34%	3.07%	0.31%
76	0.00%	2.00%	0.50%	1.86%	0.26%	2.46%	0.38%	**1.85%**	0.33%	2.27%	0.48%
77	0.00%	1.97%	0.47%	1.94%	0.24%	2.49%	0.39%	**1.85%**	0.21%	2.36%	0.44%
78	0.00%	2.04%	0.47%	**1.74%**	0.39%	2.46%	0.42%	2.07%	0.38%	2.43%	0.44%
79	0.00%	**1.78%**	0.38%	1.96%	0.31%	2.30%	0.27%	**1.78%**	0.25%	2.16%	0.35%
80	0.00%	2.07%	0.50%	1.91%	0.41%	2.21%	0.48%	**1.75%**	0.23%	2.35%	0.48%

(*continued...*)

		Heuristic									
Instance	**CPLEX**	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
		avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
101	0.00%	1.67%	0.18%	1.37%	0.26%	1.87%	0.25%	**1.24%**	0.29%	1.82%	0.27%
102	0.00%	1.69%	0.24%	**1.23%**	0.20%	1.89%	0.16%	**1.23%**	0.22%	1.87%	0.26%
103	0.00%	1.63%	0.31%	**1.48%**	0.24%	2.33%	0.35%	1.52%	0.34%	2.21%	0.24%
104	0.00%	1.61%	0.22%	1.32%	0.22%	1.90%	0.27%	**1.26%**	0.15%	1.85%	0.31%
105	0.00%	2.03%	0.40%	**1.39%**	0.21%	2.34%	0.43%	1.51%	0.30%	2.31%	0.30%
22	2.81%	4.07%	0.12%	3.60%	0.15%	**3.51%**	0.16%	3.53%	0.09%	3.60%	0.23%
41	3.23%	5.27%	0.34%	**4.51%**	0.18%	4.70%	0.24%	4.52%	0.23%	4.64%	0.26%
42	3.23%	5.34%	0.23%	4.81%	0.27%	4.79%	0.17%	**4.60%**	0.24%	4.74%	0.27%
43	3.23%	5.13%	0.26%	4.62%	0.17%	4.75%	0.31%	**4.58%**	0.20%	4.64%	0.21%
47	3.34%	4.35%	0.09%	4.01%	0.22%	**3.98%**	0.17%	**3.98%**	0.20%	4.05%	0.14%
44	3.54%	5.22%	0.34%	4.68%	0.24%	4.75%	0.26%	**4.65%**	0.41%	4.75%	0.22%
45	3.75%	5.40%	0.23%	**4.68%**	0.23%	4.74%	0.31%	4.69%	0.15%	4.84%	0.26%
46	3.88%	5.17%	0.19%	**4.62%**	0.17%	4.71%	0.25%	4.75%	0.24%	4.73%	0.20%
16	3.92%	5.31%	0.24%	4.96%	0.17%	**4.84%**	0.23%	5.05%	0.15%	4.87%	0.21%
17	3.92%	5.31%	0.18%	**4.89%**	0.19%	5.02%	0.21%	4.98%	0.18%	**4.89%**	0.21%
18	3.92%	5.33%	0.19%	4.91%	0.20%	4.91%	0.23%	5.00%	0.20%	**4.89%**	0.22%
19	3.92%	5.39%	0.21%	**4.86%**	0.17%	5.04%	0.21%	4.87%	0.16%	4.94%	0.19%
20	3.92%	5.24%	0.14%	**4.76%**	0.13%	4.91%	0.20%	4.83%	0.14%	4.77%	0.17%
21	3.97%	5.76%	0.19%	**5.01%**	0.24%	5.19%	0.26%	5.06%	0.22%	5.12%	0.21%
23	3.97%	5.82%	0.26%	**5.05%**	0.21%	5.08%	0.19%	5.27%	0.31%	5.07%	0.23%
24	3.97%	5.70%	0.16%	5.23%	0.20%	5.15%	0.20%	**5.04%**	0.26%	5.11%	0.23%
25	3.97%	5.77%	0.19%	5.12%	0.17%	**4.95%**	0.28%	5.17%	0.21%	5.04%	0.17%
66	4.34%	5.09%	0.19%	**4.66%**	0.22%	4.88%	0.21%	**4.66%**	0.25%	4.81%	0.13%
67	4.34%	5.12%	0.13%	4.82%	0.18%	4.93%	0.17%	**4.56%**	0.16%	4.87%	0.14%
68	4.34%	5.20%	0.23%	4.73%	0.21%	4.74%	0.18%	**4.70%**	0.19%	4.86%	0.16%
69	4.34%	5.19%	0.17%	**4.70%**	0.23%	4.81%	0.19%	4.74%	0.15%	4.77%	0.16%
70	4.34%	5.26%	0.20%	4.73%	0.22%	4.81%	0.15%	**4.72%**	0.25%	4.76%	0.19%
48	4.56%	5.17%	0.22%	4.71%	0.21%	4.76%	0.25%	**4.67%**	0.17%	4.76%	0.16%
49	4.56%	5.09%	0.18%	4.72%	0.25%	4.62%	0.16%	4.75%	0.20%	**4.58%**	0.27%
50	4.56%	5.16%	0.15%	4.65%	0.26%	4.81%	0.19%	4.65%	0.16%	**4.64%**	0.16%
61	5.82%	8.05%	0.20%	**7.30%**	0.29%	7.40%	0.31%	7.52%	0.24%	7.40%	0.23%
71	5.93%	6.99%	0.18%	**6.25%**	0.23%	6.32%	0.26%	6.40%	0.16%	6.38%	0.20%

(*continued...*)

		Heuristic									
Instance	**CPLEX**	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
		avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
73	5.93%	7.06%	0.13%	**6.25%**	0.16%	6.49%	0.24%	6.38%	0.28%	6.38%	0.20%
75	5.93%	6.85%	0.12%	6.33%	0.26%	**6.24%**	0.23%	6.48%	0.27%	6.41%	0.25%
36	6.33%	8.21%	0.38%	**7.35%**	0.35%	7.56%	0.32%	7.65%	0.31%	7.56%	0.19%
37	6.33%	8.31%	0.36%	7.61%	0.24%	7.59%	0.38%	**7.54%**	0.31%	7.77%	0.28%
38	6.33%	8.21%	0.37%	7.44%	0.26%	7.59%	0.28%	**7.41%**	0.30%	7.71%	0.29%
39	6.33%	8.37%	0.39%	**7.45%**	0.32%	7.74%	0.21%	7.49%	0.19%	7.47%	0.32%
40	6.33%	8.20%	0.39%	7.55%	0.30%	7.56%	0.26%	**7.47%**	0.29%	7.64%	0.39%
63	6.33%	7.88%	0.23%	7.45%	0.16%	**7.39%**	0.29%	7.57%	0.25%	7.49%	0.22%
65	6.33%	7.84%	0.35%	**7.37%**	0.35%	7.41%	0.19%	7.42%	0.27%	7.43%	0.26%
72	6.49%	7.00%	0.12%	***6.30%*	0.19%	***6.45%*	0.20%	*****6.22%***	0.31%	***6.34%*	0.20%
74	6.49%	7.06%	0.09%	***6.30%*	0.16%	***6.26%*	0.18%	*****6.21%***	0.25%	***6.31%*	0.14%
64	6.53%	7.90%	0.25%	7.47%	0.21%	**7.33%**	0.21%	7.34%	0.25%	7.47%	0.18%
11	7.96%	10.17%	0.26%	9.20%	0.37%	9.38%	0.28%	**9.16%**	0.49%	9.22%	0.43%
13	7.96%	9.89%	0.28%	9.38%	0.21%	**9.20%**	0.21%	9.23%	0.31%	9.63%	0.25%
14	7.96%	9.98%	0.43%	**9.16%**	0.44%	9.50%	0.23%	9.22%	0.37%	9.44%	0.41%
15	7.96%	10.03%	0.35%	**9.11%**	0.33%	9.42%	0.39%	9.48%	0.18%	9.29%	0.31%
96	9.05%	9.34%	0.13%	**8.76%*	0.31%	**8.78%*	0.27%	****8.75%***	0.25%	**8.86%*	0.25%
97	9.05%	9.35%	0.17%	*8.71%*	0.28%	***8.70%***	0.16%	*8.77%*	0.27%	*8.79%*	0.19%
98	9.05%	9.37%	0.21%	*8.81%*	0.17%	***8.67%***	0.17%	*8.74%*	0.30%	*8.70%*	0.35%
99	9.05%	9.37%	0.24%	*8.83%*	0.16%	***8.80%*	0.19%	***8.63%***	0.16%	***8.81%*	0.22%
100	9.05%	9.41%	0.24%	*****8.83%***	0.29%	***8.85%*	0.17%	*****8.83%***	0.22%	*****8.83%***	0.17%
62	9.11%	9.37%	0.27%	***8.93%*	0.24%	***9.05%*	0.30%	*****8.84%***	0.27%	***8.97%*	0.31%
116	9.84%	11.64%	0.26%	**10.70%**	0.38%	11.05%	0.30%	10.81%	0.24%	11.09%	0.40%
118	9.84%	11.49%	0.19%	**10.77%**	0.34%	11.05%	0.21%	10.91%	0.34%	10.91%	0.36%
119	9.84%	11.36%	0.30%	10.83%	0.30%	10.89%	0.22%	**10.72%**	0.39%	11.04%	0.22%
120	9.84%	11.38%	0.28%	**10.77%**	0.39%	10.97%	0.30%	10.78%	0.16%	11.07%	0.15%
86	13.61%	13.62%	0.28%	**13.23%*	0.42%	**13.36%*	0.36%	**13.19%*	0.28%	****13.15%***	0.29%
87	13.61%	***13.52%*	0.33%	***13.31%*	0.36%	***13.31%*	0.25%	*****13.13%***	0.34%	***13.50%*	0.24%
88	13.61%	13.64%	0.19%	***13.19%*	0.25%	***13.37%*	0.38%	*****13.12%***	0.32%	***13.31%*	0.24%
89	13.61%	13.67%	0.30%	*****13.25%***	0.38%	***13.36%*	0.32%	***13.37%*	0.19%	*****13.25%***	0.40%
90	13.61%	13.74%	0.20%	*13.18%*	0.22%	***13.47%*	0.28%	*****13.10%***	0.19%	***13.33%*	0.26%
115	14.22%	14.46%	0.31%	**13.95%*	0.33%	**14.05%*	0.43%	****13.90%***	0.32%	14.23%	0.32%

