Università degli Studi di Firenze
Engineering Faculty

Ph.D. course
Ingegneria informatica e dell'automazione

Area of study
MAT/09 – Operational Research

Decomposition methods and global optimization algorithms for large scale nonlinear problems

Ph.D. thesis by

David Di Lorenzo

Supervisors:

Fabio Schoen
Marco Sciandrone

Contents

Chapter 1

Introduction

While decades of progress in mathematics and electronics have greatly increased the range of problems that can actually be solved by computers, new and more complex problems have emerged and are posing new challenges to the scientific community. Problem size, while obviously not be the only factor that affects the efficiency of an algorithm, still plays a major role. Other properties that determine if a problem can be considered easy or hard are the problem type, like linear, convex, integer, etc..., or simply the amount of computer resources available, which is constantly changing. In light of such considerations, it is not clear what "large scale" means in this context. For the purpose of this thesis, when we talk about large scale problems, we will be referring to the size of problems, of a particular class, that cannot be easily solved by the tools we can use today, but that are nevertheless not so large to be clearly intractable without a major breakthrough in mathematics or in computer engineering. We would like to stress the importance of the problem type in this definition. While we can solve some kind of linear problems with billions of variables and constraints, some combinatorial problems which can be expressed with a few tens of variables are still considered extremely difficult, if not simply unsolvable.

Covering all the techniques that are used to tackle large scale problems is a topic so vast that must necessarily fall outside the scope of the thesis. It is however safe to say that the methods used usually lie within one of three approaches, that we will briefly summarize here, and we will describe in more detail in the other chapters of this thesis.

Large scale methods for convex programming

Let us consider problems where the objective function is convex and smooth, and the feasible set is also convex and has a simple structure. In this case, difficulties arise only where the number of variables is very high. One of the simplest strategies that can be applied to solve such problems is to sequentially solve a large number of smaller problems which are easy, if not trivial. This can be easily done by an algorithm which selects, at every iteration, a small subset of the variables, and solves the restricted problem that emerges from fixing all the non selected variables to their current value. This class of approaches are called decomposition methods, and can be classified by the way the variables are chosen during the iterations, and by the type of step that is applied.

The question arises of whenever the chosen decomposition strategy converges to the same points that we would have computed by solving the original problem. It turns out that, while little or no assumptions are needed on the objective function, strong assumptions are usually required for the structure of the feasible set. This happens because the sequence of points generated by those algorithms needs to be feasible. In fact it is easy to construct sets that have points in which all feasible directions have nonzero values in every variable, so no step can be taken from such point unless all the variables are allowed to be changed, which defeats the purpose of decomposition in the first place. This also explains in some way why no such assumption is required on the objective function. Since the points generated by an algorithm are obviously not required to be all optimal, there is no need to avoid situations like the one described above, so no additional requirement must be imposed on the objective function. We can imagine that, by introducing algorithms that do not require the sequence of points to be all feasible, the restrictions on the feasible set structure could be lifted, but such topic falls outside the scope of this thesis.

Applications where the decomposition can be applied are countless. Some examples include image processing [12], control systems [25], neural networks training [16], equilibrium problems [33] and SVM training [57]. The main contributions presented in this thesis are a rigorous analysis of the convergence properties of various decomposition strategies, and their application to the problem of network equilibrium.

Chapter 3 will describe various decomposition strategies and their theoretical properties. Section 3.2 of such chapter describes a previously known decomposition algorithm that can be used to solve SVM training problems, and partially settles an open problem about its convergence. It is well known, in fact, that the optimization problem involved in the training of SVM is convex and quadratic, and that the structure of its dual formulation allows for some decomposition methods to be successfully applied.

Chapter 4 explains how some decomposition algorithms can be used to solve the network equilibrium problem, a situation that emerges from game theory, where a set of users travel on a graph whose arc costs depend on the number of users that are using it, allowing congestion situations to be accounted for in the model. As we will see, the straightforward way to model such problem is not fit to be decomposed, but a reformulation can be applied which greatly simplifies the constraint structure of the problem, which becomes a Cartesian product of simplices, at the cost of having a number of variables that scales to the number of possible paths in the graph, which is clearly too high to even fit in memory, so decomposition methods can be used not only to speed up the convergence rate, but also to avoid the need of actually storing all such values.

Continuous global optimization methods for large scale nonlinear mixed integer problems

Problems with integrality constraints are usually extremely more difficult to solve than continuous problems of the same size. Because of this, they are considered difficult (and so "large scale") even if the number of variables is not extremely high. The techniques that can be used to tackle such problem are countless. Since obviously no single strategy exists that can be successfully applied to such wide range of problems, we will focus on describing some reformulations and the algorithms that can be used to solve them. As we will see later, it will be sometimes possible to fix a large number of variables to zero while keeping the algorithm behaviour unchanged. This cannot be done at every iteration, but nevertheless this technique, while not as powerful as the ones used for convex problems, can still be applied to great benefit.

It is well known that some mixed integer problems can be reformulated into continuous equivalent problems [73] [75] [76]. Even with the integer constraints removed, the problems are still hard to solve exactly, because of the high number of local minima introduced by the concavity of the objective function. However, by carefully choosing the local optimization algorithm, we can prove that many of variables can be fixed to zero without affecting the convergence of the algorithm. Such number of variables increases at every iteration, but is however equal to zero in the first one. Then, at the expense of still having to compute one step of the algorithm with the full set of variables, subsequent iterations can be applied to a problem whose size has been greatly reduced, so that in the end only the very first iterations are relevant when measuring the CPU time.

The main contribution for this class of problems in this thesis is presented in chapter 5, where such methods are described in detail and are applied to a problem arising in finance. Many of the results of this chapter are taken from [29]. An investor wishes to distribute a given capital to a number of assets. We are assuming that the mean return and the variance are known for every such stock. Being the user risk adverse, he wishes to minimize the total variance of the portfolio while keeping a minimum average return. Such problem was proposed by Markowitz [62], it is well known in the finance community, and easy to solve. However optimal solutions on real world data usually tend, in order to minimize the variance, to split the capital into a number of assets which is too high for the solution to have any practical sense. To mitigate this problem, we can put a bound on the number of stocks that can be purchased, but in doing so integer variables are introduced, which render the problem NP–hard. Nevertheless, we will see that the technique previously described can be applied to render the problem easier to solve.

Large scale methods for nonconvex programming

In cases where the objective function and the constraints are non convex, the main difficulty in solving the problem arises from the high number of local minima which are not global. Because of this, the size where those problems are considered difficult is much smaller with respect to the other previously mentioned classes. In fact, sometimes the problem bounds are either so complex that no simple way to handle them can be found, or are simply unknown, like in black–box functions. Taking this into account, sometimes we must give up on trying to find a certified optimum, and use an heuristic approach to find good enough solutions. Typical application arise in engineering, when the objective function and the constraints are computed through a simulation, or another iterative algorithms, and cannot be expressed in a closed form. Furthermore, even if such functions are smooth, the gradient is usually not available or too expensive to be actually computed. Heuristics algorithms rely on the assumption that, while unknown to us, such problems have some kind of structure, so that good solutions are in some way close to each other, and this allows the design of algorithms that are better than a sequence of local optimization from random starting points.

The main contribution to this field presented in this thesis is described in chapter 6, with an application related to spacecraft trajectory optimization.

Much of the work described here has been published in [93] and [94]. The problem consists in designing the trajectory of a spacecraft that leaves Earth and reaches a given planetary destination with the least possible amount of fuel. The physical equations that govern the motion of the body in the Solar system are strongly nonlinear, and the problem becomes difficult to solve because of the number of local minimizers that are not global. On the other hand, mission designers are not interested in the (nearly impossible) task of finding a certified optimal point, but just in finding good enough solutions, so heuristic approaches can be used. Automatic trajectory design has already been studied [66] [47] [17] [1], however no such analysis has been done for low–thrust missions. In standard missions there is no explicit constraint on the maximum engine power, and the optimal solutions usually resulted in the engine being lighted in only a few points in the trajectory, using all the available fuel in a very short time. In low–thrust missions, because of a different type of engine, such solution are unfeasible because the engine power is very low with respect to chemically propelled spacecrafts, the advantage being that the engine can stay on for a very long time, even months, with a very small total fuel usage.

After having written the model, we will analyze various algorithms that can be used to solve it, and describe the solutions that have been found.

1.1 Notation

The n-dimensional space of Real, Rational and Integer numbers are denoted by $\mathbb{R}^n$, $\mathbb{Q}^n$ and $\mathbb{Z}^n$. The plus subscript, like in $\mathbb{R}^n_+$, restrict the set to the nonnegative numbers.
Vectors are usually denoted by lower case letters, and will always be considered column vectors. We write x' to transpose the vector (or matrix) x. We use the subscript notation x_i to refer to the i-th value of the vector x. Indexes start from the number one. A matrix $A \in \mathbb{R}^{n \times m}$ will have n rows and m columns. The symbol $I(n)$ represents the n by n identity matrix, and will be often written as I, since the dimension can be easily inferred from the context. Similarly, the symbol $e(n)$ represents a vector of ones of dimension n, and will often be written as e, since the n can be easily deduced.

