# SET COVERING PROBLEM: A COMPUTATIONAL STUDY OF DIFFERENT APPROACHES WITH CPLEX

SUPERVISOR: PROF. DOMENICO SALVAGNIN

CANDIDATE: MASSIMO MENEGHELLO

ACADEMIC YEAR: 2018-2019


UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
SECOND CYCLE DEGREE IN COMPUTER ENGINEERING

# SET COVERING PROBLEM: A COMPUTATIONAL STUDY OF DIFFERENT APPROACHES WITH CPLEX

SUPERVISOR: PROF. DOMENICO SALVAGNIN

CANDIDATE: MASSIMO MENEGHELLO

Padua, December 16th, 2019

Academic Year: 2018-2019

# Table of Contents

**Abstract** **1**

**1 Introduction** **3**

1.1 Instances Description . . . . . 5

1.2 Concerning the Tests . . . . . 5

**2 Preprocessing** **7**

2.1 Dominance Preprocessing . . . . . 7

2.2 Preprocessing Results . . . . . 8

**3 The Balas' Framework** **15**

3.1 The Cut Generation Procedure . . . . . 15

3.2 The Branch Generation Procedure . . . . . 17

3.3 Implementation Notes . . . . . 18

**4 Computational Results** **25**

**5 Conclusions** **37**

**A Helpful Resources** **39**

A.1 Software Description . . . . . 39

A.2 CPLEX Set Up . . . . . 40

**Abstract**

Il presente lavoro tratta dell'implementazione di alcuni metodi per la pre-elaborazione e la risoluzione esatta del *Set Covering Problem* (SCP) con tecniche di *Mixed-Integer Linear Programming* (MILP).

Viene inizialmente fornita la descrizione formale del SCP (Capitolo 1). A questa si aggiungono alcuni esempi di applicazioni del SCP e la descrizione delle istanze che verranno utilizzate per provare le tecniche descritte in seguito.

Il Capitolo 2 espone un metodo originale per la pre-elaborazione di istanze del SCP. Nello stesso, vengono anche riportati i risultati sperimentali ottenuti con tale metodo i quali sono messi a confronto con i risultati ottenuti dalle procedure di pre-elaborazione di CPLEX.

Nel Capitolo 3 vengono descritti i metodi utilizzati da E. Balas e A. Ho [1] [2] per risolvere il SCP. Questi vengono quindi rielaborati e adattati per operare con un moderno risolutore MILP. Vengono infine riportati (Capitolo 4) i risultati sperimentali ottenuti con i metodi descritti nel precedente capitolo.

# Chapter 1

# Introduction

The *Set Covering Problem* (SCP) is a classical problem in mathematical optimization and it is one of the 21 Karp's problems shown to be NP-Complete in the *Reducibility Among Combinatorial Problems* [15].

Intuitively, the problem can be expressed by considering a ground set $U$, containing $m$ items, and a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets of $U$. The problem is solved by providing a minimal subcollection $\hat{\mathcal{S}}$ of $\mathcal{S}$ such that

$$U = \bigcup_{S_i \in \hat{\mathcal{S}}} S_i \tag{1.1}$$

Another way to represent the SCP uses an intuition from graph theory. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, in which each edge $(u, v)$ belongs to the graph if and only if $(u \in V_1 \wedge v \in V_2) \vee (u \in V_2 \wedge v \in V_1)$. The vertices in $V_2$ represents the elements of the universe set $\mathcal{U}$ while an edge $(u, v)$ indicates that the element $v$ is included in subset $u$.

Defining the $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$ sets, the SCP can be formulated as an integer programming model:

$$\min \mathbf{e} \cdot \mathbf{x} \tag{1.2}$$

$$A\,\mathbf{x} \geq \mathbf{f} \tag{1.3}$$

$$x_j \in \{0, 1\}\ \forall j \in N \tag{1.4}$$

where $A$ is a $m \times n$ $0-1$ matrix, $\mathbf{e} = [1, \ldots, 1]$ is a row vector with $n$ components and $\mathbf{f} = [1, \ldots, 1]$ is a column vector with $m$ components.

However, the variant of the problem used throughout the present work is a common extension of the SCP and provides a costs vector $\mathbf{c}$, which assign a cost to each column (or subset). In this way, equation 1.2 is substituted with

$$\min \mathbf{c} \cdot \mathbf{x} \tag{1.5}$$

Let $A^i$ and $A_j$ denote the $i$-th row and the $j$-th column of $A$, respectively. It is beneficial to define

$$M_j = \{i \in M \mid a_{ij} = 1\}, j \in N \tag{1.6}$$

$$N_i = \{j \in N \mid a_{ij} = 1\}, i \in M \tag{1.7}$$

We say that a vector $\mathbf{x} \in \{0,1\}^n$ is a *cover* if it satisfies all the constraints imposed by $A\mathbf{x} \geq \mathbf{f}$. The set

$$S(\mathbf{x}) = \{j \in N \mid x_j = 1\} \tag{1.8}$$

is called the *support* of vector $\mathbf{x}$.

A solution that includes all the columns of the problem is always admissible (otherwise, the problem would be infeasible). This solution can be used to get the first upper bound for a given instance.

Usually, we are interested in solutions not including redundant columns, and we call these *prime covers*. In these solutions, none of the columns can be removed and this means that each of them covers at least one row not covered by others. For a given cover $\mathbf{x}$ it is useful to define the set

$$T(\mathbf{x}) = \left\{i \in M \mid a^i = 1\right\} \tag{1.9}$$

containing all the row indices of $A$ covered by only one column of $\mathbf{x}$.

The SCP has many important applications in several domains. As Bellmore and Ratliff reported in [7], in the past the SCP was used in a wide range of contexts like network attack and defense [5] [6], information retrieval [11] and truck dispatching [10].

However, one of the most relevant utilisation of the SCP is the *crew scheduling* for railway and flight companies.

For example, considering a set of $m$ trips and a record of valid *pairings* (a *pairing* is a sequence of trips that can be achieved by a single crew), one can use the SCP to compute the minimum number of crews required to comprise all the trips.

The general approach consists of providing the starting set of trips and a process to generate a huge number of possible pairings. As a result, the models for these applications can be extremely large, counting more than 1 million columns and several thousands of rows [13].

An optimal approach for such instances is impractical, therefore, sophisticated heuristic approaches are used instead. Classical greedy algorithms for the SCP were proposed since the '80 [2] and although they are certainly fast in practice, they actually produce low-quality solutions.

Therefore, the most effective strategies to solve large SPC instances are based on *Lagrangian relaxation* and *subgradient optimization*, as reported in [8].

In more recent years, the SCP found new terrain in rapidly growing fields such as *machine learning* and *data mining*.

For example, in *data quality*, one can be interested in retrieving a minimal set of rules (without redundancy) that provides such data. Similarly, in data mining and machine learning, data are usually composed of a large number of binary features or attributes and we are interested in finding a small fraction of items that cover all these features [9].

In the era of the *Big Data*, it is clear as the procedures to solve for such problems must be reliable but essentially they must be fast. These are the main motivations why nowadays most SPC related researches deal with heuristic approaches while works that deal with exact strategies that are somewhat rare.

The present work is structured as follow.

In Chapter 2, we present a new approach to preprocess SPC instances. This results to be more effective that the preprocessing routines implemented by the CPLEX solver.

In Chapter 3, we describe our strategy to solve the SCP based on the works of E. Balas and A. Ho [1] [2]. The main idea is to apply such concepts on modern MILP solvers.

Chapter 4 deals with the computational results collected from the experiments.

## 1.1 Instances Description

The instances used throughout this work represent a subset of the well-known instances collected by J.E. Beasley in his online repository *OR-Library* [4].

In particular, we used the same instances used by Beasley in his work *A Lagrangian heuristic for set-covering problems* [3]. Table 1.1 collects the main information concerning such instances.

The repository created by J.E. Beasley also contains most of the instances used by E. Balas and A. Ho in [1] and [2].

## 1.2 Concerning the Tests

These are the parameters that must be taken into account when interpreting the data (unless otherwise stated).

All the data presented throughout the present work were collected by mean of a device equipped with 4 Intel Xeon X5670 2.93 GHz (6 cores each) CPUs and 144 GB RAM. The experiments are usually carried out using a single thread and they are repeated 5 times.