(*continued...*)

		Heuristic									
Instance	**CPLEX**	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
		avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
81	17.00%	*15.74%	0.32%	***14.98%**	0.41%	*15.30%	0.29%	*15.34%	0.34%	*15.42%	0.53%
82	17.00%	**15.67%	0.29%	****14.73%**	0.42%	**15.30%	0.30%	**14.96%	0.38%	**15.16%	0.33%
84	17.00%	*15.52%	0.37%	***14.80%**	0.52%	*15.25%	0.56%	*15.05%	0.57%	*15.10%	0.43%
85	17.00%	15.70%	0.36%	****14.98%**	0.45%	15.22%	0.41%	***14.98%**	0.54%	15.16%	0.50%
121	17.67%	12.41%	0.18%	11.95%	0.29%	11.89%	0.11%	***11.83%**	0.23%	11.93%	0.33%
123	17.67%	12.37%	0.22%	11.99%	0.16%	12.00%	0.19%	11.87%	0.23%	***11.81%**	0.22%
125	17.67%	12.36%	0.19%	11.82%	0.19%	11.89%	0.25%	***11.75%**	0.19%	11.86%	0.23%
7	19.85%	*19.45%	0.45%	*18.75%	0.62%	*18.99%	0.50%	***18.65%**	0.32%	*18.84%	0.38%
9	19.85%	*19.45%	0.27%	*18.75%	0.46%	***18.56%**	0.38%	*19.01%	0.42%	*18.80%	0.27%
10	19.85%	**19.35%	0.47%	**18.80%	0.29%	**18.99%	0.44%	****18.65%**	0.43%	**18.97%	0.34%
31	21.69%	**20.02%	0.27%	****19.22%**	0.58%	**19.80%	0.26%	**19.52%	0.44%	**19.58%	0.66%
32	21.69%	**20.23%	0.47%	****19.14%**	0.29%	**19.58%	0.33%	**19.54%	0.51%	**19.40%	0.46%
34	21.69%	*20.06%	0.60%	19.52%	0.42%	*19.72%	0.64%	***19.40%**	0.50%	*19.60%	0.57%
35	21.69%	**20.25%	0.50%	****19.16%**	0.65%	**19.46%	0.36%	**19.36%	0.27%	**19.68%	0.72%
6	22.66%	**19.37%	0.14%	**18.94%	0.33%	**19.01%	0.50%	**19.13%	0.41%	****18.88%**	0.40%
12	25.94%	**24.46%	0.47%	****23.65%**	0.41%	**23.75%	0.39%	**24.15%	0.23%	**23.73%	0.31%
83	26.83%	**24.44%	0.38%	****23.46%**	0.35%	**24.04%	0.28%	**23.61%	0.24%	**24.03%	0.42%
111	27.13%	*25.93%	0.34%	*25.19%	0.35%	*25.46%	0.31%	***25.06%**	0.29%	*25.35%	0.39%
113	27.13%	*25.81%	0.29%	*25.16%	0.28%	*25.23%	0.33%	***25.12%**	0.38%	*25.52%	0.37%
106	27.85%	**25.98%	0.43%	****25.45%**	0.63%	**25.95%	0.29%	**25.50%	0.33%	**25.82%	0.40%
107	27.85%	*25.85%	0.53%	*25.67%	0.44%	*26.01%	0.46%	***25.65%**	0.30%	*25.78%	0.38%
108	27.85%	*25.62%	0.58%	*25.58%	0.72%	*25.48%	0.62%	***25.44%**	0.47%	*25.87%	0.17%
109	27.85%	25.88%	0.44%	***25.37%**	0.52%	25.75%	0.50%	25.57%	0.44%	25.99%	0.42%
110	27.98%	25.55%	0.33%	25.52%	0.42%	*26.05%	0.51%	***25.43%**	0.50%	25.71%	0.29%
8	38.61%	**35.94%	0.27%	**35.28%	0.35%	**35.48%	0.33%	****35.03%**	0.46%	**35.25%	0.39%
33	39.57%	**36.04%	0.33%	**35.25%	0.48%	**35.64%	0.48%	****35.23%**	0.41%	**35.71%	0.52%
57	41.52%	*36.78%	0.33%	***36.36%**	0.41%	*36.72%	0.23%	*36.45%	0.17%	*36.56%	0.28%
58	41.52%	*36.47%	0.36%	***36.30%**	0.38%	*36.54%	0.40%	*36.43%	0.34%	***36.30%**	0.30%
59	41.52%	*36.65%	0.45%	*36.63%	0.49%	*36.72%	0.32%	***36.45%**	0.47%	*36.67%	0.33%
60	41.52%	*36.43%	0.44%	*36.23%	0.35%	*36.56%	0.42%	***36.18%**	0.45%	*36.47%	0.28%
56	43.87%	**36.52%	0.32%	****36.17%**	0.31%	**36.57%	0.26%	**36.39%	0.32%	**36.55%	0.38%
122	48.88%	12.47%	0.13%	11.98%	0.32%	12.03%	0.31%	***11.83%**	0.30%	11.92%	0.24%

(*continued...*)

		Heuristic									
Instance	**CPLEX**	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
		avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
124	48.88%	*12.53%*	0.25%	***11.82%***	0.21%	*11.90%*	0.31%	*12.01%*	0.21%	*12.05%*	0.20%
117	48.91%	*11.31%*	0.25%	*10.84%*	0.21%	*10.88%*	0.29%	***10.74%***	0.27%	*10.99%*	0.28%
91	52.98%	*11.85%*	0.25%	*11.06%*	0.27%	*11.22%*	0.18%	***11.05%***	0.34%	*11.33%*	0.19%
92	52.98%	*11.76%*	0.20%	***11.16%***	0.19%	*11.23%*	0.17%	*11.30%*	0.21%	*11.32%*	0.20%
93	52.98%	*11.72%*	0.18%	*11.21%*	0.23%	*11.22%*	0.14%	*11.29%*	0.17%	***11.17%***	0.24%
94	52.98%	*11.68%*	0.27%	*11.13%*	0.25%	*11.39%*	0.31%	***11.02%***	0.20%	*11.26%*	0.28%
95	52.98%	*11.78%*	0.23%	*11.12%*	0.34%	*11.31%*	0.12%	***11.12%***	0.25%	*11.12%*	0.32%
112	79.38%	*25.77%*	0.30%	*25.24%*	0.15%	*25.51%*	0.24%	***25.15%***	0.35%	*25.26%*	0.36%
114	79.38%	*25.77%*	0.38%	*25.22%*	0.33%	*25.25%*	0.42%	***25.07%***	0.33%	*25.34%*	0.38%
Average	14.30%	*11.15%*	0.28%	***10.66%***	0.29%	*10.87%*	0.29%	*10.67%*	0.28%	*10.86%*	0.29%
Standard Deviation	16.97%	9.34%	0.11%	9.28%	0.12%	9.27%	0.11%	9.29%	0.10%	9.25%	0.11%

Table D.6: Metaheuristics's Times. Each row represents the running times each of the proposed heuristics. The first column identifies the instance as described on Table 4.2. The next three the gaps, calculated with Equation 4.5, obtained by the proposed heuristic using each of the proposed randomization methods: Algorithm 16, Algorithm 16 or Algorithm 15 chosen at random and only Algorithm 15. The last two columns also show results for the two most successful approaches (the last two) combined with Tabu Search.