Since we are dealing with decomposition, we require a notation to extract the i-th subvector of a given vector x. To that end, we enclose the subscript inside a parenthesis. So, $x_{(i)}$ is the i-th subvector of the vector x. The exact variables that constitute the beginning and ending of the i-th subvector can be inferred from the context, usually in the definition of x. Similarly, ${x_{(i)}}_j$ is the

j-th variable of the i-th subvector of x.
We also define an operation that is, in some way, the inverse of $x_{(i)}$. Given a vector x, and a subvector y which belongs to the i-th subvector space of x, we can write $x = y_{(-i)}$ if x is equal to y for the variables belonging to the i-th block, and all the other values are equal to zero. Formally

$$y_{(-i)_j} = \begin{cases} y_j & \text{if } j \in i \\ 0 & \text{if } j \notin i \end{cases}$$

If easily follows that $\left(y_{(-i)}\right)_{(i)} = y$, but in general $\left(x_{(i)}\right)_{(-i)} \neq x$.
Superscripts are used instead to refer to an element of a sequence, so x^k may represent the k-th element of a previously defined sequence, or the value taken by x at the k-th step of an algorithm.
To denote the p-th norm of the vector x, we write $\|x\|_p$. If no p is specified, then the 2-norm is used. Although is not a norm, we will use $\|x\|_0$ to refer to the so called zero norm of x, the number of components of x which are different from zero.
Curly brackets are used, depending on the context, to represent both sets and sequences. $|\mathcal{S}|$ denotes the cardinality of the set $\mathcal{S}$. For any set $\mathcal{S}$, $2^{\mathcal{S}}$ is used to represent the set of subsets of $\mathcal{S}$. Obviously $\left|2^{\mathcal{S}}\right| = 2^{|\mathcal{S}|}$.
Usually, but this is not an hard rule, the star suffix, like x^*, is used to define the optimal value of a problem, while a bar, like in $\bar{x}$, defines the limit value of a sequence or algorithm.
The operation $x \bmod y$ denotes as usual the reminder of the integer division between x and y. Since we are adhering to the commonly used convention that vector indexes start from 1 instead of 0, it is useful to define a "base 1" modulus operator, denoted by $\bmod^{+1}$ and defined as

$$x \bmod^{+1} y = ((x-1) \bmod y) + 1$$

such that $1 \bmod^{+1} 5 = 1, 1 \bmod^{+1} 2 = 2, \ldots, 5 \bmod^{+1} 5 = 5, 6 \bmod^{+1} 5 = 1, \ldots$.

A function $f : \mathcal{X} \to \mathbb{R}$ will be called smooth if it is continuously differentiable everywhere on $\mathcal{X}$.

A function $\delta : \mathbb{R}_+ \to \mathbb{R}_+$ is a forcing function if for every sequence $\{x^k\} \subset \mathbb{R}$ such that

$$\lim_{k\to\infty} \delta(x^k) = 0$$

it also holds

$$\lim_{k\to\infty} x^k = 0$$

Chapter 2

Algorithms for problems with simple convex feasible sets

This chapter will introduce the Frank-Wolfe and Projected Gradient algorithms, and describe some properties that will become useful for the following chapters of the thesis. As we will see later on many methods, including the ones based on decomposition, require a procedure that generates a new point, or at least a feasible descent direction. The main reason why those algorithms are so widely used are that

- They are able to compute an improvement over a given initial point
- They are not expensive with regard to CPU resources
- They have nice convergence properties

On the downside, in cases where an exact solution is needed, such algorithms must applied for an high number of iterations, until convergence, and thus become too expensive to actually be used. However we will show that, in particular when dealing with decomposition methods, there is no need to compute an accurate solution at every iteration, but inexact solutions can be used as well, with a significant saving in computational times.

As a final note, those two algorithms require the feasible sets to be convex. However, as a practical additional requirement, the sets must also be, in some way, “simple”. This happens because in both methods a second optimization problem, dependent on the feasible set, must be solved. If the constraints are simple, the subproblems can be solved, if not analytically, at least in a trivial way. If, on the other way, the constraints are complex, the computational time required to solve the subproblems can render the whole algorithm too slow to have any practical value.

2.1 Frank-Wolfe direction

Let $\mathcal{C} \subset \mathbb{R}^n$ be a convex compact set, and let $f(x) : \mathcal{C} \to \mathbb{R}$ be a smooth function. For convenience, we will define $FW_{\mathcal{C}} : \mathcal{C} \to \mathbb{R}^n$ as a function which satisfies

$$FW_{\mathcal{C}}(x) \in \arg\min_{y \in \mathcal{C}} \nabla f(x)'(y - x) \tag{2.1}$$

Similarly, we will define function $\overline{FW}_{\mathcal{C}} : \mathcal{C} \to \mathbb{R}$ as

$$\overline{FW}_{\mathcal{C}}(x) = -\min_{y \in \mathcal{C}} \nabla f(x)'(y - x) \tag{2.2}$$

The subscript $\mathcal{C}$ might be omitted if it is clear from the context. The following theorems state some properties of such functions, which will be used later

Theorem 2.1. *If $\nabla f(x)'(FW(x) - x) = 0$, then x satisfies the first order optimality conditions of problem*

$$\min_{x \in \mathcal{C}} f(x)$$

Proof. Consider the optimization problem in equation (2.1). $y = x$ is always a feasible solution of such problem, with an objective function value of 0. Because of the hypothesis, such solution is also optimal, so we can write

$$x \in \arg\min_{y \in \mathcal{C}} \nabla f(x)'(y - x)$$

which is a first order optimality condition. □

Theorem 2.2. *Function $\overline{FW}(x)$ is continuous.*

Proof. We can write

$$\overline{FW}(x) = -\min_{y \in \mathcal{C}} (\nabla f(x)'y - \nabla f(x)'x) = \nabla f(x)'x - \min_{y \in \mathcal{C}} \nabla f(x)'y$$

The first term is obviously continuous. To prove that $-\min_{y \in \mathcal{C}} \nabla f(x)'y$ is continuous it is sufficient to prove that the function $h(c) = \min_{y \in \mathcal{C}} c'y$ is concave, as concavity on $\mathbb{R}^n$ implies continuity, and the composition of two continuous

functions – that is, $h(\nabla f(x))$ in our case – is continuous.
Let $c_1, c_2 \in \mathbb{R}^n$, and let $\lambda \in [0,1]$. Then

$$h(\lambda c_1 + (1-\lambda)c_2) = \min_{y\in\mathcal{C}} [\lambda c_1' y + (1-\lambda)c_2' y] \geq$$

$$\geq \min_{y_1\in\mathcal{C}, y_2\in\mathcal{C}} [\lambda c_1' y_1 + (1-\lambda)c_2' y_2] =$$

$$= \lambda \min_{y_1\in\mathcal{C}} c_1' y_1 + (1-\lambda) \min_{y_2\in\mathcal{C}} c_2' y_2 = \lambda h(c_1) + (1-\lambda) h(c_2)$$ □

Theorem 2.3. *Function $\nabla f(x)' FW(x)$ is continuous.*

Proof. Trivially, it holds

$$\nabla f(x)' FW(x) = \left(\min_{y\in\mathcal{C}} \nabla f(x)'(y-x)\right) + \nabla f(x)' x = -\overline{FW}(x) + \nabla f(x)' x$$

which is continuous by theorem 2.2. □

2.2 Frank-Wolfe algorithm

The Frank-Wolfe algorithm [35], also called Conditional Gradient method, is a widely used convergent optimization method which, together with the Armijo linesearch, can be used to solve general optimization problems of the form

$$\begin{aligned} &\min f(x) \\ &x \in \mathcal{C} \end{aligned} \tag{2.3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function and $\mathcal{C} \subset \mathbb{R}^n$ is a convex compact set. Algorithm 1 describes such procedure. Its convergence results are well known.

Algorithm 1: Frank-Wolfe method

```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     $\hat{x}^k \leftarrow FW(x^k)$;
4     $d^k \leftarrow \hat{x}^k - x^k$;
5     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
6     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
7     $k \leftarrow k + 1$;
8 end
```

Theorem 2.4. *Any limit point of algorithm 1 is a first order stationary point of problem* (2.3).

Proof. See [7, p. 217].

2.3 Projection

We will first recall some known results about the projection operation. Let $x \in \mathbb{R}^n$, and let $\mathcal{C} \subseteq \mathbb{R}^n$ be a closed convex nonempty set. The projection of x over $\mathcal{C}$ is denoted as $P_{\mathcal{C}}[x]$ and is defined as

$$P_{\mathcal{C}}[x] = \arg\min_{y \in \mathcal{C}} \|x - y\| \tag{2.4}$$

The following theorem ensures that $P_{\mathcal{C}}[x]$ is well defined

Theorem 2.5. *Let $x \in \mathbb{R}^n$, and let $\mathcal{C} \subseteq \mathbb{R}^n$ be a closed convex nonempty set. Then the solution to equation* (2.4) *exists and is unique. Also, $x^\star$ solves problem* (2.4) *if and only if*

$$(y - x^\star)'(x - x^\star) \leq 0 \qquad \forall y \in \mathcal{C} \tag{2.5}$$

Furthermore, the function $f : \mathbb{R}^n \to \mathcal{C}$ defined as $f(x) = P_{\mathcal{C}}[x]$ is continuous and non expansive, i.e.

$$\|f(x) - f(y)\| \leq \|x - y\| \qquad \forall x, y \in \mathbb{R}^n$$

Proof. See [7, Proposition B.11, page 685].