The MILP solver used for this work is CPLEX, version 12.8 [14]. In the following, when we mention CPLEX we do not refer the stand-alone solver but the core library instead.

| Instance | Rows | Cols | Non-zero | Density | Costs range |
|---|---|---|---|---|---|
| scpnre1 | 500 | 5000 | 249448 | 0.0998 | 1-100 |
| scpnre2 | 500 | 5000 | 249367 | 0.0997 | 1-100 |
| scpnre3 | 500 | 5000 | 249371 | 0.0997 | 1-100 |
| scpnre4 | 500 | 5000 | 249341 | 0.0997 | 1-100 |
| scpnre5 | 500 | 5000 | 249393 | 0.0998 | 1-100 |
| scpnrf1 | 500 | 5000 | 499314 | 0.1997 | 1-100 |
| scpnrf2 | 500 | 5000 | 499275 | 0.1997 | 1-100 |
| scpnrf3 | 500 | 5000 | 499250 | 0.1997 | 1-100 |
| scpnrf4 | 500 | 5000 | 499302 | 0.1997 | 1-100 |
| scpnrf5 | 500 | 5000 | 499336 | 0.1997 | 1-100 |
| scpnrg1 | 1000 | 10000 | 199471 | 0.0199 | 1-100 |
| scpnrg2 | 1000 | 10000 | 199451 | 0.0199 | 1-100 |
| scpnrg3 | 1000 | 10000 | 199498 | 0.0199 | 1-100 |
| scpnrg4 | 1000 | 10000 | 199456 | 0.0199 | 1-100 |
| scpnrg5 | 1000 | 10000 | 199450 | 0.0199 | 1-100 |
| scpnrh1 | 1000 | 10000 | 499163 | 0.0499 | 1-100 |
| scpnrh2 | 1000 | 10000 | 499167 | 0.0499 | 1-100 |
| scpnrh3 | 1000 | 10000 | 499126 | 0.0499 | 1-100 |
| scpnrh4 | 1000 | 10000 | 499149 | 0.0499 | 1-100 |
| scpnrh5 | 1000 | 10000 | 499179 | 0.0499 | 1-100 |

Table 1.1: Description of the instances used in this work.

# Chapter 2

# Preprocessing

As can be noted, in the instances we presented in the previous chapter, the number of the columns is 10 times the number of the rows. Despite these are artificial problems, in real-world configurations, this proportion can be even higher.

One can ask if all these columns are evenly useful or, in the same way, if they are all likely to appear in a solution. CPLEX can answer this question. Once the preprocessing phase completes, CPLEX will try to solve a different model, with the same number of rows but many fewer columns.

## 2.1 Dominance Preprocessing

The first step we implemented in solving our SCP instances is a preprocessing routine we denominated *Dominance Finder* (or simply *Dominance*). The approach is quite elegant and it occurred to be more effective than the preprocessing routines applied by CPLEX.

The following three points illustrate the central aspects of this method, leaving the formal description to Algorithm 1.

1. At the start, we keep all the columns of an instance with unitary cost, removing the other ones. Let $\tilde{N} = \{j \in N \mid c_j = 1\}$, we can define $\tilde{A} = [A_{j_1}, \ldots, A_{j_k}]$ and $\tilde{\mathbf{c}}^{\mathbf{T}} = [c_{j_1}, \ldots, c_{j_k}]$ with $j_i \in \tilde{N}$, which are, respectively, the matrix and the relative cost vector obtained in this way. We should also consider the set $\hat{N} = N \setminus \tilde{N}$.

2. The procedure iterates until the set $\hat{N}$ is empty. At each step, we extract the column $\hat{\jmath} \in \hat{N}$ with the minimum cost, updating $\hat{N} \leftarrow \hat{N} \setminus \hat{\jmath}$.

Then we try to solve the problem defined as

$$\min \tilde{\mathbf{c}}^{\mathbf{T}} \cdot \mathbf{x} \tag{2.1}$$

$$\tilde{A}\mathbf{x} \geq A_{\hat{\jmath}} \tag{2.2}$$

$$x_j \in \{0, 1\} \ \forall j \in \tilde{N} \tag{2.3}$$

3. Clearly, there exist two possible results regarding the previous routine: the problem is infeasible or it has a solution. However, in the latter case, we should consider the cost (or the *objective value*) of the solution such obtained. In fact, if this solution is strictly greater than the cost of the $A_{\hat{\jmath}}$ column, we must keep the column (we update $\tilde{N} \leftarrow \tilde{N} \cup \{\hat{\jmath}\}$). We act in the same way also when the problem is infeasible. Otherwise, we are allowed to remove the column from the original model.

The meaning of this preprocessing routine is the following. Considering the subproblem defined by the columns in $\tilde{N}$ and column $\hat{\jmath}$, if we can find a solution $S \subseteq \tilde{N}$ for this problem and the objective value is lower or equal than the cost $c_{\hat{\jmath}}$, this means we can *substitute* column $\hat{\jmath}$ with the columns in $S$. With the term *substitute*, here we indicate that the columns in $S$ cover all the rows covered by $A_j$ (and possibly more) for the same costs (or less). Consequently, column $\hat{\jmath}$ is unlikely to appear in a solution of the starting problem, because each row in the set $M_{\hat{\jmath}}$ can be covered by $S$. In addition, we can demonstrate that removing $\hat{\jmath}$ does not change the model and that is a valid operation.

However, it looks clear how this procedure works well only with a wide costs range. On the other hand, it does not affect instances with unitary costs, as for example, the Seymour problem [12].

From a practical perspective, we can notice that at line 8 of Algorithm 1 we have to solve a MILP problem. Therefore, if we have $n$ columns the procedure must solve approximately $n$ of such problems. This step is a really time-consuming one, making the entire procedure.

Nevertheless, as we said at the beginning of this chapter, CPLEX can remove a high number of columns in a very short amount of time by using its strong and performant preprocessing routines. Once we obtain the reduced model by CPLEX, we can provide it to the *Dominance* algorithm.

## 2.2 Preprocessing Results

As can be observed from Table 2.1, the time required to execute the *Dominance* routine on the original instances is significant. In particular, for the smaller instances $e$ and $f$ the time is even higher than the time needed by CPLEX to find the optimal solution.

This could be sufficient to demonstrate that the procedure (without the CPLEX preprocessing phase) is not suitable for practical applications. Furthermore, several variables cannot be removed by only applying the dominance procedure. This phenomenon appears more evident in the $e$ and $f$ instances.

On the other side, the CPLEX reduced $g$ and $h$ instances present a large number of variables that the *Dominance* procedure is still able to remove. This number varies from 256 ($g2$) to 1174 ($g3$) for the $g$ instances and from 103 ($h5$) to 366 ($h2$) for the $h$ instances. The discrimination between $g$ and $h$ instances became necessary in this case due to the visible difference in the two ranges.

Furthermore, the average number of columns removed by the *Dominance* procedure in the $g$ instances is 749 while for the $h$ instances this number is 253. Considering the average number of columns in the reduced instances (1561 for $g$ and 1506 for $h$) it can be noted how the difference is not significant. Moreover, these values are quite comparable for the instances not processed by CPLEX (1572 for $g$ and 1511 for $h$).

Based on this evidence, we might ask whether the preprocessing routines adopted by CPLEX are less effective in less dense instances.

We can now consider Table 2.2. The first thing one can observe is the drastic reduction in the cost ranges. We can also provide a simple rule of thumb: the greater the density, the smaller the cost range.

Besides, as we can expect, the density of the matrices is increased in all the $g$ and $h$ instances. This is a consequence of the preprocessing phase: a column with less non-zero elements has more chances to be removed from the model.

As the last analysis for this approach, we have tried to solve the instances that are most affected by the preprocessing phase. Table 2.3 and Table 2.4 report the results obtained with the instances $g$ and $h$: in Table 2.3 CPLEX applied its preprocessing routines to the original instances while in Table 2.4 the reduced instances described in Table 2.2 were supplied in the input. The duration of the experiments was set at 5 hours, a time deemed appropriate to avoid data noise.

From the results, we note that the reduced instance $g2$ is resolved on average faster than the instance to which the Dominance has not been applied. The same can similarly be said for the instance $g1$.

However, by comparing the values of the MIP gap, there are no significant discrepancies between Table 2.3 and Table 2.4.

**Input :** a *set covering* problem composed by a matrix $A$ and a costs vector $\mathbf{c}$.

**Output:** $\tilde{N}$, a set containing the indeces of the columns that must be included in the reduced problem.

$\tilde{N} \leftarrow \{j \in N \mid c_j = 1\}$
$\hat{N} \leftarrow N \setminus \tilde{N}$
$\tilde{A} \leftarrow [A_{j_1}, \ldots, A_{j_k}], \forall j_i \in \tilde{N}$
$\tilde{\mathbf{c}}^{\mathbf{T}} \leftarrow [c_{j_1}, \ldots, c_{j_k}], \forall j_i \in \tilde{N}$
**While** $\hat{N} \neq \emptyset$ **do**
    $\hat{\jmath} \leftarrow \text{argmin}_{j \in \hat{N}} \{c_j\}$
    $\hat{N} \leftarrow \hat{N} \setminus \{\hat{\jmath}\}$
    *solve problem* $\min \left\{ \tilde{\mathbf{c}}^{\mathbf{T}}\mathbf{x} \mid \tilde{A}\mathbf{x} \geq A_{\hat{\jmath}}, x_j \in \{0,1\} \right\}$
    **If not** (*previous problem has a solution* $\bar{\mathbf{x}}$ **and** $\tilde{\mathbf{c}}^{\mathbf{T}}\bar{\mathbf{x}} \leq c_{\hat{\jmath}}$) **then**
        **Comment:** $\hat{\jmath}$ *is must be included in the reduced model*
        $\tilde{N} \leftarrow \tilde{N} \cup \{\hat{\jmath}\}$
**Return** $\tilde{N}$