	Heuristic									
Instance	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
	avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
1	11.96	4.51	10.59	2.57	10.03	2.16	10.76	3.63	11.14	1.98
2	9.55	2.37	10.28	2.47	10.66	2.89	9.83	1.19	9.51	1.57
3	10.40	3.14	10.31	2.12	10.72	1.50	9.97	1.49	9.61	1.42
4	9.58	2.50	11.88	2.12	9.68	1.42	9.91	2.01	9.64	2.13
5	10.08	2.60	10.02	1.50	12.75	3.65	10.33	1.19	10.64	1.38
6	6.90	2.38	10.60	2.66	11.66	2.16	9.82	2.98	11.11	3.26
7	7.82	2.39	9.11	2.52	10.20	1.82	10.20	3.43	13.03	3.85
8	4.26	1.08	7.03	1.62	8.71	2.97	8.11	1.84	9.08	1.45
9	5.44	1.06	9.34	3.58	13.22	2.85	10.94	2.76	12.84	3.85
10	6.82	1.80	10.51	2.74	10.62	2.66	10.20	2.75	11.21	2.16
11	6.73	1.24	11.28	2.78	12.30	3.08	12.17	4.45	13.55	4.38
12	4.41	1.04	7.18	1.35	9.13	0.95	6.97	1.50	10.47	4.26
13	8.46	2.01	12.31	4.09	13.71	2.53	15.27	3.24	13.03	4.28
14	6.58	1.44	12.41	3.14	12.51	2.10	12.79	1.97	13.37	5.14
15	8.38	2.33	15.17	3.40	12.68	3.31	12.67	3.86	13.59	4.36
16	4.89	0.31	11.71	3.45	12.41	3.69	10.42	5.44	13.26	3.42
17	4.72	0.45	11.24	2.07	11.14	3.29	9.94	2.29	12.28	2.39
18	5.05	0.76	9.56	0.98	12.07	3.36	11.62	5.89	12.62	4.12
19	4.84	0.62	9.85	2.63	9.33	2.01	12.38	3.77	10.83	2.50
20	4.89	0.56	13.00	4.01	12.84	3.39	10.29	2.09	12.63	2.15
21	2.50	0.27	8.31	1.97	8.75	2.34	7.81	1.95	8.33	1.94
22	2.54	0.11	7.80	1.92	8.66	2.64	9.10	1.87	9.52	3.14
23	2.66	0.58	7.99	1.45	10.61	3.60	7.62	1.49	8.60	2.33
24	2.54	0.26	6.73	1.64	7.47	1.27	7.37	2.46	9.71	2.08
25	2.41	0.28	7.18	1.41	8.49	1.74	8.38	2.17	9.69	1.46
26	11.73	3.17	11.12	1.86	10.11	1.85	11.47	2.43	10.06	2.18
27	13.32	3.41	10.21	1.13	9.91	1.00	11.17	2.81	10.10	1.43
28	11.95	3.70	14.03	6.65	14.25	5.47	11.42	1.46	12.28	1.69
29	12.71	3.74	11.78	3.24	10.11	1.84	10.70	2.05	10.37	2.10
30	12.66	1.97	11.28	2.55	11.46	1.87	11.87	3.04	10.03	1.95
31	7.95	2.26	10.06	2.83	11.74	2.53	11.67	4.35	13.82	3.16
32	7.26	2.70	12.52	3.47	13.19	3.53	10.60	2.06	14.38	3.10
33	5.07	1.69	8.08	1.77	9.30	2.73	7.16	1.61	10.46	3.01
34	7.17	1.76	10.17	1.68	13.56	5.01	12.32	4.71	14.21	5.41
35	8.33	3.34	11.88	1.79	14.00	3.42	10.52	1.17	11.77	3.09
36	8.64	2.31	15.84	5.20	14.19	2.10	15.08	3.40	15.29	3.54
37	7.77	2.39	14.50	4.28	13.98	3.09	13.04	3.19	14.31	3.31
38	7.54	1.87	17.41	5.68	15.04	4.21	16.51	5.81	14.34	4.04
39	7.19	1.21	16.38	5.29	14.65	3.79	14.35	3.44	15.28	4.01
40	8.24	1.19	13.69	3.73	19.03	8.98	15.22	2.81	17.35	4.48
41	5.71	1.08	12.74	3.23	12.87	2.17	13.59	3.44	13.28	3.98

(*continued...*)

	Heuristic									
Instance	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
	avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
42	4.95	0.50	10.89	1.71	11.94	1.94	12.00	2.68	14.28	4.24
43	5.28	0.68	12.00	3.41	13.48	3.10	12.19	1.85	10.86	1.42
44	5.04	0.74	11.29	2.22	12.86	2.96	11.62	3.84	12.38	2.33
45	4.95	0.61	11.10	2.05	12.92	3.31	13.66	6.79	11.75	2.28
46	2.62	0.19	7.73	3.03	10.09	4.75	9.09	3.57	8.19	2.69
47	2.89	0.27	8.10	2.37	8.54	2.01	9.47	2.23	7.57	1.17
48	2.79	0.29	7.70	1.86	7.78	1.81	8.69	1.54	7.97	1.68
49	2.73	0.27	8.20	1.82	8.96	2.04	8.07	1.59	8.41	0.99
50	2.78	0.20	7.85	2.21	8.26	2.08	8.45	1.63	9.08	2.42
51	13.79	4.80	13.84	1.65	12.74	2.38	15.54	2.95	12.84	1.74
52	13.36	3.02	14.86	3.47	12.39	2.05	14.11	3.43	11.91	1.43
53	11.32	2.09	13.14	3.42	12.09	2.30	13.30	3.53	11.90	3.33
54	14.25	4.39	12.57	1.59	13.96	2.82	11.87	3.22	13.32	1.71
55	13.71	3.00	13.91	3.35	11.06	2.05	14.15	3.87	12.41	2.52
56	4.05	1.35	9.33	4.19	8.75	2.66	6.94	1.47	9.08	2.47
57	3.70	0.82	7.57	2.96	9.70	3.62	7.18	2.27	8.54	1.42
58	3.98	1.23	7.26	2.48	9.34	3.75	7.94	2.06	9.51	3.38
59	4.14	0.80	6.85	2.21	7.63	1.75	6.68	1.94	8.57	1.81
60	5.50	2.56	7.02	1.81	8.65	1.37	7.60	2.96	9.24	1.80
61	7.57	1.26	15.14	5.03	14.89	3.29	14.18	2.51	16.38	4.76
62	6.36	1.21	14.56	3.32	12.71	2.87	14.45	2.82	12.60	1.84
63	7.45	0.99	14.24	3.01	14.30	2.86	13.49	1.85	15.52	2.94
64	7.62	1.14	15.07	5.74	20.38	5.01	18.42	6.92	14.76	1.78
65	7.13	0.76	19.81	9.97	15.38	2.74	15.13	3.92	17.22	4.45
66	5.39	0.53	14.21	3.60	13.18	5.25	14.07	3.94	12.59	2.35
67	5.42	0.47	12.14	3.55	11.41	2.23	12.99	3.01	13.63	5.11
68	5.25	0.50	11.27	3.35	11.79	1.90	11.06	2.92	13.27	1.76
69	5.53	0.49	12.25	2.20	11.58	1.35	11.38	1.43	13.12	3.41
70	5.04	0.44	10.91	1.69	12.66	2.03	12.64	3.03	12.17	2.14
71	2.75	0.19	8.55	2.08	9.24	2.33	8.48	1.74	8.00	1.16
72	2.70	0.08	8.91	2.03	8.12	0.99	8.56	1.88	9.51	3.95
73	2.73	0.24	9.29	2.52	7.78	1.02	8.14	1.37	9.02	1.57
74	2.72	0.09	8.17	1.58	9.98	1.47	9.47	2.72	10.08	1.93
75	2.86	0.31	8.74	1.90	9.27	2.00	8.51	2.40	8.85	2.28
76	13.57	1.91	15.30	5.20	14.88	2.61	15.27	2.85	14.89	3.23
77	17.92	7.11	15.64	3.33	16.38	3.37	16.05	5.20	14.18	2.01
78	14.70	5.18	14.21	3.53	14.78	2.33	15.63	3.37	15.48	2.22
79	16.20	3.43	16.29	3.62	14.22	2.11	16.41	4.77	14.04	3.65
80	11.85	2.67	15.82	2.71	17.64	4.03	18.93	5.31	16.27	3.65
81	9.47	3.18	13.61	3.92	17.21	3.18	15.45	4.47	17.62	5.74
82	8.51	1.66	15.56	3.50	18.28	3.52	18.68	6.72	16.70	4.89
83	7.01	1.14	13.52	5.14	14.01	2.84	15.42	6.85	14.39	2.65
84	9.88	3.26	15.03	2.36	15.82	3.22	13.17	2.28	17.02	4.17
85	8.64	2.03	13.63	3.88	16.88	3.72	16.91	4.45	15.77	4.12
86	6.55	1.27	13.52	2.74	15.11	6.53	14.66	3.98	16.20	5.17
87	7.40	1.46	11.60	3.82	13.36	2.99	11.36	2.24	13.74	3.91
88	7.71	1.80	13.77	2.63	14.78	5.99	15.86	4.07	13.05	1.84
89	6.26	0.66	11.10	2.51	13.24	2.72	12.68	2.96	12.31	2.98
90	6.84	1.66	14.93	6.66	12.54	3.16	14.14	3.28	11.72	2.11
91	3.90	0.16	9.55	2.48	10.31	2.08	9.44	1.98	9.71	1.82
92	4.02	0.30	8.95	2.40	9.43	1.86	9.04	2.06	9.76	2.55
93	4.21	0.61	8.63	1.13	9.82	2.32	9.33	1.99	9.69	2.12

(*continued...*)