The following theorem shows how the projection operation can be used to compute a feasible descent direction

Theorem 2.6. *Let $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}$ be a convex compact set, and let $f : \mathcal{C} \to \mathbb{R}$ be a smooth function. Let $x \in \mathcal{C}$, $s \in \mathbb{R}$ and*

$$d = P_{\mathcal{C}}[x - s\nabla f(x)] - x$$

Then either $d = 0$ or d is a feasible descent direction at x.

Proof. From equation (2.5), by choosing $y = x$ we have

$$(x - (x + d))'((x - s\nabla f(x)) - (x + d)) \leq 0$$

or

$$d'(s\nabla f(x) + d) \leq 0$$

$$sd'\nabla f(x) \leq -\|d\| \leq 0$$

which implies the thesis. □

It should be noted that the projection over the gradient direction can be used to check for optimality of a point. In particular

Theorem 2.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function and let $\mathcal{C} \subseteq \mathbb{R}^n$ be a convex set. Let $s > 0$. Then $x^\star$ is a first order optimality condition of f over $\mathcal{C}$ if and only if*

$$P_{\mathcal{C}}[x - s\nabla f(x)] = x$$

Proof. See [7, cap. 2.3.1].

2.4 Projected Gradient algorithm

The Projected Gradient algorithm is a general optimization algorithm that under mild assumptions converges to a first order stationary point. It consists of a sequence of Armijo linesearches along the direction given by the gradient projected into the feasible set.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function, and let $\mathcal{C}$ be a nonempty convex set. Let $\{s^k\}$ be any sequence such that there exists $0 < l \leq u$ such that $l \leq s^k \leq u$. Algorithm 2 describes the Projected Gradient method. The

Algorithm 2: Projected Gradient

```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     $\hat{x}^k \leftarrow P_{\mathcal{C}}[x^k - s^k \nabla f(x^k)]$;
4     $d^k \leftarrow \hat{x}^k - x^k$;
5     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
6     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
7     $k \leftarrow k + 1$;
8 end
```

convergence properties of algorithm 2 are well known.

Theorem 2.8. *Any limit point of algorithm 2 is a first order stationary point of problem* (2.3).

Proof. See [7, cap. 2.3].

2.5 Relationship between the Frank-Wolfe and Projected Gradient algorithms

2.5.1 Impact of the parameter s

As previously described, the Projected Gradient algorithm depends on the sequence $\{s^k\}$, which is used to scale the gradient value. We will show that the Frank-Wolfe algorithm can be seen as an extension of the gradient method, as it produces at every iteration k the same descent direction as the gradient projection method when the parameter s^k equals infinity. Formally, at each step of the algorithm, we consider the set of directions that are computed by the Frank Wolfe algorithm. In the general case such set contains more than one element, as the Frank Wolfe solution is obtained by solving an optimization problem that does not guaranties to generate an unique result. We also consider, at each step, the direction computed by the gradient projection method as a function of the parameter s^k. We will prove that, as s^k goes to infinity, every limit point of such function generates a vector that belongs to the set of Frank Wolfe directions. Furthermore, if the feasible set is a polytope, we will prove that such property also holds for a finite value of s^k.

Equivalence proof

We will prove that, at each iteration, any limit point computed by the gradient projection method, as s goes to infinity, is also a Frank-Wolfe solution. Since the only difference between the two algorithms lies in the method used to generate the point $\hat{x}^k$, which can be easily seen from algorithms 1 and 2, it is sufficient to prove that the limit points for s going to infinity of problem

$$\begin{aligned} &\min_y \|x - s\nabla f(x) - y\| \\ &y \in \mathcal{C} \end{aligned} \tag{2.6}$$

also solve problem

$$\begin{aligned} &\min_y y'\nabla f(x) \\ &y \in \mathcal{C} \end{aligned} \tag{2.7}$$

Without additional assumptions, more than one limit point may exist, and we will prove that all of them are Frank-Wolfe points, i.e. solve problem (2.7).

Theorem 2.9. *Let $\mathcal{C} \subset \mathbb{R}^n$ be a convex compact set. Let $x, d \in \mathbb{R}^n$. For any*

$s > 0$, let

$$P_{\mathcal{C}}[x - sd] = \arg\min_{y\in\mathcal{C}} \|y - (x - sd)\| \tag{2.8}$$

Let $\{s^j\}$ be a sequence such that $s^j > 0$ and

$$\lim_{j\to\infty} s^j = \infty$$

Let $\bar{y}$ be an accumulation point of sequence $\{P_{\mathcal{C}}[x - s^j d]\}$ as j goes to infinity. Then

$$\bar{y} \in \arg\min_{y\in\mathcal{C}} d'(y - x)$$

Proof. Consider

$$\begin{aligned}
&P_{\mathcal{C}}[x - sd] = \\
&= \arg\min_{y\in\mathcal{C}} \|y - (x - sd)\| = \\
&= \arg\min_{y\in\mathcal{C}} \|(y - x) + sd\|^2 = \\
&= \arg\min_{y\in\mathcal{C}} \left\{ \|y - x\|^2 + 2sd'(y - x) + \|sd\|^2 \right\} = \\
&= \arg\min_{y\in\mathcal{C}} \left\{ \|y - x\|^2 + 2sd'(y - x) \right\} = \\
&= \arg\min_{y\in\mathcal{C}} \left\{ \frac{\|y - x\|^2}{2s} + d'(y - x) \right\}
\end{aligned} \tag{2.9}$$

Since $\mathcal{C}$ is a compact set, given any fixed $x \in \mathbb{R}$, it holds that for every $y \in \mathcal{C}$, the value $\|y - x\|$ is bounded by a constant B_x. We can write for every $s > 0$

$$\begin{aligned}
\min_{y\in\mathcal{C}} d'(y - x) &\le \min_{y\in\mathcal{C}} \frac{\|y - x\|^2}{2s} + d'(y - x) \le \\
&\le \min_{y\in\mathcal{C}} \frac{B_x^2}{2s} + d'(y - x) = \\
&= \frac{B_x^2}{2s} + \min_{y\in\mathcal{C}} d'(y - x)
\end{aligned}$$

Then it must hold

$$\lim_{s\to\infty} \min_{y\in\mathcal{C}} \frac{\|y - x\|^2}{2s} + d'(y - x) = \min_{y\in\mathcal{C}} d'(y - x) \tag{2.10}$$

It trivially holds, for every $s > 0$

$$\begin{aligned} &d'\left(P_{\mathcal{C}}[x - sd] - x\right) \leq \\ \leq &\frac{\|P_{\mathcal{C}}[x - sd] - x\|^2}{2s} + d'\left(P_{\mathcal{C}}[x - sd] - x\right) \end{aligned} \tag{2.11}$$

Consider

$$\phi(y) = \frac{\|y - x\|^2}{2s} + d'(y - x)$$

By definition of the arg min operator, it holds

$$\phi(\arg\min \phi(y)) = \min \phi(y)$$

Then using equations (2.11) and (2.9) we can write

$$d'\left(P_{\mathcal{C}}[x - sd] - x\right) \leq \min_{y \in \mathcal{C}} \frac{\|y - x\|^2}{2s} + d'(y - x) \tag{2.12}$$

Let $J \subseteq \mathbb{N}$ be an infinite set such that

$$\lim_{\substack{j \to \infty \\ j \in J}} P_{\mathcal{C}}[x - s^j d] = \bar{y} \tag{2.13}$$

Taking the limit of equation (2.12) we can write

$$\lim_{\substack{j \to \infty \\ j \in J}} d'\left(P_{\mathcal{C}}[x - s^j d] - x\right) \leq \lim_{\substack{j \to \infty \\ j \in J}} \min_{y \in \mathcal{C}} \frac{\|y - x\|^2}{2s^j} + d'(y - x)$$

Note that the left hand side limit must exist, as $P_{\mathcal{C}}[x - s^j d]$ converges to $\bar{y}$. The right hand side limit must exist because such function is bounded and non increasing as s^j goes to infinity. Then

$$d'\left(\lim_{\substack{j \to \infty \\ j \in J}} P_{\mathcal{C}}[x - s^j d] - x\right) \leq \lim_{\substack{j \to \infty \\ j \in J}} \min_{y \in \mathcal{C}} \frac{\|y - x\|^2}{2s^j} + d'(y - x)$$

By equations (2.13) and (2.10)

$$d'\left(\bar{y} - x\right) \leq \min_{y \in \mathcal{C}} d'(y - x)$$

which implies that

$$\bar{y} \in \arg\min_{y \in \mathcal{C}} d'(y - x) \qquad \square$$

Optimization over a polytope

We will derive a stronger result that holds if the feasible set $\mathcal{C}$ is a polytope.