**Algorithm 1:** The *Dominance Finder* pseudocode.

| Instance | Cols | (1) | Time (s) | (2) | Time (s) | (3) | Time (s) |
|---|---|---|---|---|---|---|---|
| scpnre1 | 5000 | 395 | 3.24 | 477 | 650.44 | 395 | 11.53 |
| scpnre2 | 5000 | 490 | 3.72 | 563 | 710.66 | 486 | 22.25 |
| scpnre3 | 5000 | 352 | 2.77 | 521 | 690.02 | 352 | 12.30 |
| scpnre4 | 5000 | 406 | 3.59 | 518 | 681.69 | 406 | 13.82 |
| scpnre5 | 5000 | 386 | 3.51 | 554 | 659.86 | 386 | 11.36 |
| scpnrf1 | 5000 | 277 | 4.17 | 386 | 2491.36 | 277 | 71.09 |
| scpnrf2 | 5000 | 254 | 4.32 | 375 | 2036.07 | 254 | 51.65 |
| scpnrf3 | 5000 | 295 | 4.27 | 393 | 2209.60 | 294 | 66.87 |
| scpnrf4 | 5000 | 276 | 3.94 | 356 | 2645.95 | 276 | 72.56 |
| scpnrf5 | 5000 | 308 | 3.36 | 350 | 3332.33 | 308 | 117.65 |
| scpnrg1 | 10000 | 2157 | 8.59 | 1641 | 1190.90 | 1633 | 43.00 |
| scpnrg2 | 10000 | 1752 | 7.09 | 1543 | 1343.70 | 1496 | 30.47 |
| scpnrg3 | 10000 | 2727 | 6.60 | 1553 | 1144.53 | 1553 | 56.26 |
| scpnrg4 | 10000 | 2399 | 7.77 | 1555 | 1316.04 | 1555 | 49.61 |
| scpnrg5 | 10000 | 2519 | 7.09 | 1569 | 1142.68 | 1569 | 52.57 |
| scpnrh1 | 10000 | 1922 | 9.85 | 1602 | 2542.16 | 1601 | 99.28 |
| scpnrh2 | 10000 | 1882 | 9.23 | 1517 | 2576.38 | 1516 | 85.27 |
| scpnrh3 | 10000 | 1772 | 9.18 | 1508 | 2740.46 | 1506 | 88.90 |
| scpnrh4 | 10000 | 1676 | 7.27 | 1469 | 2488.34 | 1465 | 82.32 |
| scpnrh5 | 10000 | 1543 | 8.29 | 1458 | 2498.75 | 1440 | 74.35 |

Table 2.1: (1) indicates the number of columns obtained after the CPLEX preprocessing routines. (2) reports the number of columns in the model after applying the Dominance procedure on the original instances. (3) indicates the number of left columns after executing the Dominance procedure on the instances obtained with (1). Aside are listed the average execution times of such methods (average on 5 runs with different seeds).

| Instance | Rows | Cols | Non-zero | Density | Costs range |
|---|---|---|---|---|---|
| scpnre1_red | 500 | 395 | 19777 | 0.1001 | 1-10 |
| scpnre2_red | 500 | 486 | 24185 | 0.0995 | 1-11 |
| scpnre3_red | 500 | 352 | 17466 | 0.0992 | 1-9 |
| scpnre4_red | 500 | 406 | 20098 | 0.099 | 1-10 |
| scpnre5_red | 500 | 386 | 19107 | 0.099 | 1-9 |
| scpnrf1_red | 500 | 277 | 27802 | 0.2007 | 1-6 |
| scpnrf2_red | 500 | 254 | 25408 | 0.2001 | 1-7 |
| scpnrf3_red | 500 | 294 | 29264 | 0.1991 | 1-8 |
| scpnrf4_red | 500 | 276 | 27462 | 0.199 | 1-7 |
| scpnrf5_red | 500 | 308 | 30789 | 0.1999 | 1-7 |
| scpnrg1_red | 1000 | 1633 | 33995 | 0.0208 | 1-28 |
| scpnrg2_red | 1000 | 1496 | 30873 | 0.0206 | 1-21 |
| scpnrg3_red | 1000 | 1553 | 32315 | 0.0208 | 1-26 |
| scpnrg4_red | 1000 | 1555 | 32415 | 0.0208 | 1-27 |
| scpnrg5_red | 1000 | 1569 | 32845 | 0.0209 | 1-27 |
| scpnrh1_red | 1000 | 1601 | 81071 | 0.0506 | 1-20 |
| scpnrh2_red | 1000 | 1516 | 76797 | 0.0507 | 1-20 |
| scpnrh3_red | 1000 | 1506 | 76210 | 0.0506 | 1-19 |
| scpnrh4_red | 1000 | 1465 | 74124 | 0.0506 | 1-19 |
| scpnrh5_red | 1000 | 1440 | 72761 | 0.0505 | 1-17 |

Table 2.2: Description of the reduced instances used in the present work.

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnrg1 | 18000.00 | 176.0 | 172.97 | 1.72 | 991658.6 | 425638.2 |
| scpnrg2 | 3147.41 | 154.0 | 154.00 | 0.00 | 199706.6 | 0.0 |
| scpnrg3 | 18000.00 | 166.6 | 161.82 | 2.87 | 1260491.6 | 739138.0 |
| scpnrg4 | 18000.00 | 168.8 | 164.10 | 2.78 | 1181769.4 | 639052.6 |
| scpnrg5 | 18000.00 | 168.0 | 162.78 | 3.11 | 1090322.8 | 688329.6 |
| scpnrh1 | 18000.00 | 64.4 | 55.65 | 13.58 | 1246697.2 | 1088933.0 |
| scpnrh2 | 18000.00 | 63.8 | 56.26 | 11.81 | 1326751.0 | 1124598.6 |
| scpnrh3 | 18000.00 | 60.2 | 52.59 | 12.64 | 1345568.6 | 1150378.0 |
| scpnrh4 | 18000.00 | 58.4 | 51.23 | 12.27 | 1331430.0 | 1121670.4 |
| scpnrh5 | 18000.00 | 55.0 | 49.98 | 9.12 | 1360743.0 | 1021025.2 |

Table 2.3: Results for the CPLEX solver on original $g$ and $h$ instances with 18000 seconds time limit (average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnrg1_red | 18000.00 | 176.00 | 174.24 | 1.00 | 1188044.2 | 205718.6 |
| scpnrg2_red | 2716.78 | 154.00 | 154.00 | 0.00 | 211023.8 | 0.0 |
| scpnrg3_red | 18000.00 | 166.80 | 162.00 | 2.88 | 1348762.4 | 829312.4 |
| scpnrg4_red | 18000.00 | 168.20 | 164.28 | 2.33 | 1251243.0 | 661292.0 |
| scpnrg5_red | 18000.00 | 168.00 | 162.96 | 3.00 | 1198035.0 | 734644.6 |
| scpnrh1_red | 18000.00 | 64.00 | 55.60 | 13.12 | 1154111.6 | 995073.4 |
| scpnrh2_red | 18000.00 | 63.80 | 56.31 | 11.73 | 1386090.8 | 1172214.8 |
| scpnrh3_red | 18000.00 | 60.00 | 52.61 | 12.30 | 1347406.0 | 1125535.0 |
| scpnrh4_red | 18000.00 | 58.80 | 51.24 | 12.85 | 1381447.0 | 1158221.8 |
| scpnrh5_red | 18000.00 | 55.00 | 50.01 | 9.07 | 1328002.8 | 995723.6 |

Table 2.4: Results for the CPLEX solver on reduced $g$ and $h$ instances with 18000 seconds time limit (average on 5 runs).

# Chapter 3

# The Balas' Framework

The central concept in the works of E. Balas is the *conditional bound* [1]. We considered this idea for the approach discussed in the present dissertation.

A conditional bound is a suitable restriction for the original feasible set derived from an available solution. It is possible to use the conditional bound to iteratively produce cuts, obtaining a *cutting-planes* procedure, or to generate disjunctions in a different (and stronger) way than the usual dichotomy on a single variable. Using both the previous strategies, one can implement a *Branch-and-Cut* algorithm. The procedure is suitably explained in [2].

In the present work, we attempt to accommodate those techniques for a modern MILP solver like CPLEX.

First, we introduce the E. Balas procedure to generate cuts derived from conditional bounds. In a second moment, we present the entire method to produce nodes in a Branch-and-Cut environment and some notes regarding the implementation.

## 3.1 The Cut Generation Procedure

To obtain a conditional bound for the SCP, the fundamental problem in [1] (Theorem 2) consists in finding a set of column indices $S = \{j(1), ..., j(p)\}$ with $\emptyset \neq S \subseteq N$ such that

$$\sum_{j \in S} s_j \geq z_u - \mathbf{u} \cdot \mathbf{e} \tag{3.1}$$

where $z_u$ is the current upper bound (the cost of the best solution found so far). Hence, for any set of $p$ rows indeces $h(i) \in M$, with $i = 1, \ldots, p$ and any collection of $p$ subsets $Q_i \subseteq N$ with $i = 1, \ldots, p$ satisfying

$$\sum_{i|j \in Q_i} s_{j(i)} \leq s_j, \quad j \in N \tag{3.2}$$

every cover $\mathbf{x}$ such that $\mathbf{c} \cdot \mathbf{x} < z_u$ satisfies the inequality

$$\sum_{j \in W} x_j \geq 1 \tag{3.3}$$

where

$$W = \bigcup_{i=1}^{p} \left( N_{h(i)} \setminus Q_i \right) \tag{3.4}$$

From this central theorem, E. Balas developed the main procedure to produce his cutting-plane algorithm, here reported as Algorithm 2.