	Heuristic									
Instance	**MH1**		**MH2**		**MH3**		**MH4**		**MH5**	
	avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
94	4.01	0.45	11.00	3.62	9.64	3.39	9.55	2.21	9.61	1.62
95	4.29	0.67	8.95	2.03	8.29	0.80	9.86	2.60	11.34	4.06
96	2.92	0.09	6.77	1.38	8.33	1.85	7.18	1.54	7.03	1.11
97	2.89	0.11	7.68	1.22	9.14	2.03	7.32	1.57	8.14	1.28
98	3.01	0.19	6.86	1.09	8.10	1.86	7.25	1.06	8.26	1.49
99	2.94	0.20	6.84	0.88	7.60	1.27	7.52	1.31	7.61	0.68
100	2.92	0.19	6.68	1.46	7.68	1.25	7.68	1.39	9.86	2.91
101	12.03	2.56	17.03	3.64	12.78	1.82	16.11	2.73	14.11	2.22
102	12.79	2.43	17.64	5.75	13.78	2.33	16.04	3.45	14.51	3.93
103	13.78	2.16	16.87	3.71	15.53	1.92	20.47	7.50	16.71	3.77
104	11.97	1.84	16.62	2.96	14.02	2.19	19.03	4.46	12.33	2.43
105	13.63	2.53	19.86	4.20	17.26	3.27	18.10	2.15	16.98	4.66
106	7.69	1.94	12.43	3.13	15.98	4.06	11.38	3.51	13.83	3.47
107	9.70	2.95	14.01	7.27	13.69	2.83	10.85	2.45	14.60	3.97
108	12.19	4.23	14.62	6.50	18.04	6.77	10.86	3.23	15.53	2.26
109	9.61	3.62	12.32	2.29	14.55	3.69	13.24	4.09	14.86	3.92
110	9.92	2.52	11.79	3.45	12.73	4.03	12.44	3.06	16.70	2.90
111	5.20	0.93	11.35	3.21	10.43	2.98	9.84	2.24	12.76	3.28
112	5.42	0.91	10.54	3.36	9.19	1.96	10.77	2.29	10.77	2.10
113	5.47	1.20	8.90	1.77	11.74	2.57	9.60	1.57	9.96	2.84
114	5.58	1.32	8.47	1.98	10.27	3.73	9.93	2.85	10.32	3.32
115	7.46	1.91	13.36	2.22	12.89	3.70	12.21	2.98	14.72	3.92
116	4.98	0.37	10.01	1.91	9.00	1.07	9.59	1.25	9.79	1.29
117	4.80	0.50	7.90	0.62	9.79	1.15	8.50	1.12	8.92	1.14
118	4.74	0.62	8.85	1.88	9.70	1.68	9.01	1.80	10.23	2.69
119	4.75	0.44	9.45	1.50	9.70	1.66	10.11	2.88	9.28	1.38
120	5.02	0.83	10.11	2.87	8.78	1.38	8.93	1.15	9.92	1.80
121	2.78	0.14	6.43	1.35	7.13	1.06	7.25	0.93	7.81	1.74
122	2.75	0.11	5.80	0.81	8.20	2.14	6.28	1.08	7.72	1.95
123	2.87	0.25	6.19	1.08	7.30	1.96	6.66	1.28	8.41	2.60
124	2.93	0.23	6.52	0.96	7.34	2.05	5.79	0.84	6.68	1.35
125	2.91	0.22	6.68	1.16	6.87	1.42	6.98	0.95	7.67	2.64
Average	6.92	1.51	11.25	2.85	11.73	2.70	11.41	2.83	11.85	2.75
Standard Deviation	3.64	1.31	3.20	1.48	2.91	1.26	3.24	1.40	2.76	1.14

D.3 Chapter 6

The following tables complement the material presented on Chapter 6

Table D.7 and Table D.7 detail, for each instance, the average results presented on Table 7.11 and Table 7.12, respectively, according to the indexes presented on Table 4.2.

Table D.7: Dual Bounds for the Maximisation of β. Each row represents the results obtained with each proposed method to obtain upper bounds for the maximisation of β, in ascending order of optimality gap obtained with the exact approach. The first column identifies the instance as described on Table 4.2. The second shows the gaps obtained using the exact approach. The next two pairs show the optimality gaps calculated with Equation 4.6, using the greedy method detailed on Section 6.1.2 and Lagrangian Dual; and their respective running times.

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
1	0.00	1.78	1.00	0.28	0.00	207.35	7	2.00	3814.76	2.00	0.48	2.00	669.45
2	0.00	2.32	2.00	0.28	2.00	211.56	32	2.00	10710.56	2.00	0.53	2.00	658.64
3	0.00	9.85	6.00	0.29	3.00	309.07	37	2.00	5990.02	2.00	0.45	3.00	313.10
4	0.00	38.35	12.00	0.29	4.00	397.15	62	2.00	6506.55	2.00	0.47	4.00	649.92
5	0.00	11.73	15.00	0.29	4.00	381.24	82	2.00	3616.47	2.00	0.57	3.00	865.19
6	0.00	22.78	1.00	0.31	0.00	182.09	87	2.00	7140.02	2.00	0.46	4.00	221.57
11	0.00	29.01	1.00	0.41	0.00	284.82	107	2.00	4578.11	2.00	0.33	2.00	238.92
16	0.00	23.23	1.00	0.55	1.00	166.58	8	3.00	4849.14	8.00	0.28	3.00	269.50
21	0.00	25.13	1.00	0.59	1.00	485.74	58	3.20	12872.81	9.00	0.53	3.00	1279.18
22	0.00	5.39	0.00	0.65	3.00	417.05	33	3.24	4273.29	9.00	0.39	3.00	1069.24
26	0.00	1.82	1.00	0.29	0.00	203.26	98	4.00	6073.88	4.00	0.74	6.00	1048.13
27	0.00	2.53	2.00	0.29	2.00	202.04	23	4.00	6330.04	4.00	0.59	5.00	353.92
28	0.00	16.13	8.00	0.29	3.00	392.43	59	4.66	15533.25	17.00	0.29	4.00	1112.33
29	0.00	18.41	10.00	0.29	3.00	348.82	83	4.95	12114.71	7.00	0.40	5.00	1161.41
30	0.00	15.35	11.00	0.29	4.00	344.46	108	4.96	9847.98	8.00	0.59	5.00	1730.33
31	0.00	22.51	1.00	0.31	0.00	172.88	60	5.00	16169.38	17.00	0.30	5.00	888.72
36	0.00	30.03	1.00	0.44	1.00	152.13	68	5.00	5013.08	5.00	0.68	6.00	365.54
41	0.00	106.72	1.00	0.57	1.00	162.03	73	5.00	4569.11	5.00	0.67	6.00	1448.09
46	0.00	70.05	1.00	0.63	1.00	162.65	38	5.00	3698.03	6.00	0.45	5.00	337.59
51	0.00	3.35	1.00	0.29	0.00	169.88	13	5.00	5827.61	7.00	0.41	6.00	394.19
52	0.00	11.14	2.00	0.29	2.00	161.69	18	5.00	4717.43	5.00	0.56	6.00	381.33
53	0.00	28.22	8.00	0.29	3.00	298.54	43	5.00	6232.06	5.00	0.58	6.00	361.92
54	0.00	2015.07	14.00	0.29	3.00	310.88	118	5.00	7654.72	5.00	0.64	7.00	312.71
55	0.00	56.34	14.00	0.29	4.00	303.98	49	5.00	5112.37	5.00	0.63	6.00	358.91
56	0.00	43.94	1.00	0.29	0.00	134.44	113	5.37	6504.03	8.00	0.39	5.00	896.15
61	0.00	27.93	1.00	0.49	1.00	474.04	9	5.81	4485.28	12.00	0.31	6.00	290.03
67	0.00	708.18	0.00	0.68	3.00	316.21	63	5.87	6015.72	7.00	0.49	6.00	384.50

(*continued...*)

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
76	0.00	7.21	1.00	0.29	0.00	159.04	34	5.90	3928.71	13.00	0.32	6.00	891.99
77	0.00	15.54	2.00	0.29	2.00	149.03	14	5.99	4313.50	9.00	0.41	6.00	451.26
78	0.00	828.07	8.00	0.29	3.00	274.65	19	6.00	8807.05	6.00	0.61	7.00	1811.93
80	0.00	59.76	16.00	0.30	3.00	333.87	99	6.00	10580.99	6.00	0.71	9.00	1476.61
81	0.00	65.47	1.00	0.35	0.00	239.72	15	6.00	4524.88	15.00	0.41	7.00	351.21
91	0.00	3.39	0.00	0.60	0.00	252.05	48	6.00	9079.93	6.00	0.63	8.00	351.07
92	0.00	2.50	0.00	0.60	4.00	328.39	20	6.00	7858.85	10.00	0.99	7.00	1494.19
93	0.00	3.51	0.00	0.61	3.00	464.47	25	6.00	4481.78	10.00	0.59	7.00	445.45
94	0.00	3.32	0.00	0.60	4.00	561.12	109	6.11	30353.52	14.00	0.41	6.00	1817.24
95	0.00	1.85	0.00	0.82	3.00	2171.26	64	6.14	6429.11	11.00	0.49	6.00	412.32
101	0.00	3.14	1.00	0.29	0.00	152.72	10	6.32	4713.51	18.00	0.31	6.00	322.68
102	0.00	15.09	2.00	0.29	2.00	139.13	44	6.42	5503.90	7.00	0.81	7.00	1564.32
103	0.00	28.57	7.00	0.29	3.00	278.68	35	6.43	3989.54	14.00	0.32	6.00	375.22
104	0.00	83.82	13.00	0.29	3.00	368.87	88	6.50	24499.53	8.00	0.46	7.00	263.88
105	0.00	3190.20	14.00	0.30	3.00	349.04	110	6.77	22707.50	15.00	0.34	7.00	880.03
106	0.00	73.65	1.00	0.33	0.00	156.07	85	6.79	35976.86	16.00	0.37	7.00	920.57
111	0.00	23.87	1.00	0.38	0.00	652.13	39	7.00	4040.56	10.00	0.45	7.00	395.33
112	0.00	5.23	0.00	0.39	4.00	1145.06	45	7.00	4326.06	7.00	0.94	8.00	1413.95
114	0.00	4.46	0.00	0.38	4.00	568.82	50	7.00	7720.61	7.00	0.63	9.00	375.11
117	0.00	5.43	0.00	0.64	3.00	384.87	40	7.00	4745.19	10.00	0.45	7.00	444.61
121	0.00	122.89	0.00	0.66	0.00	165.87	84	7.00	16122.19	12.00	0.37	7.00	931.49
122	0.00	3.56	0.00	0.90	3.00	3018.81	125	7.00	4939.96	7.00	0.67	9.00	312.09
123	0.00	867.24	0.00	1.09	3.00	2003.70	24	7.00	4631.05	7.00	0.58	8.00	492.27
124	0.00	2.97	0.00	0.67	3.00	571.02	74	7.33	7016.72	9.00	0.67	8.00	1119.19
66	1.00	5008.28	1.00	0.67	1.00	166.18	65	7.46	4427.36	11.00	0.49	8.00	353.46
71	1.00	3741.13	1.00	0.66	1.00	521.01	90	7.86	15364.48	15.00	0.46	8.00	293.75
86	1.00	5650.63	1.00	0.46	1.00	467.21	69	7.89	23599.10	9.00	0.68	9.00	1524.69
96	1.00	3883.89	1.00	0.97	1.00	656.77	115	7.89	7055.09	13.00	0.47	8.00	258.30
116	1.00	4239.25	1.00	0.64	1.00	143.50	89	7.98	12724.39	11.00	0.46	8.00	279.66
79	1.00	3920.06	12.00	0.29	4.00	400.88	119	8.54	7115.17	9.00	0.64	9.00	283.80
17	1.00	3921.63	1.00	0.56	3.00	341.10	75	8.83	13277.99	9.00	0.68	10.00	1170.26
42	1.00	4360.84	1.00	0.57	3.00	322.30	70	9.00	5565.29	9.00	0.70	10.00	1984.72
47	1.00	4954.73	1.00	0.65	3.00	395.35	100	9.77	14770.70	11.00	0.71	10.00	1395.14
72	1.00	6093.50	1.00	0.67	4.00	1034.99	120	10.00	4080.71	10.00	0.64	11.00	283.36
97	1.00	4972.28	1.00	0.71	4.00	1236.49	Average	2.91	4939.38	5.85	0.49	4.14	594.45