Theorem 2.10. *Let $\mathcal{C} \subset \mathbb{R}^n$ be a polytope, defined as $\{x : Ax \leq b\}$ for some $A \in \mathbb{R}^{n \times l}$ and $b \in \mathbb{R}^n$. Let $x \in \mathbb{R}^n$, $d \in \mathbb{R}^n$. Consider function $p : \mathbb{R} \to \mathbb{R}^n$ defined as*

$$p(s) = P_{Ax \leq b}[x - sd]$$

Then

(i) The image of p for $s \geq 0$ is a piecewise linear curve, i.e. there exist $s_1, \ldots, s_{m-1} \in \mathbb{R}$, $\alpha_1, \ldots, \alpha_m \in \mathbb{R}^n$, and $\beta_1, \ldots, \beta_m \in R^n$ such that

$$p(s) = \begin{cases} \alpha_1 s + \beta_1 & \text{if } s \in [0, s_1) \\ \alpha_2 s + \beta_2 & \text{if } s \in [s_1, s_2) \\ \vdots & \\ \alpha_{m-1} s + \beta_{m-1} & \text{if } s \in [s_{m-2}, s_{m-1}) \\ \alpha_m s + \beta_m & \text{if } s \in [s_{m-1}, \infty) \end{cases}$$

and such that for every i, $\alpha_i s_i + \beta_i = \alpha_{i+1} s_i + \beta_{i+1}$.

(ii) For every $s \geq s_{m-1}$, it holds

$$p(s) \in \arg\min_{y \in \mathcal{C}} d'(y - x)$$

Sketch of proof

In order to prove the first point of the theorem, we will first write the optimality condition for the minimization problem which defines the projection operation. We will see that, inside each interval (s_{i-1}, s_i), all the equations that describe the optimality conditions are linear. Then the image of the projection for $s \in (s_{i-1}, s_i)$ is an open segment. Since the projection operation is also continuous, then the image of $p(s)$ must be a continuous chain of segments, i.e. a piecewise linear curve. Finally, we will note that the vector α_m must be equal to zero, otherwise $p(s)$ would be unbounded as s goes to infinity, which is not possible since the feasible set is a compact. This implies that $p(s)$ is constant for $s \geq s_{m-1}$. Then using theorem 2.9 we can prove the second point of the theorem. An example of $p(s)$ is displayed in figure 2.1.

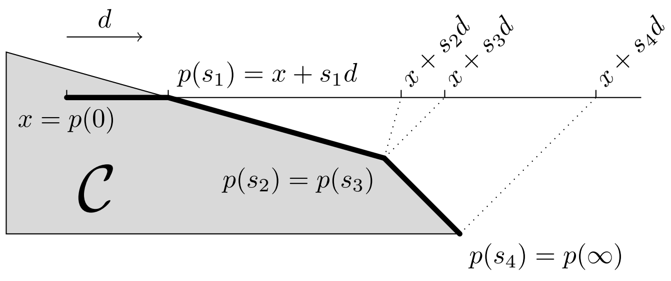

Figure 2.1: Projections on a polyhedral set $\mathcal{C}$ as a function of s. The horizontal line represents $x+sd$ for $s \in [0,\infty)$. The thicker piecewise linear curve represents the image of p for $s \in [0,\infty)$. Note that for each segment $(p(s_i), p(s_{i+1}))$, as will be proved later, all the points contained in such interval have the same active constraints. In this example $\alpha_3 = 0$, as $p(s_2) = p(s_3)$.

Actual proof

We will first present three theorems that will be used in the proof.

Theorem 2.11. *Consider*

$$\begin{aligned} \min_{z\in\mathbb{R}^n} \frac{\|z\|^2}{2} + c'z \\ Az \le b \end{aligned} \tag{2.14}$$

Then, z solves problem (2.14) *if and only if there exist $u \in \mathbb{R}^n$ such that*

$$\begin{aligned} z + A'u + c &= 0 \\ u &\ge 0 \\ Az &\le b \\ (A_i z - b_i)' u_i &= 0 \quad \forall i \end{aligned}$$

Proof. Can be easily derived from the KKT conditions.

Theorem 2.12. *Consider the following optimization problem, depending on the parameter s*

$$\min_{z\in\mathbb{R}^n} \quad \frac{\|z\|^2}{2} + s d_1' z + d_2' z \tag{2.15}$$

$$Ax \le b \tag{2.16}$$

Let $s_1 \geq 0$ and $s_2 \geq s_1$. For any given s, let (z^s, u^s) satisfy the optimality conditions expressed in theorem 2.11. Let

$$Z^s \equiv \{i : A_i z_i^s - b_i = 0\}$$
$$U^s \equiv \{i : u_i^s = 0\}$$

Then, if $Z^{s_1} = Z^{s_2}$ and $U^{s_1} = U^{s_2}$, it holds

(i) For all $\theta \in [0,1]$, let $s_\theta = \theta s_1 + (1-\theta)s_2$. Then

$$z^{s_\theta} = \theta z^{s_1} + (1-\theta) z^{s_2}$$

and

$$u^{s_\theta} = \theta u^{s_1} + (1-\theta) u^{s_2}$$

(ii) For all $s \in [s_1, s_2]$, $Z^s = Z^{s_1} = Z^{s_2}$ and $U^s = U^{s_1} = U^{s_2}$.

Proof. According to theorem 2.11, the optimality conditions are

$$z + A'u + sd_1 + d_2 = 0 \tag{2.17}$$
$$u \geq 0 \tag{2.18}$$
$$Az \leq b \tag{2.19}$$
$$(A_i z - b_i)' u_i = 0 \quad \forall i \tag{2.20}$$

We will prove the first point by showing that z^{s_θ} and u^{s_θ} satisfy equations (2.17) – (2.20). The set of all (z, u, s) which satisfies equations (2.17) – (2.19) is convex, so any convex combination of feasible values (z^{s_1}, u^{s_1}, s_1) and (z^{s_2}, u^{s_2}, s_2) is feasible. We will show that also equation (2.20) holds. Suppose $A_i z_i^{s_1} - b_i = 0$. Then, since $Z^{s_1} = Z^{s_2}$, also $A_i z_i^{s_2} - b_i = 0$. Then

$$\begin{aligned} &A_i\left(\theta z_i^{s_1} + (1-\theta) z_i^{s_2}\right) - b_i = \\ &= \theta A_i z_i^{s_1} + (1-\theta) A_i z_i^{s_2} - \theta b_i - (1-\theta) b_i = \\ &= \theta\left(A_i z_i^{s_1} - b_i\right) + (1-\theta)\left(A_i z_i^{s_2} - b_i\right) = 0 \end{aligned}$$

Suppose on the other hand that $A_i z_i^{s_1} - b_i \neq 0$. Then it must hold $u_i^{s_1} = 0$ and, since $U^{s_1} = U^{s_2}$, also $u_i^{s_2} = 0$. Then

$$\theta u_i^{s_1} + (1-\theta) u_i^{s_2} = 0$$

In both cases, equation (2.20) holds. This also proves the second point of the theorem. □

Theorem 2.13. *Let $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$, with $\mathcal{C}_i \subseteq [0, \infty)$, be a finite number of convex sets that is also a partition of the set $[0, \infty)$. Let $\operatorname{int}(\mathcal{C}_i)$ be the interior of $\mathcal{C}_i$. Then there exists $m+1$ ordered values $\{s_0, \ldots, s_m\} \subset \mathbb{R} \cup \{\infty\}$, with $s_0 = 0$ and $s_m = \infty$, such that for every interval (s_{i-1}, s_i) it exists a j such that*

$$(s_{i-1}, s_i) \equiv \operatorname{int}(\mathcal{C}_j)$$

Proof. Since $\mathcal{C}_i$ is convex, its interior can be expressed as an open interval (l_i, u_i) with $l_i \leq u_i$. For any i and j, since $\mathcal{C}_i$ and $\mathcal{C}_j$ are disjoint, so are (l_i, u_i) and (l_j, u_j). Then, for every i and j, either any point of (l_i, u_i) is lesser than any point of (l_j, u_j), or the other way around. Without loss of generality, and with a slight abuse of notation, we can assume that $(l_i, u_i) < (l_{i+1}, u_{i+1})$. In order for the sets to be disjoint, it must hold $u_i \leq l_{i+1}$. We will show that it must hold $u_i = l_{i+1}$. Suppose by contradiction that $u_i < l_{i+1}$. Then it exists $x \in (u_i, l_{i+1})$. It must hold $x \in \mathcal{C}_j$ for some j. As $x > u_i$ and $x < l_{i+1}$, it must hold $j \neq i$ and $j \neq i+1$. Then we can write $(l_i, u_i) < (l_j, u_j) < (l_{i+1}, u_{i+1})$, and this implies $i < j < i+1$, a contradiction. Then $u_i = l_{i+1}$, and the theorem holds with $s_i = l_i = u_{i-1}$. □

We can now prove theorem 2.10

Theorem 2.10. *Let $\mathcal{C} \subset \mathbb{R}^n$ be a polytope, defined as $\{x : Ax \leq b\}$ for some $A \in \mathbb{R}^{n \times l}$ and $b \in \mathbb{R}^n$. Let $x \in \mathbb{R}^n$, $d \in \mathbb{R}^n$. Consider function $p : \mathbb{R} \to \mathbb{R}^n$ defined as*

$$p(s) = P_{Ax \leq b}[x - sd]$$

Then

(i) The image of p for $s \geq 0$ is a piecewise linear curve, i.e. there exist $s_1, \ldots, s_{m-1} \in \mathbb{R}$, $\alpha_1, \ldots, \alpha_m \in \mathbb{R}^n$, and $\beta_1, \ldots, \beta_m \in R^n$ such that

$$p(s) = \begin{cases} \alpha_1 s + \beta_1 & \text{if } s \in [0, s_1) \\ \alpha_2 s + \beta_2 & \text{if } s \in [s_1, s_2) \\ \vdots & \\ \alpha_{m-1} s + \beta_{m-1} & \text{if } s \in [s_{m-2}, s_{m-1}) \\ \alpha_m s + \beta_m & \text{if } s \in [s_{m-1}, \infty) \end{cases}$$

and such that for every i, $\alpha_i s_i + \beta_i = \alpha_{i+1} s_i + \beta_{i+1}$.