A critical concept in Algorithm 2 regards the avoidance of duplicated cuts. In fact, the way E. Balas conceived his procedure does not allow that a cut to appears twice. This prevents the algorithm to be trapped in a cycle.

The description made by E. Balas of Algorithm 2 requires a deep understanding of his work. In the following, we report some relevant aspects that should make it clearer.

In order to operate, the procedure needs a feasible primal solution $\mathbf{x}$, a feasible dual solution $\mathbf{u}$ and the reduced costs vector $\mathbf{s}$ associated to $\mathbf{u}$. Both the primal and the dual solution can be obtained running appropriate heuristic algorithms. Alternatively, one can directly solve the dual problem

$$\max \mathbf{u} \cdot \mathbf{f} \tag{3.5}$$

$$\mathbf{u}\, A \leq \mathbf{c} \tag{3.6}$$

$$u_i \geq 0 \; \forall i \in M \tag{3.7}$$

to obtain $\mathbf{u}$. A smarter way to get the $\mathbf{u}$ vector is to apply the *dual simplex* on the primal problem (1.3) - (1.5). This is exactly what CPLEX does and we can take advantage of this by directly requiring the $\mathbf{u}$ vector to CPLEX.

Once gain those vectors, the reduced costs vector $\mathbf{s}$ is provided by

$$\mathbf{s} = \mathbf{c} - \mathbf{u}\, A \tag{3.8}$$

The initialization phase of the algorithm demands to define several sets. The $S$ set is the subset of the support of vector $\mathbf{x}$, as defined in (1.8), containing only the columns with a strictly positive reduced cost. The $T(\mathbf{x})$ set includes the rows that are minimally covered by the current solution $\mathbf{x}$ (there is only 1 column in $\mathbf{x}$ that cover each row in $T(\mathbf{x})$). The $W$ set is empty at the beginning and this will be the support of the new generated cut at the end of Algorithm 2.

The goal of the algorithm is to produce a strong cut, this means to produce a new constraint, in the SCP form (3.3) containing as less non-zero elements as possible. In the procedure, this is equivalent to minimize the size of the $W$ set. This process is performed at line 11 of Algorithm 2.

Another relevant point to notice is that none of the columns in the current solution $\mathbf{x}$ appears in set $W$. During iteration $t$, the procedure examines column $j(t) \in J$ (a subset of $S$). At line 13, it subtracts set $Q$ (whom $J$ is a subset) from $W$. In this way, none of the columns in $S$ appears in $W$.

This is the most important part of the cycle avoidance mechanism: once the new cut is appended to the model, the primal heuristics have to find a new solution valid for the new extended model. This solution is necessarily different from the previous ones.

Algorithm 2 can be iteratively applied to a SCP instance until the condition

$$z_l = \mathbf{u} \cdot \mathbf{f} \geq \mathbf{c} \cdot \mathbf{x} = z_u \tag{3.9}$$

is met. However, this approach is not effective in practice due to the huge number of cuts required to complete.

## 3.2 The Branch Generation Procedure

The procedure for producing new branches for the Branch and Bound routine differs a bit from the cut-generation one. Both in [1] and [2] E. Balas and A. Ho do not provide an explicit description of the algorithm but they suggests that the procedure does not differ much from Algorithm 2.

Therefore, in the following, we will explain our interpretation of this procedure. The pseudo-code is reported in Algorithm 3.

The main difference from Algorithm 2 is that we no longer need the $W$ set to store the indices of the columns. Instead, we want to return a collection $\mathcal{R}$ containing the sets of the variables required to generate each branch.

Once we obtain the collection $\mathcal{R}$, the rule developed by E. Balas and A. Ho in [2] is the following

$$\bigvee_{i=1}^{p} \left( x_j = 0,\, j \in \mathcal{R}[i] \mid \sum_{j \in \mathcal{R}[k]} x_j \geq 1,\, k = 1, \ldots, i-1 \right) \tag{3.10}$$

where $\mathcal{R}[i]$ is the $i$-th set in collection $\mathcal{R}$. We reported (3.10) as pseudo-code in Algorithm 4.

Due to the nonhomogeneous performances of the current branch rule, E. Balas also developed a method to decide in which cases to apply his rule. This method can be derived from the following 3 points:

1.

$$\sum_{i=1}^{p} |\mathcal{R}[i]| > p \log_2 p \tag{3.11}$$

2. there exist at most 1 singleton (a set containing exactly 1 element) among the sets $\mathcal{R}[i]$, $i = 1, \ldots, p$

$$\exists! \, 1 \leq i \leq p \mid |\mathcal{R}[i]| = 1 \tag{3.12}$$

3. $p$ does not exceed a specific constant

$$p \leq M_{branches} \tag{3.13}$$

In the case in which all these conditions are not met, an alternative branch rule is proposed. This rule (BR2) is less complex than the previous one and it is obtained by a dichotomy on a single variable. The pseudo-code of this approach is reported in Algorithm 5.

As reported by the author, the cardinality of the set $N_k \cap N_i$ usually equals to 1. This means that most of the times the method branches on a single variable. In this way, the BR2 does not differ from other branching rules that select a single variable (differently from BR1 and BR2 that can use multiple variables to branch instead). Nevertheless, due to its performances and its simplicity, this is considered by E. Balas and A. Ho the best branching rule to adopt when BR1 fails.

In the present work, we performed experiments substituting BR2 with the branches computed by CPLEX (CPLEX Branch Rule, CBR).

## 3.3 Implementation Notes

The first critical step in the realization of this project was the implementation of Algorithm 2. This procedure required a long testing phase.

The first version of Algorithm 2 was realized with the Python programming language [17]. This approach allowed rapid prototyping and the finding of the main implementation obstacles. Several tests were performed on this version, however, large instances could not have experimented. Besides, the CPLEX interface for Python is limited and it does not support several operations that are permitted only by directly accessing the CPLEX library.

For example, working with the Python interface, inside the body of a branch callback, it is not possible to require the matrix of the model.

Without this necessary component, this first testing phase was limited to a framework not dissimilar to the one described in [2]. Consequently, the heuristic procedures there defined were

also implemented and tested in the same instances reported in that study.

The next step consisted in the translation and the adaptation of the software for the C language and the CPLEX native library.

The algorithms presented in the current chapter were developed either in dense and sparse form. Inside a callback body, the user can request the CPLEX native library to provide the matrix of the current problem. CPLEX store the matrix in a sparse structure and a developer can require this structure to be arranged by rows or by columns.

Since CPLEX does not allow the user to generate more than 2 children for a single node, we developed the following recursive solution to generate as many nodes as we needed.

Once the procedure decides to use the BR1 rule, we generate the first node as in the first iteration of Algorithm 4.

Then we generate the second node and we supply it a structure containing

- an index $i \leftarrow 2$, indicating that we pass the first level and we are ready for the second one,
- the collection $\mathcal{R}$,
- the integer $p$, indicating the number of elements in $\mathcal{R}$.

When our branch callback is newly called, we check if the index $i$ is lower or equal than $p$: in this case, we generate the node corresponding to the $i$-th element of the collection $\mathcal{R}$. Otherwise, the procedure advances to the branch generation step.

**Input :** a primal solution $\mathbf{x}$ and a dual solution $\mathbf{u}$ for the SCP, the reduced costs vector $\mathbf{s}$ associated to $\mathbf{u}$, the upper bound $z_u$ provided by $\mathbf{x}$.

**Output:** a valid cut for the SCP.

$W \leftarrow \emptyset$

$S \leftarrow \{j \in S(\mathbf{x}) \mid s_j > 0\}$

$T(\mathbf{x}) \leftarrow \{i \in M \mid A^i \cdot \mathbf{x} = 1\}$

$y \leftarrow \mathbf{u} \cdot \mathbf{e}$

$t \leftarrow 0$

**While** *True* **do**

  $v_t \leftarrow \min\{\max_{j \in S} s_j,\ \min_{j \in S}\{s_j \mid s_j \geq z_u - y\}\}$

  $J \leftarrow \{j \in S \mid s_j = v_t\}$

  $Q \leftarrow \{j \in N \mid s_j \geq v_t\}$

  $M_J \leftarrow \bigcup_{j \in J} M_j$

  $i(t) \leftarrow \text{argmin}_{i \in T(\mathbf{x}) \cap M_J} \mid N_i \setminus Q \cap W \mid$

  $j(t) \leftarrow J \cap N_{i(t)}$

  $W \leftarrow W \cup \left(N_{i(t)} \setminus Q\right)$

  $y \leftarrow y + s_{j(t)}$

  **If** $y \geq z_u$ **then**

    **Break**

  $S \leftarrow S \setminus \{j(t)\}$

  $s_j \leftarrow s_j - s_{j(t)}\ \forall j \in N_{i(t)} \cap Q$

  $t \leftarrow t + 1$

**Return** $\sum_{j \in W} x_j \geq 1$

**Algorithm 2:** *Balas cut generation* (BCG) procedure.

**Input :** a primal solution $\mathbf{x}$ and a dual solution $\mathbf{u}$ for the SCP, the reduced costs vector $\mathbf{s}$ associated to $\mathbf{u}$, the upper bound $z_u$ provided by $\mathbf{x}$.