(*continued...*)

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
57	1.67	14580.87	2.00	0.50	1.00	1032.05	StdDev	3.10	6308.06	5.15	0.18	2.80	522.91
12	1.94	3805.79	2.00	0.32	2.00	146.03							

Table D.8: Dual Bounds for the maximisation of the Total Covering. Each row represents the results obtained with each proposed method to obtain upper bounds for the maximisation of β, in ascending order of optimality gap obtained with the exact approach. The first column identifies the instance as described on Table 4.2. The second shows the gaps obtained using the exact approach. The next two pairs show the optimality gaps calculated with Equation 4.5, using greedy method detailed on Section 6.1.3 and Lagrangian Dual, and considering the best primal bound obtained with the exact method before one hour of computation time.

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
1	0.00%	0.85	19.15%	0.29	7.75%	3600.00	21	3.10%	3600.06	6.44%	0.64	5.21%	3600.00
2	0.00%	0.87	19.15%	0.29	7.72%	3600.00	23	3.11%	3600.63	6.44%	2.13	4.42%	3600.00
3	0.00%	0.95	19.31%	0.29	7.60%	3600.00	25	3.11%	3600.08	6.44%	2.38	4.12%	3600.00
4	0.00%	1.03	20.09%	1.64	8.51%	3600.00	46	3.11%	3601.47	6.57%	0.63	5.01%	3600.00
5	0.00%	0.91	19.30%	1.71	7.37%	3600.00	47	3.11%	3600.35	6.71%	2.56	5.19%	3600.00
26	0.00%	0.96	18.25%	0.29	7.30%	3600.00	24	3.12%	3600.07	6.44%	2.03	4.27%	3600.00
27	0.00%	0.94	18.25%	0.29	7.27%	3600.00	111	3.13%	3600.17	3.63%	1.11	3.19%	1636.18
28	0.00%	0.96	20.41%	2.19	8.02%	3600.00	113	3.13%	3600.44	3.63%	1.02	3.19%	1557.91
29	0.00%	0.92	18.75%	2.44	7.27%	3600.00	67	3.14%	3602.48	6.26%	2.41	4.56%	3600.00
30	0.00%	0.93	18.49%	2.42	7.07%	3600.00	41	3.55%	3600.31	8.26%	0.58	6.26%	3600.00
51	0.00%	3.33	17.31%	0.29	6.64%	3600.00	16	3.56%	3600.05	6.96%	0.56	5.29%	3600.00
52	0.00%	5.19	17.31%	0.30	6.61%	3600.00	19	3.56%	3600.24	6.96%	1.50	5.48%	3600.00
53	0.00%	3.27	17.55%	0.30	6.60%	3600.00	42	3.56%	3600.23	8.26%	0.58	6.16%	3600.00
55	0.00%	6.84	17.69%	0.30	6.42%	3600.00	17	3.57%	3600.14	6.96%	1.99	5.22%	3599.99
76	0.00%	4.47	5.28%	0.29	2.88%	3600.00	18	3.57%	3600.50	6.96%	2.18	5.06%	3600.00
77	0.00%	5.59	5.28%	0.30	2.86%	3600.00	20	3.57%	3600.41	6.96%	1.71	5.42%	3600.00
78	0.00%	5.31	5.38%	0.30	2.84%	3600.00	43	3.58%	3600.59	8.26%	2.41	5.92%	3600.00
91	0.00%	2.66	0.00%	0.61	0.00%	3.59	44	3.64%	3600.33	8.11%	0.88	6.50%	3600.00
92	0.00%	2.11	0.00%	2.32	0.00%	3.60	48	3.98%	3600.56	7.08%	0.64	5.38%	3600.00
93	0.00%	2.36	0.00%	2.24	0.00%	3.83	49	3.98%	3602.36	7.08%	2.46	5.03%	3600.00
94	0.00%	2.31	0.00%	2.37	0.00%	3.61	50	3.98%	3605.07	7.08%	2.46	5.07%	3600.00
95	0.00%	1.13	0.00%	1.62	0.00%	7.37	66	4.14%	3600.38	7.22%	0.67	5.51%	3600.00
105	0.00%	10.61	2.26%	1.56	0.99%	2439.25	69	4.15%	3601.16	7.22%	1.16	5.69%	3600.00
112	0.00%	2.33	0.00%	0.96	0.00%	6.37	70	4.15%	3602.75	7.22%	0.79	5.70%	3600.00
114	0.00%	4.26	0.00%	2.30	0.00%	3.33	68	4.16%	3605.92	7.22%	2.07	5.24%	3600.00
117	0.00%	2.36	0.00%	2.00	0.00%	3.59	45	4.21%	3607.24	8.56%	0.68	6.90%	3600.00
122	0.00%	2.22	0.00%	1.69	0.00%	11.04	72	4.80%	3600.21	6.67%	1.34	5.88%	3600.00

(*continued...*)