(ii) For every $s \geq s_{m-1}$, it holds

$$p(s) \in \arg\min_{y \in \mathcal{C}} d'(y - x)$$

Proof. We can write $P_{Ax \leq b}[x - sd]$ in the form described in theorem 2.12

$$\begin{aligned}
&P_{Ax \leq b}[x - sd] = \\
&= \arg\min_{y \in \mathcal{C}} \left\{ \frac{\|y - x\|^2}{2s} + d'(y - x) \right\} = \\
&= \arg\min_{y \in \mathcal{C}} \left\{ \frac{\|y\|^2}{2} + \frac{\|x\|^2}{2} - x'y + sd'(y - x) \right\} = \\
&= \arg\min_{y \in \mathcal{C}} \left\{ \frac{\|y\|^2}{2} + sd'y - x'y - sd'x \right\} = \\
&= \arg\min_{y \in \mathcal{C}} \left\{ \frac{\|y\|^2}{2} + sd'y - x'y \right\}
\end{aligned}$$

Consider the set $[0, \infty)$. For a given value of $(\bar{Z}, \bar{U})$, as defined in theorem 2.12, let $I^{(\bar{Z}, \bar{U})}$ be the subset of $[0, \infty)$ such that, for every $s \in I^{(\bar{Z}, \bar{U})}$, $(Z^s, U^s) = (\bar{Z}, \bar{U})$. Since the number of different values for (Z, U) is finite, there exist only a finite number of sets $I^{(Z,U)}$, and such sets are a partition of the set $[0, \infty)$. Because of the second point of theorem 2.12, such sets are convex subsets of $[0, \infty)$. Then, because of theorem 2.13, there exist an ordered set $\{s_0, s_1, \ldots, s_m\}$ with $s_0 = 0$ and $s_m = \infty$, such that for every i and $s_\alpha, s_\beta \in (s_{i-1}, s_i)$, $(Z^{s_\alpha}, U^{s_\alpha}) = (Z^{s_\beta}, U^{s_\beta})$. Because of the first point of theorem 2.12, in each interval (s_{i-1}, s_i), z^s is linear with respect to s. Then there exists $\alpha_i \in \mathbb{R}^n$ and $\beta_i \in \mathbb{R}^n$ such that, for every $s \in (s_{i-1}, s_i)$ it holds

$$z^s = \alpha_i s + \beta_i$$

Also, since the projection is a continuous function with respect to s, it must hold

$$\alpha_i s_i + \beta_i = \alpha_{i+1} s_i + \beta_{i+1}$$

Finally, it must hold $\alpha_m = 0$, otherwise we can write for every $s \in (s_{m-1}, s_m = \infty)$

$$z^s = \alpha_i s + \beta_i$$

and the feasible set $Ax \leq b$ would be unbounded. Then, for every $\hat{s} \geq s_{m-1}$

$$p(s_{m-1}) = p(\hat{s}) = \lim_{s\to\infty} p(s)$$

and theorem 2.9 implies the thesis. $\square$

2.5.2 Similarities between the optimality conditions

We will present here some results that will be used in chapter 3. Let $\mathcal{C} \subset \mathbb{R}^n$ be a convex compact set, let $f : \mathcal{C} \to \mathbb{R}$ be a smooth function. Consider problem

$$\min_{x\in\mathcal{C}} f(x) \tag{2.21}$$

We recall function (2.2), which we will rewrite here for convenience

$$\overline{FW}(x) = -\min_{y\in\mathcal{C}} (\nabla f(x)'y - \nabla f(x)'x) = \nabla f(x)'x - \min_{y\in\mathcal{C}} \nabla f(x)'y$$

and define function $\Psi : \mathbb{R}^n \to \mathbb{R}$ which returns the norm of the projected gradient direction

$$\Psi(x) = \|P_{\mathcal{C}}[x - \nabla f(x)] - x\|$$

We are assuming for the moment that $s = 1$ in the definition of Ψ but, as we will see in corollary 2.15, the reasoning holds for any value of s.

Theorem 2.14. *There exists a forcing function $\delta : \mathbb{R}_+ \to \mathbb{R}_+$ such that*

$$\delta(\Psi(x)) \leq \overline{FW}(x) \tag{2.22}$$

Proof. Consider

$$\delta(\alpha) = \min_{y\in\mathcal{C}:\Psi(y)=\alpha} \overline{FW}(y)$$

Obviously equation (2.22) holds for such a function as, for any x and $\alpha = \Psi(y)$, $y = x$ is a feasible solution of the minimization problem. To verify that δ is a forcing function, suppose there exists a sequence $\{\alpha^k\}$ such that $\lim_{k\to\infty} \delta(\alpha^k) = 0$. Then it exists a sequence $\{y^k\}$ which satisfies

$$\Psi(y^k) = \alpha^k \tag{2.23}$$

so that

$$\lim_{k\to\infty} \overline{FW}(y^k) = 0$$

Let y^* be any accumulation point of such sequence. By theorem 2.2, $\overline{FW}$ is continuous so it holds $\overline{FW}(y^*) = 0$. Then, by theorem 2.1, y^* is a first order stationary point of problem (2.21) and, because of theorem 2.7, it also holds $\Psi(y^*) = 0$. Since Ψ is continuous, it also holds

$$\lim_{k\to\infty} \Psi(y^k) = 0$$

and, by equation (2.23), we can write

$$\lim_{k\to\infty} \alpha^k = 0 \qquad \square$$

Such theorem can be easily generalized to a wide class of functions

Corollary 2.15. *Consider problem 2.21. Let $\alpha : \mathbb{R}^n \to \mathbb{R}$ and $\beta : \mathbb{R}^n \to \mathbb{R}$ be two functions such that*

(i) *α and β are continuous*

(ii) *$\alpha(x) = \beta(x) = 0$ if and only if x is a first order stationary point of problem 2.21*

Then, there exists a forcing function $\delta : \mathbb{R}_+ \to \mathbb{R}_+$ such that

$$\delta(\alpha(x)) \leq \beta(x)$$

Proof. We can follow the proof of theorem 2.14, as the only properties of Ψ and FW that are actually used in the proof are conditions *(i)* and *(ii)* of this corollary.

Then we can see that also the following theorem holds

Theorem 2.16. *There exists a forcing function $\delta : \mathbb{R}_+ \to \mathbb{R}_+$ such that*

$$\delta(\overline{FW}(x)) \leq \Psi(x) \tag{2.24}$$

Proof. Directly follows from corollary 2.15.

2.6 Rosen's Projected Gradient method

J.B. Rosen proposed an algorithm [77], similar to the Projected Gradient method, for problems with linear constraints. Given a smooth function $f : \mathbb{R} \to \mathbb{R}^n$, an m by n matrix A and a vector $b \in \mathbb{R}^n$, consider a problem expressed in the following form

$$\max_{x \in \mathbb{R}^n} f(x) \qquad (2.25)$$
$$Ax \geq b$$

Algorithm 3 describes Rosen's Projected Gradient method.

Algorithm 3: Rosen's Projected Gradient

```
   Data: A feasible point $x^0$
1  $k \leftarrow 0$;
2  repeat
3      $J^k \leftarrow \{j : A'_j x_j = b_j\}$;
4      $r^k \leftarrow \left(A'_{J^k} A_{J^k}\right)^{-1} A^{-1}_{J^k} \nabla_{J^k} f(x^k)$;
5      $m^k \leftarrow \max_i\{r^k_i\}$;
6      $P^k \leftarrow I - A_{J^k}\left(A'_{J^k} A_{J^k}\right)^{-1} A^{-1}_{J^k}$;
7      if $P^k \nabla f(x^k) = 0$ and $r^k_{m^k} \leq 0$ then
8          STOP;
9      end
10     if $\left\|P^k \nabla f(x^k)\right\| > c r^k_{m^k}$ then
11         $d^k \leftarrow P^k \nabla f(x^k)$;
12     else
13         $\hat{J}^k \leftarrow J^k \setminus m^k$;
14         $\hat{P}^k \leftarrow I - A_{\hat{J}^k}\left(A'_{\hat{J}^k} A_{\hat{J}^k}\right)^{-1} A^{-1}_{\hat{J}^k}$;
15         $d^k \leftarrow \hat{P}^k \nabla f(x^k)$;
16     end
17     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
18     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
19     $k \leftarrow k + 1$;
20 until STOP instruction not reached ;
```

Matrix P^k computes the projection over the hyperspace defined by $A'_j x_j = b_j$ for every $j \in J^k$. This projection is applied to the gradient direction, unless by line 10 the gradient is, in some way, strongly oriented towards the interior of the feasible set. In that case, we project over a wider hyperspace by using $\hat{P}^k$, which does not restrict variable m^k. Finally, at every iteration, an Armijo linesearch is used to compute a feasible step along direction d^k.