**Output:** a collection $\mathcal{R}$ of branching sets.

$S \leftarrow \{j \in S(\mathbf{x}) \mid s_j > 0\}$

$\mathcal{R} \leftarrow \emptyset$

$T(\mathbf{x}) \leftarrow \{i \in M \mid A^i \cdot \mathbf{x} = 1\}$

$y \leftarrow \mathbf{u} \cdot \mathbf{e}$

$t \leftarrow 0$

**While** *True* **do**

  $v_t \leftarrow \min\{\max_{j \in S} s_j,\ \min_{j \in S}\{s_j \mid s_j \geq z_u - y\}\}$

  $J \leftarrow \{j \in S \mid s_j = v_t\}$

  $Q \leftarrow \{j \in N \mid s_j \geq v_t\}$

  $M_J \leftarrow \bigcup_{j \in J} M_j$

  $i(t) \leftarrow \text{argmin}_{i \in T(\mathbf{x}) \cap M_J} \mid N_i \setminus Q \mid$

  $j(t) \leftarrow J \cap N_{i(t)}$

  $y \leftarrow y + s_{j(t)}$

  $\mathcal{R} \leftarrow \mathcal{R} \cup \{Q\}$

  **If** $y \geq z_u$ **then**

    **Break**

  $S \leftarrow S \setminus \{j(t)\}$

  $s_j \leftarrow s_j - s_{j(t)}\ \forall j \in N_{i(t)} \cap Q$

  $t \leftarrow t + 1$

**Return** $\mathcal{R}$

**Algorithm 3:** *Balas Branch Generation* (BBG) procedure.

**Input :** a collection $\mathcal{R}$ of branching sets (as provided by Algorithm 3), a parent subproblem $P$ from a Branch and Bound tree.

**Output:** a Branch and Bound subtree derived from $\mathcal{R}$.

$Nodes \leftarrow \emptyset$
**For** $i \leftarrow 1, \ldots, p$ **do**
    Let $C_i$ be a copy of problem $P$
    In subproblem $C_i$, set variables $x_j = 0$, $j \in \mathcal{R}[i]$
    **If** $i \neq 1$ **then**
        $W \leftarrow \emptyset$
        **For** $k \leftarrow 1, \ldots, i-1$ **do**
            $W \leftarrow W \cup \mathcal{R}[k]$
        Append constraint $\sum_{j \in W} x_j \geq 1$ to $C_i$
    $Nodes \leftarrow Nodes \cup \{C_i\}$
**Return** $Nodes$

**Algorithm 4:** *Balas Branch Rule 1* (BR1) procedure.

**Input :** a parent subproblem $P$ from a Branch and Bound tree.

**Output:** a set Branch and Bound nodes.

$i \leftarrow$ *last element of the ordered set* $M$
$k \leftarrow \text{argmin}_{h \in M \setminus \{i\}} |N_h \cap N_i|$
Let $C_1$ be a copy of problem $P$
In subproblem $C_1$, set variables $x_j = 0$, $j \in N_k \cap N_i$
Let $C_2$ be a copy of problem $P$
Append constraint $\sum_{j \in N_k \cap N_i} x_j \geq 1$ to $C_2$
**Return** $\{C_1, C_2\}$

**Algorithm 5:** *Balas Branch Rule 2* (BR2) procedure.

**Input :** a subproblem $P$ from a Branch and Bound tree ($P.\mathbf{c}$ indicates the costs vector of $P$, $P.A$ refers to the matrix of the subproblem), an integer constant $M_{branches}$ limiting the maximum number of branches, a *default branch rule.*

**Output:** a set containing the subproblems generated from $P$.

$\mathbf{x} \leftarrow$ *a primal solution for* $P$

$\mathbf{u} \leftarrow$ *a dual solution for* $P$

$\mathbf{s} \leftarrow P.\mathbf{c} - \mathbf{u}\, P.A$

$z_u \leftarrow P.\mathbf{c} \cdot \mathbf{x}$

$\mathcal{R} \leftarrow BalasBranchGeneration(P, \mathbf{x}, \mathbf{u}, \mathbf{s}, z_u)$

$p \leftarrow |\mathcal{R}|$

**If** $\sum_{i=1}^{p} |\mathcal{R}[i]| > p \log_2 p$ **and** $\exists!\, 1 \leq i \leq p \mid |\mathcal{R}[i]| = 1$ **and** $p \leq M_{branches}$ **then**

  **Return** *nodes generated with BR1*

**Else**

  **Return** *nodes generated with the default branch rule*

**Algorithm 6:** The complete branch procedure.

# Chapter 4

# Computational Results

In this chapter, we report the data collected by means of the software developed in the present work. The tables show

- the *time* taken to find the optimal solution (or, alternatively, the set time limit),
- the value of the *best integer solution* found (before the time limit expires),
- the value of the *objective value* (or *lower bound*),
- the *Mixed Integer Programming* (MIP) *gap* calculated by means of the two previous values,
- the number of *nodes generated* in total and
- the number of *unprocessed nodes*.

All the previous values are the result of the average on 5 computations obtained with different seeds. For this reason, even values usually represented as integers (such as the total number of nodes) can have decimal values.

In Table 4.1 we reported the results obtained with CPLEX using the default parameters while in Table 4.2 we reported the results of the same solver but disabling all the cuts. The cuts made by CPLEX are also disabled in all the following experiments.

Tables 4.3, 4.4, 4.5 show the data collected applying the BR1 branching rule and by respectively setting the $M_{branches}$ parameter to the values $4, 6, 8$.

Using the BR1 in this context indicates performing the procedure described in Algorithm 3 thus obtaining the collection $\mathcal{R}$. If all the conditions (3.11) - (3.13) are satisfied, then the nodes are generated as described in Algorithm 4. Otherwise, the nodes generated by CPLEX are used.

Similarly, in Tables 4.6, 4.7, 4.8 the values collected using the BR1 are reported as a branching rule but using functions that implement sparse structures.

In Table 4.9 we explore a bit of BR2. The first point one can notice is the extremely great number of nodes required to solve even the smallest instances.

Therefore, it is clear that the branching rule used by CPLEX is a more desirable alternative than BR2.

A more accessible consultation of the collected data is provided by the Tables 4.10 and 4.11.

In Table 4.10 it is possible to compare the times obtained from the described solvers in instances $e$ and $f$ while in Table 4.11 we can examine the MIP gaps of instances $g$ and $h$. The best result (for both time and MIP gap) in each row is displayed in bold.

It is therefore very clear to see how CPLEX is always the best solver. The only exception is the instance $e4$.

The reason for this event is given by the choice of a relatively low value of the parameter $M_{branches} = 4$ and by the dimensions of the considered instance. As a result, the nodes generated with the BR1 branching rule must not have been on sufficient numbers to compromise the CPLEX computation.

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 77.76 | 29.0 | 29.00 | 0.00 | 26629.0 | 0.0 |
| scpnre2_red | 666.22 | 30.0 | 30.00 | 0.00 | 197333.6 | 0.0 |
| scpnre3_red | 73.32 | 27.0 | 27.00 | 0.00 | 21104.4 | 0.0 |
| scpnre4_red | 209.86 | 28.0 | 28.00 | 0.00 | 55721.6 | 0.0 |
| scpnre5_red | 71.36 | 28.0 | 28.00 | 0.00 | 17741.8 | 0.0 |
| scpnrf1_red | 74.90 | 14.0 | 14.00 | 0.00 | 27196.4 | 0.0 |
| scpnrf2_red | 47.04 | 15.0 | 15.00 | 0.00 | 18821.2 | 0.0 |
| scpnrf3_red | 18.83 | 14.0 | 14.00 | 0.00 | 5260.2 | 0.0 |
| scpnrf4_red | 118.06 | 14.0 | 14.00 | 0.00 | 49586.6 | 0.0 |
| scpnrf5_red | 946.24 | 13.0 | 13.00 | 0.00 | 410140.6 | 0.0 |
| scpnrg1_red | 3600.00 | 176.0 | 170.82 | 2.94 | 197675.0 | 121286.8 |
| scpnrg2_red | 2622.58 | 154.0 | 154.00 | 0.00 | 179330.4 | 0.0 |
| scpnrg3_red | 3600.00 | 166.6 | 159.35 | 4.35 | 220745.2 | 138591.8 |
| scpnrg4_red | 3600.00 | 168.8 | 161.35 | 4.41 | 196260.6 | 120852.4 |
| scpnrg5_red | 3600.00 | 168.0 | 159.97 | 4.78 | 207402.6 | 137877.0 |
| scpnrh1_red | 3600.00 | 64.60 | 53.93 | 16.51 | 204673.2 | 168508.2 |
| scpnrh2_red | 3600.00 | 64.80 | 54.80 | 15.43 | 254589.4 | 212189.8 |
| scpnrh3_red | 3600.00 | 60.20 | 51.18 | 14.96 | 247778.8 | 191524.8 |
| scpnrh4_red | 3600.00 | 58.60 | 49.98 | 14.71 | 253733.8 | 199092.6 |
| scpnrh5_red | 3600.00 | 55.20 | 48.71 | 11.76 | 251290.6 | 189100.8 |