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
124	0.00%	2.14	0.00%	2.32	0.00%	3.59	74	4.80%	3600.56	6.67%	1.44	5.51%	3600.00
54	0.01%	18.36	18.39%	0.30	6.93%	3600.00	86	4.80%	3600.05	5.89%	0.50	4.99%	3600.00
56	0.01%	1797.54	3.32%	0.29	1.65%	3600.00	87	4.80%	3600.38	5.89%	0.47	4.94%	2073.22
79	0.01%	7.57	5.35%	1.84	2.74%	3600.00	88	4.80%	3604.94	5.89%	0.47	4.93%	1946.92
80	0.01%	7.87	5.34%	2.04	2.68%	3600.00	89	4.80%	3611.02	5.89%	0.47	4.91%	1791.77
101	0.01%	3.96	2.27%	0.29	1.06%	2410.70	90	4.80%	3656.86	5.89%	0.47	4.91%	1671.48
102	0.01%	7.01	2.27%	0.30	1.05%	2404.61	71	5.12%	3600.02	7.22%	0.71	6.35%	3600.00
103	0.01%	7.58	2.18%	0.30	0.97%	2463.67	73	5.13%	3600.35	7.22%	1.15	6.09%	3600.00
104	0.01%	8.86	2.27%	2.08	0.95%	2719.92	75	5.13%	3600.15	7.22%	1.78	5.86%	3600.00
6	0.02%	3600.24	5.18%	0.31	3.40%	3600.00	61	5.35%	3600.53	10.55%	0.54	8.11%	3600.00
125	0.02%	3600.44	0.18%	0.67	0.12%	655.13	36	5.78%	3600.07	12.00%	0.45	8.33%	3600.00
121	0.04%	3600.19	0.46%	0.66	0.36%	1090.00	37	5.78%	3600.62	12.00%	0.45	8.27%	3600.00
33	0.08%	3600.26	2.97%	1.09	2.29%	3600.00	81	5.78%	3601.55	7.80%	0.36	6.09%	3253.96
57	0.24%	3600.06	4.00%	1.10	2.80%	3600.00	82	5.78%	3601.27	7.80%	0.58	6.48%	3600.00
59	0.24%	3600.32	4.00%	0.75	2.56%	3600.00	85	5.78%	3613.83	7.80%	1.00	6.25%	3600.00
60	0.24%	3600.90	4.00%	1.33	2.55%	3600.00	38	5.79%	3601.36	12.00%	0.45	7.96%	3600.00
58	0.25%	3600.34	4.00%	0.63	2.81%	3600.00	39	5.79%	3600.12	12.00%	2.01	7.71%	3600.00
8	0.31%	3600.34	3.79%	0.29	2.19%	3600.00	40	5.79%	3600.37	12.00%	1.95	7.61%	3600.00
7	0.45%	3600.04	6.46%	0.81	5.20%	3600.00	84	5.79%	3617.75	7.80%	1.03	6.27%	3600.00
12	0.59%	3600.42	3.72%	0.33	1.88%	3600.00	63	5.85%	3602.58	10.79%	2.22	7.21%	3600.00
123	1.14%	3600.75	1.36%	1.08	1.18%	3357.90	65	5.85%	3600.24	10.79%	0.50	6.91%	3600.00
119	1.60%	3602.58	2.21%	0.65	1.65%	1164.14	64	6.39%	3602.39	11.24%	1.90	7.37%	3600.00
116	1.61%	3603.45	2.21%	0.64	1.67%	1693.37	11	7.23%	3600.32	14.24%	0.42	9.49%	3600.00
118	1.61%	3602.41	2.21%	0.66	1.66%	1398.99	13	7.24%	3600.08	14.24%	2.25	9.18%	3600.00
120	1.61%	3607.88	2.21%	0.65	1.65%	1161.59	14	7.24%	3600.33	14.24%	2.29	8.81%	3600.00
115	2.12%	3604.25	2.84%	0.47	2.19%	1142.39	15	7.24%	3600.84	14.24%	1.98	8.80%	3600.00
22	2.62%	3600.14	5.69%	2.03	4.44%	3600.00	62	8.08%	3603.56	12.16%	1.32	9.81%	3600.00
110	3.01%	3601.12	3.89%	1.30	3.12%	3026.12	83	8.09%	3600.14	9.99%	1.30	8.49%	3599.99
96	3.06%	3601.67	3.69%	1.77	3.24%	3600.00	9	9.81%	3602.18	16.36%	0.32	11.92%	3600.00
97	3.06%	3601.22	3.69%	1.19	3.20%	3600.00	10	9.84%	3600.59	16.36%	1.96	11.91%	3600.00
98	3.06%	3601.85	3.69%	1.18	3.15%	3600.00	31	11.89%	3600.05	17.93%	0.32	13.30%	3600.00
99	3.06%	3600.13	3.69%	2.13	3.12%	3600.00	32	11.89%	3601.54	17.93%	1.16	15.05%	3600.00
100	3.06%	3600.51	3.69%	1.20	3.12%	3600.00	34	11.91%	3600.31	17.93%	0.95	14.26%	3600.00
106	3.09%	3601.04	3.99%	0.33	3.23%	1222.16	35	11.92%	3600.03	17.93%	1.86	12.99%	3600.00
107	3.09%	3600.98	3.99%	0.33	3.23%	1200.76	avg	3.00%	2580.31	7.82%	1.15	4.96%	2997.86

(*continued...*)

	CPLEX		*Greedy*		*Lagrangian*			*CPLEX*		*Greedy*		*Lagrangian*	
Instance	gap	time	gap	time	gap	time	Instance	gap	time	gap	time	gap	time
108	3.09%	3603.08	3.99%	1.00	3.30%	3600.00	stdev	2.99%	1621.13	5.76%	0.75	3.19%	1148.44
109	3.09%	3604.13	3.99%	1.37	3.29%	3600.00							

Bibliography

T. Abeel, Y. Saeys, P. Rouzé, and Y. de Peer. ProSOM: core promoter prediction based on unsupervised clustering of DNA physical profiles. Bioinformatics, 24(13):i24, 2008.

D. W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. International Journal of Man-Machine Studies, 36(2):267–287, 1992.

D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. Machine learning, 6(1):37–66, 1991.

A. Arefin, M. Inostroza-Ponta, L. Mathieson, R. Berretta, and P. Moscato. Clustering nodes in large-scale biological networks using external memory algorithms. Algorithms and Architectures for Parallel Processing, pages 375–386, 2011.

E. Balas and M. C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. Operations Research, 44(6):875–890, 1996.

E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. Math. Program. Study, 12:37–60, 1980.

P. Baldi and S. r. Brunak. Bioinformatics: The Machine Learning Approach. MIT Press, 2001.

P. Baldi and A. D. Long. A Bayesian framework for the analysis of microarray expression data: regularized t-test and statistical inferences of gene changes. Bioinformatics, 17(6): 509, 2001.

J. Beasley and K. Jørnsten. Enhancing an algorithm for set covering problems. European Journal of Operational Research, 58(2):293–300, 1992.

J. E. Beasley. An algorithm for set covering problem. European Journal of Operational Research, 31(1):85–93, 1987.

J. E. Beasley. A lagrangian heuristic for set-covering problems. Naval Research Logistics, 37 (1):151–164, 1990a.

J. E. Beasley. OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society, 41(11):1069–1072, 1990b.

J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. European Journal of Operational Research, 94(2):392–404, 1996.

A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. Journal of Computational Biology, 7(3-4):559–583, 2000.

B. Bergeron. Bioinformatics Computing. Prentice Hall, 2002.

P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks. Discrete Applied Mathematics, 155(6-7):733–749, 2007.

R. Berretta, A. Mendes, and P. Moscato. Selection of Discriminative Genes in Microarray Experiments Using Mathematical Programming. Journal of Research and Practice in Information Technology, 39(4):287–299, 2007.

R. Berretta, W. Costa, and P. Moscato. Combinatorial optimization models for finding genetic signatures from gene expression datasets. Methods Mol Biol, 453:363–377, 2008.

D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. Mathematical Programming, 80(1):63–89, 1998.

R. Blanco, P. Larranaga, I. Inza, and B. Sierra. Gene selection for cancer classification using wrapper approaches. International Journal of Pattern Recognition and Artificial Intelligence, 18(8):1373–1390, 2004.

T. Bo and I. Jonassen. New feature subset selection procedures for classification of expression profiles. Genome biology, 3(4):17, 2002.

P. C. Boutros and A. B. Okey. Unsupervised pattern recognition: an introduction to the whys and wherefores of clustering microarray data. Briefings in bioinformatics, 6(4):331–343, 2005.

S. Boyd, L. Xiao, and A. Mutapcic. Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter, 2004, 2003.

R. Breitling, P. Armengaud, A. Amtmann, and P. Herzyk. Rank products: a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments. FEBS letters, 573(1-3):83–92, 2004.

D. Brinza and A. Zelikovsky. Design and validation of methods searching for risk factors in genotype case-control studies. J Comput Biol, 15(1):81–90, 2008.

D. Brinza, J. He, and A. Zelikovsky. Combinatorial search methods for multi-SNP disease association. Conf Proc IEEE Eng Med Biol Soc, 1:5802–5805, 2006.

S. Busygin, O. A. Prokopyev, and P. M. Pardalos. Feature Selection for Consistent Biclustering via Fractional 0–1 Programming. Journal of Combinatorial Optimization, (10):7–21, 2005.

A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. Operations Research, 47(5):730–743, 1999.

M. Caserta. Tabu search-based metaheuristic algorithm for large scale set covering problems. Metaheuristics: progress in complex systems optimization, 39:43, 2007.

S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. Mathematical Programming, 81(2):215–228, 1998.

H. Chan, B. Sahiner, R. Wagner, and N. Petrick. Classifier design for computer-aided diagnosis: Effects of finite sample size on the mean performance of classical and neural network classifiers. Medical physics, 26:2654, 1999.

U. Chandran, C. Ma, R. Dhir, M. Bisceglia, M. Lyons-Weiler, W. Liang, G. Michalopoulos, M. Becich, and F. Monzon. Gene expression profiles of prostate cancer reveal involvement of multiple molecular pathways in the metastatic process. BMC Cancer, 7(1):64, 2007. ISSN 1471-2407. doi: 10.1186/1471-2407-7-64. URL http://www.biomedcentral.com/1471-2407/7/64.

J. Charlesworth, J. Curran, M. Johnson, H. Göring, T. Dyer, V. Diego, J. Kent, M. Mahaney, L. Almasy, J. MacCluer, et al. Transcriptomic epidemiology of smoking: the effect of smoking on gene expression in lymphocytes. BMC medical genomics, 3(1):29, 2010.

V. Chvatal. A greedy heuristic for the set-covering problem. Mathematics of operations research, 4(3):233–235, 1979.

M. Clark, R. Johnston, M. Inostroza-Ponta, A. Fox, E. Fortini, P. Moscato, M. Dinger, and J. Mattick. Genome-wide analysis of long noncoding rna stability. Genome research, 22(5): 885–898, 2012.

J. Cohen. Bioinformatics - An Introduction for Computer Scientists. ACM Computing Surveys, 36(2):122–158, 2004.

T. H. Cormen. Introduction to algorithms. MIT Press, 2 edition, 2001.

C. Cotta and P. Moscato. The k-Feature Set Problem is W[2]-complete. Journal of Computer and System Sciences, 67(4):686–690, December 2003a.

C. Cotta and P. Moscato. A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny. Biosystems, 72(1-2): 75–97, 2003b.

C. Cotta, C. Sloper, and P. Moscato. Evolutionary Search of Thresholds for Robust Feature Set Selection: Application to the Analysis of Microarray Data. In G. R. Raidl, S. Cagnoni,

J. Branke, D. W. Corne, R. Drechsler, J. Jin Y., M. C.G., M. P., R. E., S. F., and G. G.D. Squillero, editors, Applications of Evolutionary Computing, Lecture Notes in Computer Science, pages 21–30. Springer Berlin / Heidelberg, 2004.