In [77], Rosen presents a convergence proof of the algorithm, to a maximizer of $f(x)$, under the assumption that such function is concave. However there is a subtle error in such proof. On page 197, the value δ is defined such that $\lambda_i(x^0) \geq \delta > 0$. δ is a step length that the algorithm is able to take at every iteration without violating feasibility, and it is used in equation (4.36), which states

$$\psi^k = \max\left\{\min\left[\frac{\left\|Pg^k\right\|^2}{4\gamma}, \frac{\delta\left\|Pg^k\right\|}{2}\right], \min\left[\frac{(\rho^k)^2}{16\gamma\gamma_v}, \frac{\delta\rho^k}{4\sqrt{\gamma_v}}\right]\right\}$$

Rosen then proves that $\lim_{k\to\infty}\rho^k = 0$, from which he deduces that also $\lim_{k\to\infty}\left\|Pg^k\right\| = 0$ and $\lim_{k\to\infty}\rho^k = 0$, which are used to imply the convergence of the algorithm. However the value δ is not guaranteed to exist, as the sequence $\lambda_i(x^k)$ could converge to zero as k goes to infinity. Then, to be formally correct, we should use δ^k instead of δ, because this value can only be defined relative to iteration k. Equation (4.36) should be written as

$$\psi^k = \max\left\{\min\left[\frac{\left\|Pg^k\right\|^2}{4\gamma}, \frac{\delta^k\left\|Pg^k\right\|}{2}\right], \min\left[\frac{(\rho^k)^2}{16\gamma\gamma_v}, \frac{\delta^k\rho^k}{4\sqrt{\gamma_v}}\right]\right\}$$

from which we can only deduce that either $\lim_{k\to\infty}\left\|Pg^k\right\| = 0$ and $\lim_{k\to\infty}\rho^k = 0$, or $\lim_{k\to\infty}\delta^k = 0$, and this prevents us from proving the convergence. If the values δ^k converge to zero, it means that at every iteration, the maximum feasible step that can be taken is too small to guarantee a large enough reduction of the objective function.

Years later, Du and Zhang proved [31] the convergence of the Rosen algorithm to a first order stationary point, without using the concavity assumption.

If the problem structure allows it, Rosen's method can naturally lead to a column generation strategy. In fact, many of the algorithms that we will examine in chapter 4 strongly resemble Rosen's algorithm, the main difference being that a new variable is potentially added in the set of free variables at every iteration, while in algorithm 3 this does not always happen. In particular, it is like if the parameter c in line 10 is so high that the comparison is always false, so lines 13–15 are executed at every iteration.

Chapter 3

Inexact decomposition methods for constrained optimization

Decomposition is the process of solving a large optimization problem by sequentially solving many smaller size problems, obtained by fixing at every iteration a subset of the variables and allowing the solver to modify only the unfixed variables, which we will call the working set.

Subproblems generated by a decomposition algorithm can be solved either exactly or inexactly. In this thesis, we will consider decomposition methods where the subproblems are solved inexactly, meaning that usually only one iteration of the solving algorithm is performed.

The main difference between the various decomposition strategies in inexact decomposition algorithms lies in the way the working set is selected at every iteration. In general this strongly depends on the structure of the problem, and often also on the current point.

Strategies can be divided into two main groups

- In Gauss-Seidel based decomposition algorithms, the order in which the variables are selected is predetermined, and does not depend on the current point.
- In Gauss-Southwell based decomposition algorithms, the working set is recomputed at every iteration, and depends on the current point. In general the selected variables are the ones that, in some way, mostly violate some optimality condition.

Many algorithms based on those ideas have been studied. Patriksson [71] proposes some convergent algorithms for parallel algorithms based on the Gauss-Seidel exact decomposition method. Grippo et al. [42] analyzed the convergence properties of unconstrained Gauss-Seidel decomposition. In [82] Tseng proposes an Gauss-Seidel algorithm for constrained optimization. Bonettini et al. [12] used an inexact Gauss-Seidel algorithm to solve problems related to image processing. Grippo et al. [43] reviewed some exact decomposition methods. Chen et al. [19] proposed an application to decomposition methods for variational inequalities, which was later extended by Tseng [83] and Solodov [80]. Luo [60] studied the convergence of some Gauss-Southwell approaches for convex problems.

3.1 Separable feasible set

The problem structure determines the decomposition algorithms that can and cannot be used. One of the simplest case happens when the feasible set can be expressed as a Cartesian product of other sets. Let $\mathcal{X}_1 \subseteq \mathbb{R}^{n_1}, \mathcal{X}_2 \subseteq \mathbb{R}^{n_2}, \dots, \mathcal{X}_m \subseteq \mathbb{R}^{n_m}$, let $n = \sum_{i=0}^{m} n_i$, and let the feasible set $\mathcal{X} \in \mathbb{R}^n$ be defined as

$$\mathcal{X} \equiv \prod_{i=1}^{m} \mathcal{X}_i$$

Let $f : \mathbb{R}^n \to \mathbb{R}$, and consider the problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & f(x) \\ & x \in \mathcal{X} \end{aligned} \tag{3.1}$$

Instead of solving directly problem 3.1 an algorithm can solve, at every iteration k only a subproblem in a space which is a subset of $\{\mathcal{X}_1, \dots, \mathcal{X}_m\}$, which we will call a working set and denote by W^k. The general scheme is described in algorithm 4.

In this section, we will describe some common choices for lines 3 and 4. We will refer to the literature for the cases already analyzed by other authors, and we will give a convergence proof for the cases in which, to our knowledge, no such proof have been formulated before.

The possible choices we will consider are, for line 3

a) $W^k = k \operatorname{mod}^{+1} n$

b) $W^k = \arg\min_i \left\| P_{\mathcal{X}_i}[{x^k}_{(i)} - \nabla f(x^k)_{(i)}] - {x^k}_{(i)} \right\|$

Algorithm 4: General decomposition scheme
```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     choose $W^k$, possibly depending on $k$ and $x^k$;
4     compute, using some optimization procedure, a feasible point $x^{k+1}$
      such that ${x^k}_{(i)} = {x^{k+1}}_{(i)}$ for any $i \notin W^k$;
5     $k \leftarrow k + 1$;
6 end
```

c) $W^k = \arg\min_i \left\| FW_{\mathcal{X}_i}({x^k}_{(i)} - \nabla f(x^k)_{(i)}) - {x^k}_{(i)} \right\|$

Condition a) is a Gauss-Seidel rule, the blocks are selected sequentially and the choice does not depend on x^k. Condition b), a la Gauss-Southwell, selects the block that mostly violates an optimality condition given by the norm of the projected gradient direction, as such value is zero if and only if x^k is a first order stationary point. Similarly, condition c) uses a Frank-Wolfe rule for selecting, a la Gauss-Southwell, the most violating block.

The possible choices for line 4 are

a) A single step of the Projected Gradient method.

b) A single step of the Frank-Wolfe method.

3.1.1 Projected Gradient method with Gauss-Seidel selection rule

We are solving problem 3.1 using algorithm 5.

Algorithm 5: Projected Gradient with Gauss-Seidel decomposition
```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     $W^k \leftarrow k \bmod^{+1} m$;
4     $\hat{x}^k \leftarrow P_{\mathcal{X}_{W^k}}[{x^k}_{(W^k)} - s^k \nabla f(x^k)_{(W^k)}]$;
5     $d^k \leftarrow \left(\hat{x}^k - {x^k}_{(i)}\right)_{(-i)}$;
6     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
7     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8     $k \leftarrow k + 1$;
9 end
```

This algorithm is a special case of the Scaled Gradient Projection Method described in [12], where the scaling matrix D_k is s^k times the identity. The convergence of the algorithm to a first order stationary point is already provided in [12, Chapter 2.3], by using a scaling matrix equal to $s^k I$.

3.1.2 Projected Gradient method with Gauss-Southwell selection rule

The Gauss-Southwell approach requires a function that in some way captures the distance to optimality for a given block of variables. If we intend to use the Projected Gradient method to produce a descent direction, since in a first order stationary point it holds for every i

$$P_{\mathcal{X}_i}[x_{(i)} - s^k \nabla f(x)_{(i)}] = x_{(i)}$$

a reasonable choice is to define the set of functions $\Psi_i : \mathcal{X}_i \to \mathbb{R}$ as

$$\Psi_i(x_{(i)}) = \left\| P_{\mathcal{X}_i}[x_{(i)} - s^k \nabla f(x)_{(i)}] - x_{(i)} \right\|$$

which is equal to zero if and only if $x_{(i)}$ is stationary in the ith subproblem. Let us then consider algorithm 6.