Table 4.1: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 70.71 | 29.00 | 29.00 | 0.00 | 24730.4 | 0.0 |
| scpnre2_red | 388.94 | 30.00 | 30.00 | 0.00 | 180884.4 | 0.0 |
| scpnre3_red | 60.30 | 27.00 | 27.00 | 0.00 | 17950.0 | 0.0 |
| scpnre4_red | 127.12 | 28.00 | 28.00 | 0.00 | 40329.2 | 0.0 |
| scpnre5_red | 35.26 | 28.00 | 28.00 | 0.00 | 11785.6 | 0.0 |
| scpnrf1_red | 79.07 | 14.00 | 14.00 | 0.00 | 29603.0 | 0.0 |
| scpnrf2_red | 50.13 | 15.00 | 15.00 | 0.00 | 18529.0 | 0.0 |
| scpnrf3_red | 36.36 | 14.00 | 14.00 | 0.00 | 10982.4 | 0.0 |
| scpnrf4_red | 132.68 | 14.00 | 14.00 | 0.00 | 49942.8 | 0.0 |
| scpnrf5_red | 493.57 | 13.00 | 13.00 | 0.00 | 308797.2 | 0.0 |
| scpnrg1_red | 3600.00 | 176.00 | 171.01 | 2.84 | 193197.8 | 112009.6 |
| scpnrg2_red | 2886.31 | 154.00 | 153.70 | 0.20 | 203194.0 | 5025.6 |
| scpnrg3_red | 3600.00 | 167.00 | 159.41 | 4.54 | 228072.6 | 159140.4 |
| scpnrg4_red | 3600.00 | 170.20 | 161.32 | 5.22 | 207796.4 | 142896.2 |
| scpnrg5_red | 3600.00 | 168.00 | 160.43 | 4.51 | 217112.8 | 142887.6 |
| scpnrh1_red | 3600.00 | 64.00 | 54.16 | 15.37 | 208974.8 | 164743.0 |
| scpnrh2_red | 3600.00 | 64.20 | 54.81 | 14.62 | 241695.4 | 189153.0 |
| scpnrh3_red | 3600.00 | 60.20 | 51.16 | 15.00 | 241911.6 | 186912.4 |
| scpnrh4_red | 3600.00 | 59.00 | 49.89 | 15.42 | 239950.0 | 185396.8 |
| scpnrh5_red | 3600.00 | 55.00 | 48.66 | 11.52 | 236905.8 | 173051.0 |

Table 4.2: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances, with cuts disabled (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 164.79 | 29.00 | 29.00 | 0.00 | 47974.6 | 0.0 |
| scpnre2_red | 798.47 | 30.00 | 30.00 | 0.00 | 278730.6 | 0.0 |
| scpnre3_red | 113.49 | 27.00 | 27.00 | 0.00 | 26559.8 | 0.0 |
| scpnre4_red | 123.24 | 28.00 | 28.00 | 0.00 | 29339.8 | 0.0 |
| scpnre5_red | 44.72 | 28.00 | 28.00 | 0.00 | 10556.8 | 0.0 |
| scpnrf1_red | 154.81 | 14.00 | 14.00 | 0.00 | 60912.4 | 0.0 |
| scpnrf2_red | 113.54 | 15.00 | 15.00 | 0.00 | 45221.6 | 0.0 |
| scpnrf3_red | 150.39 | 14.00 | 14.00 | 0.00 | 59600.0 | 0.0 |
| scpnrf4_red | 344.91 | 14.00 | 14.00 | 0.00 | 143987.2 | 0.0 |
| scpnrf5_red | 2368.40 | 13.00 | 13.00 | 0.00 | 1550872.4 | 0.0 |
| scpnrg1_red | 3600.01 | 176.00 | 169.97 | 3.42 | 91335.0 | 60209.6 |
| scpnrg2_red | 3600.00 | 155.00 | 151.01 | 2.58 | 112815.0 | 69566.0 |
| scpnrg3_red | 3600.02 | 167.00 | 156.99 | 6.00 | 85395.4 | 69928.4 |
| scpnrg4_red | 3600.01 | 170.60 | 158.96 | 6.82 | 73034.4 | 59184.4 |
| scpnrg5_red | 3600.01 | 168.80 | 158.35 | 6.19 | 86656.0 | 68098.2 |
| scpnrh1_red | 3600.01 | 65.00 | 53.33 | 17.96 | 78488.6 | 69554.0 |
| scpnrh2_red | 3600.02 | 64.00 | 54.10 | 15.47 | 89514.4 | 74078.0 |
| scpnrh3_red | 3600.01 | 60.60 | 50.41 | 16.79 | 92861.0 | 79194.4 |
| scpnrh4_red | 3600.01 | 58.40 | 49.14 | 15.85 | 101327.0 | 86350.0 |
| scpnrh5_red | 3600.01 | 56.20 | 47.88 | 14.79 | 102984.0 | 86475.2 |

Table 4.3: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 4$ (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 190.69 | 29.00 | 29.00 | 0.00 | 86729.2 | 0.0 |
| scpnre2_red | 1960.90 | 30.00 | 30.00 | 0.00 | 1083208.0 | 0.0 |
| scpnre3_red | 90.38 | 27.00 | 27.00 | 0.00 | 22023.6 | 0.0 |
| scpnre4_red | 134.54 | 28.00 | 28.00 | 0.00 | 41029.2 | 0.0 |
| scpnre5_red | 68.74 | 28.00 | 28.00 | 0.00 | 27748.2 | 0.0 |
| scpnrf1_red | 245.22 | 14.00 | 14.00 | 0.00 | 168383.4 | 0.0 |
| scpnrf2_red | 106.39 | 15.00 | 15.00 | 0.00 | 65470.0 | 0.0 |
| scpnrf3_red | 101.24 | 14.00 | 14.00 | 0.00 | 43370.2 | 0.0 |
| scpnrf4_red | 678.97 | 14.00 | 14.00 | 0.00 | 488393.6 | 0.0 |
| scpnrf5_red | 3073.02 | 13.80 | 12.48 | 9.45 | 2566319.2 | 359237.2 |
| scpnrg1_red | 3600.01 | 176.00 | 170.41 | 3.18 | 118756.6 | 75902.8 |
| scpnrg2_red | 3600.01 | 154.80 | 151.21 | 2.32 | 154678.6 | 83600.2 |
| scpnrg3_red | 3600.00 | 167.00 | 157.38 | 5.76 | 112174.0 | 92202.0 |
| scpnrg4_red | 3600.00 | 168.00 | 159.3 | 5.18 | 93332.0 | 53506.0 |
| scpnrg5_red | 3600.01 | 168.40 | 158.45 | 5.91 | 96405.4 | 71390.4 |
| scpnrh1_red | 3600.00 | 65.00 | 53.36 | 17.91 | 81866.8 | 71481.4 |
| scpnrh2_red | 3600.01 | 64.60 | 54.06 | 16.31 | 90747.6 | 77679.2 |
| scpnrh3_red | 3600.00 | 60.80 | 50.38 | 17.13 | 90341.8 | 76717.6 |
| scpnrh4_red | 3600.01 | 58.80 | 49.12 | 16.46 | 103517.0 | 90105.6 |
| scpnrh5_red | 3600.01 | 56.00 | 47.97 | 14.32 | 110160.2 | 92256.8 |

Table 4.4: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 6$ (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 509.01 | 29.00 | 29.00 | 0.00 | 259270.0 | 0.0 |
| scpnre2_red | 1768.17 | 30.20 | 29.81 | 1.25 | 817920.4 | 67061.6 |
| scpnre3_red | 126.96 | 27.00 | 27.00 | 0.00 | 40393.8 | 0.0 |
| scpnre4_red | 307.00 | 28.00 | 28.00 | 0.00 | 136325.6 | 0.0 |
| scpnre5_red | 176.62 | 28.00 | 28.00 | 0.00 | 106428.0 | 0.0 |
| scpnrf1_red | 159.62 | 14.00 | 14.00 | 0.00 | 106680.8 | 0.0 |
| scpnrf2_red | 67.52 | 15.00 | 15.00 | 0.00 | 39772.6 | 0.0 |
| scpnrf3_red | 92.29 | 14.00 | 14.00 | 0.00 | 52526.4 | 0.0 |
| scpnrf4_red | 330.85 | 14.00 | 14.00 | 0.00 | 231312.4 | 0.0 |
| scpnrf5_red | 2702.13 | 13.60 | 12.48 | 7.99 | 2554692.8 | 262469.0 |
| scpnrg1_red | 3600.01 | 176.00 | 170.05 | 3.38 | 99506.2 | 64136.8 |
| scpnrg2_red | 3600.00 | 154.40 | 151.76 | 1.71 | 193753.6 | 57422.8 |
| scpnrg3_red | 3600.00 | 167.00 | 157.32 | 5.80 | 125925.0 | 92922.0 |
| scpnrg4_red | 3600.00 | 169.25 | 159.69 | 5.64 | 117335.25 | 86499.5 |
| scpnrg5_red | 3600.01 | 168.80 | 158.28 | 6.23 | 84697.2 | 63875.4 |
| scpnrh1_red | 3600.00 | 64.20 | 53.31 | 16.96 | 83938.2 | 67028.2 |
| scpnrh2_red | 3600.01 | 64.20 | 54.07 | 15.77 | 94082.2 | 73929.0 |
| scpnrh3_red | 3600.02 | 61.00 | 50.37 | 17.41 | 91135.0 | 75517.8 |
| scpnrh4_red | 3600.01 | 58.80 | 49.10 | 16.50 | 105531.6 | 89843.4 |
| scpnrh5_red | 3600.00 | 55.60 | 47.95 | 13.76 | 106726.8 | 85340.8 |