M. Dash and H. Liu. Feature Selection for Classification. Intelligent Data Analysis, 1:131–156, 1997.

S. Davies and S. Russell. NP-Completeness of Searches for Smallest Possible Feature Sets. In AAAI Symposium on Intelligent Relevance, pages 37–39. AAAI Press, 1994.

L. Davis et al. Handbook of genetic algorithms, volume 115. 1991.

R. Díaz-Uriarte and A. Andrés. Gene selection and classification of microarray data using random forest. BMC bioinformatics, 7(1):3, 2006.

C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression dataMinimum redundancy feature selection from microarray gene expression data. J Bioinform Comput Biol;, 3(2):185–205, 2005. ISSN 0219-7200.

R. G. Downey and M. R. Fellows. Parameterized Complexity. Monographs in computer science. Springer, 1999.

S. Dudoit, J. P. Shaffer, and J. C. Boldrick. Multiple hypothesis testing in microarray experiments. Statistical Science, 18(1):71–103, 2003.

B. Duval and J. K. Hao. Advances in metaheuristics for gene selection and classification of microarray data. Briefings in Bioinformatics, 2009.

B. Duval, J. K. Hao, and J. C. Hernandez Hernandez. A memetic algorithm for gene selection and molecular classification of cancer. In Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pages 201–208. ACM, 2009.

B. Efron, R. Tibshirani, J. D. Storey, and V. Tusher. Empirical Bayes analysis of a microarray experiment. Journal of the American Statistical Association, 96(456):1151–1161, 2001.

U. M. Fayyad and K. B. Irani. On the Handling of Continuous-Valued Attributes in Decision Tree Generation. Mach. Learn., 8(1):87–102, 1992.

U. M. Fayyad and K. B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learnin 13th International Joint Conference on Artificial Intelligence. Morgan Kaufmann, 1993.

U. Feige. A threshold of ln n for approximating set cover. Journal of the ACM (JACM), 45 (4):634–652, 1998.

T. Feo and M. Resende. Greedy randomized adaptive search procedures. Journal of Global Optimization, 6(2):109–133, 1995.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters, 8(2):67–71, 1989.

M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. Management Science, 27:1–18, 1981.

J. Flum and M. Grohe. Parameterized Complexity Theory. Texts in theoretical computer science - E A T C S MONOGRAPHS ON THEORETICAL COMPUTER SCIENCE. Birkh{ä}user, 2006.

R. J. Fox and M. W. Dimmic. A two-sample Bayesian t-test for microarray data. BMC bioinformatics, 7(1):126, 2006.

D. Gamberger and N. Lavrac. Conditions for Occam's Razor applicability and noise elimination. LECTURE NOTES IN COMPUTER SCIENCE, pages 108–123, 1997.

M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, 1979.

A. Geoffrion. Lagrangian relaxation for integer programming. Mathematical Programming Study, 2:82–114, 1974.

O. Gevaert, F. Smet, D. Timmerman, Y. Moreau, and B. Moor. Predicting the prognosis of breast cancer by integrating clinical and microarray data with Bayesian networks.

Bioinformatics, 22(14):e184—-190, 2006. ISSN 1460-2059. doi: 10.1093/bioinformatics/btl230.

F. Glover. Tabu search-part ii. ORSA Journal on computing, 2(1):4–32, 1990.

F. Glover and G. A. Kochenberger. Handbook of metaheuristics. Springer, 2003.

F. Glover and M. Laguna. Tabu search, volume 1. Springer, 1998.

F. Glover et al. Tabu search-part i. ORSA Journal on computing, 1(3):190–206, 1989.

M. Gómez Ravetti and P. Moscato. Identification of a 5-protein biomarker molecular signature for predicting Alzheimer's disease. PLoS One, 3(9):e3111, 2008.

M. Gómez Ravetti, R. Berretta, and P. Moscato. Novel Biomarkers for Prostate Cancer Revealed by (α,β)-k-Feature Sets. In Foundations of Computational Intelligence Volume 5, volume 5, chapter 7, pages 149–175. Springer Berlin / Heidelberg, 2009.

T. Gonzalez. Handbook of approximation algorithms and metaheurististics, volume 10. Chapman & Hall, 2007.

J. González-Barrios and A. Quiroz. A clustering procedure based on the comparison between the< i> k</i> nearest neighbors graph and the minimal spanning tree. Statistics & probability letters, 62(1):23–34, 2003.

V. Goss Tuscher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. Proc Natl Acad Sci, 98:5116–5121, 2001.

T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. European Journal of Operational Research, 101(1):81–92, 1997.

M. Guignard. Efficient cuts in lagrangean 'relax-and-cut' schemes. European Journal of Operation Research, 105(1):216–223, 1998.

M. Guignard. Lagrangean relaxation. TOP: Sociedad de Estatistica e Investigación Operational, 11(2):151–228, 2003.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. Machine learning, 46(1):389–422, 2002.

M. A. Hall. Correlation-based Feature Subset Selection for Machine Learning. PhD thesis, Department of Computer Science, University of Waikato, 1999.

N. G. Hall and D. S. Hochbaum. A fast approximation algorithm for the multicovering problem. Discrete Applied Mathematics, 15(1):35 – 40, 1986. ISSN 0166-218X. doi: 10.1016/0166-218X(86)90016-8. URL http://www.sciencedirect.com/science/article/pii/0166218X86900168.

R. Hassin and A. Levin. A better-than-greedy approximation algorithm for the minimum set cover problem. SIAM Journal on Computing, 35(1):189–200, 2006.

M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. Operations Research, 18:1138–1162, 1970.

M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. Mathematical Programming, 1(1):1436–4646, 1971.

Q. Hua, D. Yu, F. Lau, and Y. Wang. Exact algorithms for set multicover and multiset multicover problems. Algorithms and Computation, pages 34–44, 2009.

Q. Hua, Y. Wang, D. Yu, and F. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. Theoretical Computer Science, 411(26-28): 2467–2474, 2010.

M. Inostroza-Ponta. An Integrated and Scalable Approach Based on Combinatorial Optimization Techniques for the PhD thesis, The University of Newcastle, 2008.

M. Inostroza-Ponta, R. Berretta, and P. Moscato. Qapgrid: A two level qap-based approach for large-scale data analysis and visualization. PloS one, 6(1):e14468, 2011.

I. Inza, P. Larrañaga, R. Blanco, and A. J. Cerrolaza. Filter versus wrapper gene selection approaches in DNA microarray domains. Artificial Intelligence in Medicine, 31(2):91–103, 2004.

L. W. Jacobs and M. J. Brusco. A simulated annealing-based heuristic for the set-covering problem. In Proceedings of Decision Sciences Institute, volume 2, pages 1189–1191, 1994.

P. Jafari and F. Azuaje. An assessment of recently published gene expression data analyses: reporting experimental design and statistical factors. BMC Medical Informatics and Decision Making, 6(1):27, 2006.

H. Jiang, Y. Deng, H. S. Chen, L. Tao, Q. Sha, J. Chen, C. J. Tsai, and S. Zhang. Joint analysis of two microarray gene-expression data sets to select lung adenocarcinoma marker genes. BMC bioinformatics, 5(1):81, 2004.

R. M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, pages 85–103, 1972.

R. M. Karp. Mathematical Challenges from Genomics and Molecular Biology. Notices of the AMS, 49(5):544–553, 2002.

I. S. Kohane, A. Kho, and A. J. Butte. Microarrays for an Integrative Genomics. The MIT Press, August 2002.

S. G. Kolliopoulos and N. E. Young. Approximation algorithms for covering/packing integer programs. Journal of Computer and System Sciences, 71(4):495–505, 2005.

G. Lan, G. W. DePuy, and G. E. Whitehouse. An effective and simple heuristic for the set covering problem. European journal of operational research, 176(3):1387–1403, 2007.

G. Lancia. Mathematical Programming in Computational Biology: an Annotated Bibliography. Algorithms, 1:100–129, 2008.

P. Langley and S. Sage. Oblivious decision trees and abstract cases, 1994.

P. Langley and S. Sage. Induction of selective Bayesian classifiers. Institute for the Study of, 19950328:163, 1995.

P. Langley and S. Sage. Scaling to domains with irrelevant features. Computational Learning Theory and Natural Learning Systems: Making learning systems practical, page 51, 1997.

A. M. Lesk. Introduction to Bioinformatics. Oxford University Press, 2002.

T. G. Lesnick, S. Papapetropoulos, D. C. Mash, J. Ffrench-Mullen, L. Shehadeh, M. de Andrade, J. R. Henley, W. A. Rocca, J. E. Ahlskog, and D. M. Maraganore. A Genomic Pathway Approach to a Complex Disease: Axon Guidance and Parkinson Disease. PLoS Genetics, 3(6):e98, 2007. doi: 10.1371/journal.pgen.0030098. URL http://dx.plos.org/10.1371%2Fjournal.pgen.0030098.

L. Li, C. R. Weinberg, T. A. Darden, and L. G. Pedersen. Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the GA/KNN method. Bioinformatics, 17(12):1131, 2001.

H. E. Lockstone, L. W. Harris, J. E. Swatton, M. T. Wayland, A. J. Holland, and S. Bahn. Gene expression profiling in the adult Down syndrome brain. Genomics, 90(6):647–660, 2007. ISSN 0888-7543. doi: 10.1016/j.ygeno.2007.08.005. URL http://www.sciencedirect.com/science/article/pii/S0888754307002054.