Algorithm 6: Projected Gradient with Gauss-Southwell decomposition

```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     $W^k \leftarrow \arg\max_{i \in \{1,\dots,m\}} \Psi_i(x_{(i)})$;
4     $\hat{x}^k \leftarrow P_{\mathcal{X}_{W^k}}[{x^k}_{(W^k)} - s^k \nabla f(x^k)_{(W^k)}]$;
5     $d^k \leftarrow \left(\hat{x}^k - {x^k}_{(i)}\right)_{(-i)}$;
6     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
7     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8     $k \leftarrow k + 1$;
9 end
```

The following convergence result can be derived

Theorem 3.1. *Any accumulation point of sequence $\{x^k\}$, generated by algorithm 6, is a first order stationary point of problem 3.1.*

Proof. Since $\mathcal{X}$ is convex, by construction for every k, $\hat{x}^k \in \mathcal{X}$ and $x^k \in \mathcal{X}$. Then, because of theorem 2.5 it holds for every k

$$\left(x^k - \nabla f(x^k)_{(W^k)} - \hat{x}^k\right)' \left(x^k - \hat{x}^k\right) \leq 0$$

which can be written as

$$\nabla f(x^k)_{(W^k)}{}' \left(x^k - \hat{x}^k\right) \leq -\frac{1}{s^k}\left\|x^k - \left(x^k - \hat{x}^k\right)\right\|^2$$

or

$$\nabla f(x^k)' d^k \leq -\frac{1}{s^k}\left\|x^k - \hat{x}^k\right\|^2 \tag{3.2}$$

By point *(ii)* of theorem A.2, since d^k is bounded, we can write

$$\lim_{k\to\infty} \nabla f(x^k)' d^k = 0 \tag{3.3}$$

Then, using equation (3.2) and the boundness of sequence $\{s^k\}$, we can write

$$\lim_{k\to\infty} \left\|x^k - \hat{x}^k\right\| = 0 \tag{3.4}$$

Let $\bar{x}$ and $K \subseteq \mathbb{N}$ be a point and a set of indexes such that

$$\lim_{k\to\infty, k\in K} x^k = \bar{x} \tag{3.5}$$

Let $\overline{W}$ be an index such that the set $\{k \in K : W^k = \overline{W}\}$ is infinite. We can write

$$\lim_{k\to\infty, k\in K} \left\|{x^k}_{(\overline{W})} - P_{\mathcal{X}_{\overline{W}}}[{x^k}_{(\overline{W})} - s^k \nabla f(x^k)_{(\overline{W})}]\right\| = 0 \tag{3.6}$$

By the definition of $\overline{W}$ we can also write, for every i

$$\lim_{k\to\infty, k\in K} \left\|{x^k}_{(i)} - P_{\mathcal{X}_i}[{x^k}_{(i)} - s^k \nabla f(x^k)_{(i)}]\right\| = 0$$

Since such function is continuous, both with respect to x^k and to s^k, we obtain for every i

$$\left\|\bar{x}_{(i)} - P_{\mathcal{X}_i}[\bar{x}_{(i)} - \bar{s} \nabla f(\bar{x})_{(i)}]\right\| = 0$$

where $\bar{s} > 0$ is an accumulation point of s^k. Such equation is a first order optimality condition. $\square$

By examining the proof, we must remark that function Ψ_i is not the only possible choice for line 3 of algorithm 6. In fact any family of functions that satisfies condition 3.2 can be used.

Condition 3.2. *For every i, let $\Psi_i : \mathcal{X}_i \to \mathbb{R}$ be a function such that*

1. *Ψ_i is continuous.*
2. *$\Psi_i(x) = 0$ if and only if x is a first order stationary point of problem 3.1.*

3. *There exists a forcing function δ such that, for every $x \in \mathcal{X}$ and (bounded) s*

$$\left\| x - P_{\mathcal{X}_i}[x_{(i)} - s\nabla f(x)_{(i)}] \right\| \geq \delta(\Psi_i(x))$$

We can easily verify that the proof still holds, as the forcing function allows to infer equation (3.6) from equation (3.4), and the rest of the proof naturally follows. This means that it is possible to use, for example, a Frank-Wolfe like block selection criterion by defining

$$\Psi_i(x) = -\min_{y \in \mathcal{X}_i} \nabla f(x)'_{(i)}(y - x_{(i)})$$

Theorems 2.2, 2.1, and 2.14 prove that such function satisfies condition 3.2.

3.1.3 Frank-Wolfe method with Gauss-Seidel selection rule

One of the major differences between using the Projected Gradient and using the Frank-Wolfe methods to compute a descent direction lies in the linesearch type that must be used. As we have seen in sections 3.1.1 and 3.1.2, the Armijo linesearch fits naturally with the Projected Gradient algorithm, and can be used to show its convergence properties. On the other hand, using such linesearch with the Frank-Wolfe direction is more problematic as there is no easy way, using only that assumption, to prove that the step goes to zero. Then a different linesearch must be used, called quadratic linesearch and described in the appendix, section A.2.

Let us consider algorithm 7, which is similar to algorithm 5 but uses the Frank-Wolfe algorithm to generate a descent direction.

```
Algorithm 7: Frank-Wolfe with Gauss-Seidel decomposition
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     $W^k \leftarrow k \bmod^{+1} m$;
4     $\hat{x}^k \leftarrow FW_{\mathcal{X}_{W^k}}({x^k}_{(W^k)})$;
5     $d^k \leftarrow \left(\hat{x}^k - {x^k}_{(i)}\right)_{(-i)}$;
6     $\alpha^k \leftarrow$ quadratic linesearch along $d^k$;
7     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8     $k \leftarrow k + 1$;
9 end
```

Theorem 3.3. *Let $\{x^k\}$ be an infinite sequence of points generated by algorithm 7. Let $\bar{x}$ be an accumulation point of such sequence. Then $\bar{x}$ is a first order* *stationary point of problem 3.1.*

Proof. Let $K \subseteq \mathbb{N}$ be such that

$$\lim_{k\to\infty, k\in K} x^k = \bar{x} \tag{3.7}$$

Let $K_2 \subseteq \mathbb{N}$ be set that contains K and, for every element in K, all the indexes located at most after $m-1$ steps. Formally

$$K_2 \equiv \{\bar{k} : \exists\, k \in K, i \in \{0, \dots, m-1\} \text{ such that } \bar{k} = k + i\}$$

By theorem A.2 and equation (3.7) it holds

$$\lim_{k\to\infty, k\in K_2} x^k = \bar{x}$$

Then by theorem A.4 it must hold, for every i

$$\lim_{k\to\infty, k\in K_2} \nabla f(x^k)' {d^k}_{(i)} = 0$$

which can be written as

$$\lim_{k\to\infty, k\in K_2} \nabla f_{(i)}(x^k)'(FW_{\mathcal{X}_i}({x^k}_{(i)}) - {x^k}_{(i)}) = 0$$

By theorem 2.3, such function is continuous so, for every i

$$\nabla f_{(i)}(\bar{x})'(FW_{\mathcal{X}_i}({x^k}_{(i)}) - \bar{x}_{(i)}) = 0$$

which is equivalent to the optimality condition

$$\nabla f(\bar{x})'(FW_{\mathcal{X}_i}(\bar{x}) - \bar{x}) = 0$$ □

3.1.4 Frank-Wolfe method with Gauss-Southwell selection rule

Algorithm 8, presented below, is similar to the one in section 3.1.2, with the difference that the Frank-Wolfe algorithm is used to determine the block that mostly violates the optimality condition, and to compute the descent direction.

We will prove the following theorem

Theorem 3.4. *Any accumulation point of sequence $\{x^k\}$, generated by algorithm 8, is a first order stationary point of problem 3.1.*

Algorithm 8: Frank-Wolfe with Gauss-Southwell decomposition

```
  **Data**: A feasible point $x^1$
1 $k \leftarrow 1$;
2 **while** *stopping criterion not satisfied* **do**
3     $W^k \leftarrow \arg\max_{i\in\{1,\dots,m\}} \overline{FW}_{\mathcal{X}_i}(x_{(i)})$;
4     $\hat{x}^k \leftarrow FW_{\mathcal{X}_{W^k}}(x^k{}_{(W^k)})$;
5     $d^k \leftarrow \left(\hat{x}^k - x^k{}_{(i)}\right)_{(-i)}$;
6     $\alpha^k \leftarrow$ quadratic linesearch along $d^k$;
7     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8     $k \leftarrow k+1$;
9 **end**
```

Proof. Let $K \subseteq \mathbb{N}$ be such that

$$\lim_{k\to\infty, k\in K} x^k = \bar{x}$$

By theorem A.4 it must hold

$$\lim_{k\to\infty, k\in K} -\nabla f(x^k)' d^k{}_{(W^k)} = \overline{FW}_{W^k}(x^k{}_{(i)}) = 0 \tag{3.8}$$

From the definition of W^k it must hold for every i

$$\overline{FW}_{\mathcal{X}_i}(x^k{}_{(i)}) \leq \overline{FW}_{W^k}(x^k{}_{(i)})$$

so we can write, for every i

$$\lim_{k\to\infty, k\in K} = \overline{FW}_{\mathcal{X}_i}(x^k{}_{(i)}) = 0$$

Theorem 2.2 states the continuity of $\overline{FW}$ so

$$\overline{FW}_{\mathcal{X}_i}(\bar{x}_{(i)}) = 0$$

which is equivalent to the optimality condition

$$\overline{FW}(\bar{x}) = 0$$ □

Here the quadratic linesearch is not the only available option for line 6 of the algorithm. If the Armijo linesearch is used instead, equation (3.8) still holds, and the rest of the proof follows. We can easily change the linesearch because, contrary to algorithm 7, we do not require the result of theorem A.2 to prove convergence.