Table 4.5: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 8$ (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 176.19 | 29.00 | 29.00 | 0.00 | 39902.4 | 0.0 |
| scpnre2_red | 1886.76 | 30.00 | 30.00 | 0.00 | 424340.6 | 0.0 |
| scpnre3_red | 99.37 | 27.00 | 27.00 | 0.00 | 13258.6 | 0.0 |
| scpnre4_red | 139.65 | 28.00 | 28.00 | 0.00 | 27220.4 | 0.0 |
| scpnre5_red | 62.42 | 28.00 | 28.00 | 0.00 | 8270.8 | 0.0 |
| scpnrf1_red | 143.12 | 14.00 | 14.00 | 0.00 | 25114.2 | 0.0 |
| scpnrf2_red | 116.08 | 15.00 | 15.00 | 0.00 | 19205.6 | 0.0 |
| scpnrf3_red | 123.43 | 14.00 | 14.00 | 0.00 | 18789.4 | 0.0 |
| scpnrf4_red | 400.25 | 14.00 | 14.00 | 0.00 | 67269.8 | 0.0 |
| scpnrf5_red | 1275.33 | 13.00 | 13.00 | 0.00 | 264024.8 | 0.0 |
| scpnrg1_red | 3600.03 | 176.00 | 169.07 | 3.94 | 51942.8 | 32845.2 |
| scpnrg2_red | 3600.03 | 154.80 | 150.47 | 2.80 | 72958.2 | 41494.2 |
| scpnrg3_red | 3600.01 | 167.20 | 156.57 | 6.36 | 57570.0 | 46974.2 |
| scpnrg4_red | 3600.03 | 170.00 | 158.10 | 7.00 | 45750.6 | 32852.8 |
| scpnrg5_red | 3600.01 | 168.40 | 157.68 | 6.37 | 53510.0 | 40741.6 |
| scpnrh1_red | 3600.07 | 64.80 | 52.87 | 18.40 | 42319.2 | 36045.8 |
| scpnrh2_red | 3600.02 | 64.60 | 53.57 | 17.06 | 44536.8 | 37754.6 |
| scpnrh3_red | 3600.05 | 61.00 | 49.89 | 18.20 | 44286.2 | 36927.6 |
| scpnrh4_red | 3600.01 | 58.60 | 48.71 | 16.87 | 53176.6 | 46208.2 |
| scpnrh5_red | 3600.00 | 55.60 | 47.55 | 14.45 | 57995.4 | 47433.0 |

Table 4.6: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 4$ with sparse structures (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 247.69 | 29.00 | 29.00 | 0.00 | 38841.4 | 0.0 |
| scpnre2_red | 2311.04 | 30.00 | 30.00 | 0.00 | 297641.6 | 0.0 |
| scpnre3_red | 174.91 | 27.00 | 27.00 | 0.00 | 23458.8 | 0.0 |
| scpnre4_red | 180.93 | 28.00 | 28.00 | 0.00 | 26730.6 | 0.0 |
| scpnre5_red | 60.52 | 28.00 | 28.00 | 0.00 | 8279.4 | 0.0 |
| scpnrf1_red | 134.10 | 14.00 | 14.00 | 0.00 | 24664.8 | 0.0 |
| scpnrf2_red | 116.01 | 15.00 | 15.00 | 0.00 | 19427.4 | 0.0 |
| scpnrf3_red | 102.34 | 14.00 | 14.00 | 0.00 | 14699.2 | 0.0 |
| scpnrf4_red | 353.92 | 14.00 | 14.00 | 0.00 | 65887.2 | 0.0 |
| scpnrf5_red | 1658.74 | 13.00 | 13.00 | 0.00 | 327632.0 | 0.0 |
| scpnrg1_red | 3600.03 | 176.00 | 169.04 | 3.96 | 51374.6 | 33833.4 |
| scpnrg2_red | 3600.03 | 154.60 | 150.44 | 2.69 | 73946.2 | 43766.0 |
| scpnrg3_red | 3600.03 | 167.60 | 156.28 | 6.75 | 55365.2 | 44862.6 |
| scpnrg4_red | 3600.06 | 170.80 | 158.13 | 7.42 | 44764.2 | 35219.6 |
| scpnrg5_red | 3600.03 | 168.80 | 157.50 | 6.69 | 48997.6 | 38422.6 |
| scpnrh1_red | 3600.03 | 65.20 | 52.91 | 18.83 | 45390.4 | 39639.4 |
| scpnrh2_red | 3600.03 | 64.60 | 53.60 | 17.02 | 45928.0 | 39620.6 |
| scpnrh3_red | 3600.02 | 60.60 | 49.96 | 17.55 | 49204.0 | 40120.2 |
| scpnrh4_red | 3600.03 | 59.00 | 48.76 | 17.34 | 56940.2 | 50737.8 |
| scpnrh5_red | 3600.02 | 55.80 | 47.49 | 14.87 | 57990.0 | 46045.6 |

Table 4.7: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 6$ (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 216.83 | 29.00 | 29.00 | 0.00 | 38206.8 | 0.0 |
| scpnre2_red | 1820.35 | 30.00 | 30.00 | 0.00 | 364074.4 | 0.0 |
| scpnre3_red | 128.99 | 27.00 | 27.00 | 0.00 | 18927.2 | 0.0 |
| scpnre4_red | 164.38 | 28.00 | 28.00 | 0.00 | 25032.8 | 0.0 |
| scpnre5_red | 63.35 | 28.00 | 28.00 | 0.00 | 8270.8 | 0.0 |
| scpnrf1_red | 139.93 | 14.00 | 14.00 | 0.00 | 24664.8 | 0.0 |
| scpnrf2_red | 119.86 | 15.00 | 15.00 | 0.00 | 19427.4 | 0.0 |
| scpnrf3_red | 109.54 | 14.00 | 14.00 | 0.00 | 14699.2 | 0.0 |
| scpnrf4_red | 399.77 | 14.00 | 14.00 | 0.00 | 65887.2 | 0.0 |
| scpnrf5_red | 1701.49 | 13.00 | 13.00 | 0.00 | 327632.0 | 0.0 |
| scpnrg1_red | 3600.02 | 176.00 | 169.12 | 3.91 | 56097.8 | 35440.6 |
| scpnrg2_red | 3600.01 | 154.60 | 150.58 | 2.60 | 81699.6 | 45690.0 |
| scpnrg3_red | 3600.01 | 167.80 | 156.5 | 6.73 | 62684.2 | 51315.0 |
| scpnrg4_red | 3600.03 | 170.40 | 158.34 | 7.07 | 51763.4 | 39266.6 |
| scpnrg5_red | 3600.03 | 168.60 | 157.69 | 6.47 | 54118.0 | 41545.6 |
| scpnrh1_red | 3600.04 | 65.00 | 52.92 | 18.58 | 45013.8 | 39231.2 |
| scpnrh2_red | 3600.03 | 64.40 | 53.60 | 16.76 | 46199.0 | 38755.6 |
| scpnrh3_red | 3600.02 | 61.00 | 49.94 | 18.13 | 47034.6 | 40187.8 |
| scpnrh4_red | 3600.02 | 59.00 | 48.73 | 17.41 | 54182.6 | 48787.0 |
| scpnrh5_red | 3600.02 | 56.20 | 47.50 | 15.47 | 57942.8 | 48687.6 |