L. A. N. Lorena, B. Lopes, and Others. A surrogate heuristic for set covering problems. European Journal of Operational Research, 79(1):138–150, 1994.

H. Lourenço, O. Martin, and T. Stützle. Iterated local search. Handbook of metaheuristics, pages 320–353, 2003.

L. Lovasz. On the ratio of optimal integral and fractional covers. Discrete mathematics, 13 (4):383–390, 1975.

A. Lucena. Non delayed relax-and-cut algorithms. Annals of Operations Research, 140(1): 375–410, 2005.

C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. Journal of the ACM (JACM), 41(5):981, 1994.

N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is bioinformatics? A proposed definition and overview of the field. Methods Inf Med, 40(4):346–358, 2001.

S. Ma and J. Huang. Regularized ROC method for disease classification and biomarker selection with microarray data. Bioinformatics, 21(24):4356, 2005.

H. Mamitsuka. Selecting features in microarray classification using ROC curves. Pattern Recognition, 39(12):2393–2404, 2006.

F. Margot. Symmetry in integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, 50 Years of Integer Programming 1958-2008, pages 647–686. Springer Berlin Heidelberg, 2010. ISBN 978-3-540-68279-0.

D. Mellor, E. Prieto, L. Mathieson, and P. Moscato. A kernelisation approach for multiple d-hitting set and its application in optimal multi-drug therapeutic combinations. PloS one, 5(10):e13055, 2010.

A. Mendes, R. J. Scott, and P. Moscato. Microarrays—Identifying Molecular Portraits for Prostate Tumors with Different Gleason Patterns. In Clinical Bioinformatics, pages 131–151. REVIEW, 2008.

N. Mladenovic and P. Hansen. Variable neighborhood search. Computers & OR, 24(11): 1097–1100, 1997.

L. C. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: a survey and experimental evaluation. In Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference, 2002. ISBN 0-7695-1754-4.

J. Moore, F. Asselbergs, and S. Williams. Bioinformatics challenges for genome-wide association studies. Bioinformatics, 26(4):445–455, 2010.

P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program, C3P Report, 826:1989, 1989.

P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In Handbook of metaheuristics, pages 105–144. Springer, 2003.

P. Moscato, C. Cotta, and A. Mendes. Memetic Algorithms. In New optimization techniques in engineering. Springer Verlag, 2004.

P. Moscato, L. Mathieson, A. Mendes, and R. Berretta. The electronic primaries: predicting the U.S. presidency using feature selection with safe data reduction. In ACSC '05: Proceedings of the Twenty-eighth Australasian conference on Computer Science, pages 371–379, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

G. L. Nemhauser and L. A. Wolsey. Integer programming. Elsevier North-Holland, Inc., 1989.

F. Neri, C. Cotta, and P. Moscato. Handbook of Memetic Algorithms, volume 379. Springer Verlag, 2012.

M. A. Newton, C. M. Kendziorski, C. S. Richmond, F. R. Blattner, and K. W. Tsui. On differential variability of expression ratios: improving statistical inference about gene expression changes from microarray data. Journal of Computational Biology, 8(1):37–52, 2001.

C. H. Ooi and P. Tan. Genetic algorithms applied to multi-class prediction for the analysis of gene expression data. Bioinformatics, 19(1):37, 2003.

W. Pan. On the use of permutation in and the performance of a class of nonparametric methods to detect differential gene expression. Bioinformatics, 19(11):1333, 2003.

P. J. Park, M. Pagano, and M. Bonetti. A nonparametric scoring algorithm for identifying informative genes from microarray data. In Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, page 52, 2001.

L. Pessoa, M. Resende, and C. Ribeiro. A hybrid lagrangean heuristic with grasp and path-relinking for set k-covering. Technical report, 2010.

L. Pessoa, M. Resende, and C. Ribeiro. Experiments with lagrasp heuristic for set k-covering. Optimization Letters, pages 1–13, 2011.

S. Rajagopalan and V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. SIAM J. Comput., 28(2):525–540, 1998.

S. Ray, M. Britschgi, C. Herbert, Y. Takeda-Uchimura, A. Boxer, K. Blennow, L. F. Friedman, D. R. Galasko, M. Jutel, A. Karydas, J. A. Kaye, J. Leszek, B. L. Miller, L. Minthon, J. F. Quinn, G. D. Rabinovici, W. H. Robinson, M. N. Sabbagh, Y. T. So, D. L. Sparks, M. Tabaton, J. Tinklenberg, J. A. Yesavage, R. Tibshirani, and T. Wyss-Coray. Classification and prediction of clinical Alzheimer's diagnosis based on plasma signaling proteins. Nat Med, 13(11):1359–1362, 2007.

C. Riveros, D. Mellor, K. Gandhi, F. McKay, M. Cox, R. Berretta, S. Vaezpour, M. Inostroza-Ponta, S. Broadley, R. Heard, et al. A transcription factor map as revealed by a genome-wide gene expression analysis of whole-blood mrna transcriptome in multiple sclerosis. PloS one, 5(12):e14176, 2010.

R. Rizzi, P. Mahata, L. Mathieson, and P. Moscato. Hierarchical clustering using the arithmetic-harmonic cut: complexity and experiments. PloS one, 5(12):e14067, 2010.

M. Rocha de Paula, M. Ravetti, R. Berretta, and P. Moscato. Differences in abundances of cell-signalling proteins in blood reveal novel biomarkers for early detection of clinical alzheimer's disease. PloS One, 6(3):e17481, 2011.

O. A. Rosso, A. Mendes, R. Berretta, J. A. Rostas, M. Hunter, and P. Moscato. Distinguishing childhood absence epilepsy patients from controls by the analysis of their background brain electrical activity (II): a combinatorial optimization approach for electrode selection. J Neurosci Methods, 181(2):257–267, 2009.

C. R. Scherzer, A. C. Eklund, L. J. Morse, Z. Liao, J. J. Locascio, D. Fefer, M. A. Schwarzschild, M. G. Schlossmacher, M. A. Hauser, J. M. Vance, L. R. Sudarsky, D. G. Standaert, J. H. Growdon, R. V. Jensen, and S. R. Gullans. Molecular markers of early Parkinson's disease based on gene expression in blood. Proceedings of the National Academy of Sciences, 104(3):955–960, 2007. doi: 10.1073/pnas.0610204104. URL http://www.pnas.org/content/104/3/955.abstract.

M. Solar, V. Parada, and R. Urrutia. A parallel genetic algorithm to solve the set-covering problem. Computers & Operations Research, 29(9):1221–1235, 2002.

A. C. Tan and D. Gilbert. An empirical comparison of supervised machine learning techniques in bioinformatics. In Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics 2003-Volume 19, pages 219–222. Australian Computer Society, Inc., 2003.

D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis. Unsupervised clustering in mRNA expression profiles. Computers in Biology and Medicine, 36(10):1126–1142, 2006.

J. G. Thomas, J. M. Olson, S. J. Tapscott, and L. P. Zhao. An efficient and robust statistical modeling approach to discover differentially expressed genes using genomic expression profiles. Genome Research, 11(7):1227, 2001.

S. Umetani and M. Yagiura. Relaxation heuristics for the set covering problem. Journal of the Operations Research Society of Japan, 50(4):350–375, 2007.

T. Umpai and S. Aitken. Feature selection and classification for microarray data analysis: Evolutionary methods for identifying predictive genes. BMC Bioinformatics, 6(1):148, 2005. ISSN 1471-2105. doi: 10.1186/1471-2105-6-148.

M. Van De Vijver, Y. He, L. Van't Veer, H. Dai, A. Hart, D. Voskuil, G. Schreiber, J. Peterse, C. Roberts, M. Marton, et al. A gene-expression signature as a predictor of survival in breast cancer. New England Journal of Medicine, 347(25):1999–2009, 2002.

L. Wang, A. Ngom, and R. Gras. Non-Unique Oligonucleotide Microarray Probe Selection Method Based on Genetic Algorithms, June 2008.

Y. Wang, I. V. Tetko, M. A. Hall, E. Frank, A. Facius, K. F. X. Mayer, and H. W. Mewes. Gene selection from microarray data for cancer classification—a machine learning approach. Computational Biology and Chemistry, 29(1):37–46, 2005.

I. H. Witten and E. Frank. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2nd edition, 2005.

L. A. Wolsey. Integer programming. Wiley New York, 1998.

E. P. Xing, M. I. Jordan, and R. M. Karp. Feature selection for high-dimensional genomic microarray data. In MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-, pages 601–608. Citeseer, 2001.

M. Xiong, X. Fang, and J. Zhao. Biomarker identification by feature wrappers. Genome Research, 11(11):1878, 2001.

Y. Yamanishi, J. P. Vert, and M. Kanehisa. Protein network inference from multiple genomic data: a supervised approach. Bioinformatics, 20(Suppl 1):i363, 2004.

E. J. Yeoh, M. E. Ross, S. A. Shurtleff, W. K. Williams, D. Patel, R. Mahfouz, F. G. Behm, S. C. Raimondi, M. V. Relling, A. Patel, and Others. Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. Cancer cell, 1(2):133–143, 2002.

K. Y. Yeung and R. E. Bumgarner. Multiclass classification of microarray data with repeated measurements: application to cancer. Genome Biology, 4(12):83, 2003.

I. n. I. Yvan Saeys and P. Larrañaga. A review of feature selection techniques in bioinformatics. Bioinformatics, 23(19):2507–2517, 2007.