Table 4.8: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR1 and $M_{branches} = 8$ (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | Time (s) | Best Int. | Obj Val | MIP Gap | Nodes | Nodes Left |
|---|---|---|---|---|---|---|
| scpnre1_red | 1397.60 | 29.00 | 29.00 | 0.00 | 711816.8 | 0.0 |
| scpnre2_red | 3600.05 | 30.80 | 27.34 | 11.21 | 1529080.2 | 959051.4 |
| scpnre3_red | 554.43 | 27.00 | 27.00 | 0.00 | 212619.2 | 0.0 |
| scpnre4_red | 1128.05 | 28.00 | 28.00 | 0.00 | 500035.4 | 0.0 |
| scpnre5_red | 324.79 | 28.00 | 28.00 | 0.00 | 129196.6 | 0.0 |
| scpnrf1_red | 203.96 | 14.00 | 14.00 | 0.00 | 91477.0 | 0.0 |
| scpnrf2_red | 133.72 | 15.00 | 15.00 | 0.00 | 59029.6 | 0.0 |
| scpnrf3_red | 127.05 | 14.00 | 14.00 | 0.00 | 51472.8 | 0.0 |
| scpnrf4_red | 548.66 | 14.00 | 14.00 | 0.00 | 242715.4 | 0.0 |
| scpnrf5_red | 1447.60 | 13.00 | 13.00 | 0.00 | 641883.4 | 0.0 |
| scpnrg1_red | 3600.06 | 176.00 | 164.58 | 6.49 | 187268.2 | 146501.8 |
| scpnrg2_red | 3600.00 | 155.20 | 146.47 | 5.62 | 189623.6 | 141899.6 |
| scpnrg3_red | 3600.02 | 167.60 | 152.6 | 8.95 | 217498.8 | 178434.0 |
| scpnrg4_red | 3600.00 | 170.80 | 152.52 | 10.70 | 207629.8 | 175621.0 |
| scpnrg5_red | 3600.00 | 168.40 | 151.63 | 9.96 | 198831.4 | 169853.4 |
| scpnrh1_red | 3600.00 | 64.20 | 50.71 | 21.01 | 343996.4 | 320998.2 |
| scpnrh2_red | 3600.00 | 64.60 | 51.17 | 20.78 | 347998.0 | 324861.6 |
| scpnrh3_red | 3600.01 | 61.20 | 47.76 | 21.95 | 353973.6 | 328563.0 |
| scpnrh4_red | 3600.01 | 58.80 | 46.80 | 20.41 | 374854.0 | 343725.6 |
| scpnrh5_red | 3600.00 | 55.40 | 45.33 | 18.17 | 375063.0 | 336239.8 |

Table 4.9: Results for the CPLEX solver on reduced $e$, $f$, $g$ and $h$ instances using BR2 (3600 seconds time limit, 1 thread, average on 5 runs).

| Instance | CPX1 | CPX2 | BR1 4 | BR1 6 | BR1 4 sp | BR1 6 sp | BR2 |
|---|---|---|---|---|---|---|---|
| scpnre1_red | 77.76 | **70.71** | 164.79 | 190.69 | 176.19 | 247.69 | 1397.60 |
| scpnre2_red | 666.22 | **388.94** | 798.47 | 1960.90 | 1886.76 | 2311.04 | 3600.05 |
| scpnre3_red | 73.32 | **60.30** | 113.49 | 90.38 | 99.37 | 174.91 | 554.43 |
| scpnre4_red | 209.86 | 127.12 | **123.24** | 134.54 | 139.65 | 180.93 | 1128.05 |
| scpnre5_red | 71.36 | **35.26** | 44.72 | 68.74 | 62.42 | 60.52 | 324.79 |
| scpnrf1_red | **74.90** | 79.07 | 154.81 | 245.22 | 143.12 | 134.10 | 203.96 |
| scpnrf2_red | 47.04 | **50.13** | 113.54 | 106.39 | 116.08 | 116.01 | 133.72 |
| scpnrf3_red | **18.83** | 36.36 | 150.39 | 101.24 | 123.43 | 102.34 | 127.05 |
| scpnrf4_red | **118.06** | 132.68 | 344.91 | 678.97 | 400.25 | 353.92 | 548.66 |
| scpnrf5_red | 946.24 | **493.57** | 2368.40 | 3073.02 | 1275.33 | 1658.74 | 1447.60 |

Table 4.10: Results comparison of the previous tables: *time* comparison for $e$ and $f$ instances.

| Instance | CPX1 | CPX2 | BR1 4 | BR1 6 | BR1 4 sp | BR1 6 sp | BR2 |
|---|---|---|---|---|---|---|---|
| scpnrg1_red | 2.94 | **2.84** | 3.42 | 3.18 | 3.94 | 3.96 | 6.49 |
| scpnrg2_red | **0.00** | 0.20 | 2.58 | 2.32 | 2.80 | 2.69 | 5.62 |
| scpnrg3_red | **4.35** | 4.54 | 6.00 | 5.76 | 6.36 | 6.75 | 8.95 |
| scpnrg4_red | **4.41** | 5.22 | 6.82 | 5.18 | 7.00 | 7.42 | 10.70 |
| scpnrg5_red | 4.78 | **4.51** | 6.19 | 5.91 | 6.37 | 6.69 | 9.96 |
| scpnrh1_red | 16.51 | **15.37** | 17.96 | 17.91 | 18.40 | 18.83 | 21.01 |
| scpnrh2_red | 15.43 | **14.62** | 15.47 | 16.31 | 17.06 | 17.02 | 20.78 |
| scpnrh3_red | **14.96** | 15.00 | 16.79 | 17.13 | 18.20 | 17.55 | 21.95 |
| scpnrh4_red | **14.71** | 15.42 | 15.85 | 16.46 | 16.87 | 17.34 | 20.41 |
| scpnrh5_red | 11.76 | **11.52** | 14.79 | 14.32 | 14.45 | 14.87 | 18.17 |

Table 4.11: Results comparison of the previous tables: *MIP Gap* comparison for $g$ and $h$ instances.

# Chapter 5

# Conclusions

The negative results of this research show how the technology in the field of MILP solvers has advanced since the years in which [1] [2] were published. Although it is already possible to observe the integrated use of branching and cutting plane techniques, it is worth remembering that the publication of the *Branch-and-Cut* method by Padberg and Rinaldi [16] would have taken place only a decade later.

As far as the SCP is concerned, even removing the cutting planes method, we notice how the branching techniques implemented by CPLEX remain extremely competitive.

This could be due to the very nature of the problem. As reported in Chapter 1, since the birth of linear programming, the SCP has always been one of the key problems of the discipline, and it is straightforward to describe and extremely abstract.

Consequently, decades of research in the field of mathematical optimization have led to an extremely complex and refined product, whose performance can hardly be improved when general problems such as the SCP are treated.

# Appendix A

# Helpful Resources

## A.1 Software Description

The software developed for this project is organized as reported here

- *main.c* contains the starting routines for reading the input file, parsing parameters passed as an argument and to initialize the *instance* structure.
- *aux.c* contains the definition for the *instance* structure. This structure defines the data we require to pass throughout the software.
- Inside *sc.c* is organized the central logic of the software. The `SCMILPsolver` function executes the following phases:
  1. the model is read from an *.lp* file or, alternatively, is built from a raw file (as described in OR-Library [4]);
  2. the subsequent phase is the *pre-solver* selection (this is an optional stage);
  3. the next passage consists of deciding the solver as required by the user;
  4. ultimately, we require CPLEX the solution and the information that we print out for the user.
- In *preprocessing.c* is included the `SCdominancepresolver` function. The function is the direct implementation of Algorithm 1 as reported in Chapter 2.
- *callbacks.c* contains the bodies of the main callback functions. These callbacks are direct called by the solvers in *sc.c*. Several utility functions are included in the same file, such us the `SCgetmatrix` function that processes the matrix provided by CPLEX.
- *balas_ dense.c* contains the algorithms reported in [2].

- *balas_ sparse.c* contains the algorithms reported in [2], implemented by means of sparse data structures.

## A.2 CPLEX Set Up

Using the CPLEX library with the C programming language can sometimes be complicated for a novice.

The following online resource can be helpful if is required to adopt this approach and by using CMake to manage the project. It can be used both for OSX and Linux systems.

`https://github.com/ampl/mp/blob/master/support/cmake/FindCPLEX.cmake`

# References


[1] Egon Balas. *Cutting planes from conditional bounds: A new approach to set covering*, pages 19–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980.

[2] Egon Balas and Andrew Ho. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, pages 37–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980.

[3] J. E. Beasley. A lagrangian heuristic for set-covering problems. *Naval Research Logistics (NRL)*, 37(1):151–164, 1990.

[4] J. E. Beasley. Or-library, 1990.

[5] M. Bellmore, H. J. Geenberg, and J. J. Jarvis. Multi-commodity disconnecting sets. *Management Science*, 16(6):B–427–B–433, 1970.

[6] Mandell Bellmore and H. Donald Ratliff. Optimal defense of multi-commodity networks. *Management Science*, 18(4):B174–B185, 1971.

[7] Mandell Bellmore and H. Donald Ratliff. Set covering and involutory bases. *Management Science*, 18(3):194–206, 1971.

[8] Alberto Caprara and Matteo Fischetti. A heuristic method for the set covering problem. *Operations Research*, 47, 02 2000.

[9] Graham Cormode, Howard Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 479–488, New York, NY, USA, 2010. ACM.

[10] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Manage. Sci.*, 6(1):80–91, October 1959.

[11] Richard H. Day. On optimal extracting from a multiple file data storage system: An application of integer programming. *Operations Research*, 13(3):482–494, 1965.

[12] Michael Ferris. Solving the seymour problem. 2001.

[13] Matteo Fischetti. *Lezioni di Ricerca Operativa*. Libreria Progetto, Padova, 1995.

[14] Ibm. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*, 2011.

[15] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[16] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, February 1991.

[17] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.