3.2 Feasible set with box and one linear equality constraints

Let us consider a problem with box and one linear equality constraint. Such kind of problem arises when training Support Vector Machines (SVM) [13] [57] and in the reformulation of the network equilibrium problem that we will examine in chapter 4. So, let us write

$$\begin{aligned} &\min_{y\in\mathbb{R}^n} f(y) \\ &\quad a'y = b \\ &\quad l \le y \le u \end{aligned}$$

Here, $a \in \mathbb{R}^n$, $b \in \mathbb{R}$ and $l, u \in \mathbb{R}^n$. We can transform this problem in an equivalent problem, by introducing variables $x_i = a_i y_i$, and appropriately changing $f(y)$, l and u.

$$\begin{aligned} &\min_{y\in\mathbb{R}^n} f(x) \\ &\quad e'x = b \\ &\quad l \le x \le u \end{aligned} \tag{3.9}$$

So, without loss of generality, we can assume that $a_i = 1$ and analyze problem 3.9. Given a feasible point x, let $w(x)$ be the set of i such that x_i can be increased while maintaining feasibility

$$w(x) = \{i : x_i < u_i\}$$

similarly, let $W(x)$ be the set of i such that x_i can be decreased

$$W(x) = \{i : x_i > l_i\}$$

Let, at every iteration k

$$m^k \in \arg\min_{i\in w(x^k)} \nabla f_i(x^k)$$

$$M^k \in \arg\max_{i\in W(x^k)} \nabla f_i(x^k)$$

with ties being broken arbitrarily. The pair of indexes (m^k, M^k) is in some way a maximum violating pair, i.e. the pair of variables that is mostly away from optimality. Let $D_+^k, D_-^k \subset \{1, \dots, n\}$ such that for every k

(i) $D_+^k \cap D_-^k = \emptyset$

(ii) $\left|D_+^k\right| = \left|D_-^k\right|$

(iii) $m^k \in D_+^k$ and $M^k \in D_-^k$

(iv) $\sum_{p \in D_+^k} \nabla f_p(x^k) - \sum_{p \in D_-^k} \nabla f_p(x^k) < 0$

We want D_+^k and D_-^k to contain the indexes of the variables that will actually be increased and decreased at iteration k. Let

$$s^k = \min \left\{ \min_{i \in D_+^k} \left(u_i - x_i^k\right), \min_{i \in D_-^k} \left(x_i^k - l_i\right) \right\}$$

Then we define d^k as

$$d_i^k = \begin{cases} s^k & \texttt{for} \quad i \in D_+^k \\ -s^k & \texttt{for} \quad i \in D_-^k \\ 0 & \texttt{otherwise} \end{cases} \tag{3.10}$$

We can see that s^k is the maximum length that x^k can be moved along the chosen direction without becoming unfeasible. d^k is computed using s^k, such that the unitary step can always be taken, and in fact it is the maximum allowed step. Note that condition *(ii)* and the rule to compute s^k ensure that $x^k + d^k$ is always feasible. Condition *(iv)* forces d^k to be a descent direction.

Consider algorithm 9. In order to prove the convergence, we must make an

```
Algorithm 9: Decomposition with a linear constraint
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     compute $d^k$ with equation (3.10);
4     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$ with starting step $\Delta^k = 1$;
5     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
6     $k \leftarrow k + 1$;
7 end
```

additional technical assumption. Although we do not believe that such assumption is necessary for the algorithm to converge, we have been unable to prove it without. Note that algorithm 9 is a generalization of algorithm I.2–SVM$^{\text{light}}$ in [57, p. 1289] and, as stated in the paper, it is an open problem whether such algorithm converges.

Assumption 3.5. *The vectors x^1, l, u belong to $\mathbb{Q}^n$. Also in algorithm 18 (the Armijo linesearch algorithm), $\delta \in \mathbb{Q}$.*

Under such assumption, theorem 3.11 gives a convergence result for algorithm 9. Some additional results will be needed in order to prove theorem 3.11

Theorem 3.6. *For all k, $x^k \in \mathbb{Q}^n$.*

Proof. It is easy to see that, if $x^k \in \mathbb{Q}^n$, then $x^{k+1} \in \mathbb{Q}^n$, as by construction, since $x^k, l, u \in \mathbb{Q}^n$, then also $s^k \in \mathbb{Q}$, which implies $d^k \in \mathbb{Q}^n$. By the Armijo linesearch, we have $\alpha^k = \delta^j$ for some $j \in \mathbb{N}$, so also $\alpha^k \in \mathbb{Q}$, so it must hold $x^{k+1} \in \mathbb{Q}^n$. Then the theorem follows from induction on the assumption that $x^1 \in \mathbb{Q}$. □

Theorem 3.7. *Let $\{x^k\}$ be the sequence of points generated by the algorithm. Suppose such sequence is infinite. Let K be the set of indexes such that the full Armijo step is not taken, i.e. such that $\Delta^k \neq 1$. Then K has infinite cardinality.*

Proof. Consider a generic iteration k. Let $m \in \mathbb{Q}$ be a number such that, for every i, $x_i^k - l_i$ and $u_i - x_i^k$ are equal to a multiple of m. By theorem 3.6 such values are rational, so m exists.
If a full Armijo step is taken at iteration k, then every x_i^k is either left unchanged or it is changed by s^k, which is by construction a multiple of m. Then also the values $x_i^{k+1} - l_i$ and $u_i - x_i^{k+1}$ are a multiple of m.
By induction, if a full Armijo step is taken at iteration $k+1$, then also $x_i^{k+2} - l_i$ and $u_i - x_i^{k+2}$ are multiple of m, and so on.
However the number of different x^k such that every x_i^k is a multiple of m and is bounded by l_i and u_i is finite. Then, since by the Armijo rule all the x^k must be different, there cannot be an infinite number of consecutive iterations such that a full Armijo step is taken. □

Theorem 3.8. *Let $\{x^k\}$ be the sequence of points generated by the algorithm. Then there exists a $\bar{k}$ such that, for every $k \geq \bar{k}$ there exists an index p^k such that $p^k \in w(x^k)$ and $p^k \in W(x^k)$.*

Proof. For every iteration k such that $w(x^k) \cap W(x^k) = \emptyset$, every component x_i^k is either equal to l_i or to u_i. Only a finite number of points exists with such property, and the Armijo linesearch prevents each point from appearing more than once in the sequence, so the value $\bar{k}$ must exist. □

Theorem 3.9. *There exists an infinite set K such that*

$$\lim_{k\to\infty, k\in K} \nabla f_{m^k}(x^k) - \nabla f_{M^k}(x^k) = 0 \tag{3.11}$$

Proof. By theorem 3.7, there must exist an infinite set K such that for every $k \in K$ the full Armijo step is never taken. Consider theorem A.2. All the hypothesis of the theorem are satisfied by the algorithm, except for hypothesis *(c)*. However such assumption is only used to write equation (A.5), which is used to prove that there must exist an infinite number of iterations in which the full Armijo step is not taken. However we already proved that in theorem 3.7, so theorem A.2 holds even without this assumption. Then

$$\lim_{k\to\infty, k\in K} \frac{\nabla f(x^k)' d^k}{\|d^k\|} = 0 \tag{3.12}$$

For every $k \in K$

$$\frac{\nabla f(x^k)' d^k}{\|d^k\|} = \frac{\displaystyle\sum_{p\in D_+^k} \nabla f_p(x^k) - \sum_{p\in D_-^k} \nabla f_p(x^k)}{\sqrt{|D_+^k| + |D_-^k|}}$$

by definition of m^k and M^k, it holds

$$\nabla f_{m^k}(x^k) \le \frac{1}{|D_+^k|} \sum_{p\in D_+^k} \nabla f_p(x^k) \qquad \nabla f_{M^k}(x^k) \ge \frac{1}{|D_-^k|} \sum_{p\in D_-^k} \nabla f_p(x^k)$$

then, since $m^k \in D_+^k$ and $M^k \in D_-^k$, and recalling that $|D_+^k| = |D_-^k| \le \frac{n}{2}$

$$\frac{\displaystyle\sum_{p\in D_+^k} \nabla f_p(x^k) - \sum_{p\in D_-^k} \nabla f_p(x^k)}{\sqrt{2|D_+^k|}} \le \frac{\nabla f_{m^k}(x^k) - \nabla f_{M^k}(x^k)}{|D_+^k|\sqrt{2|D_+^k|}} \le \frac{\nabla f_{m^k}(x^k) - \nabla f_{M^k}(x^k)}{\frac{n}{2}\sqrt{n}}$$

so equation (3.12) implies the thesis. □

Theorem 3.10. *There exists a sequence $K \subseteq \mathbb{N}$ and a point $x^* \in \mathbb{R}^n$ such that*

$$\lim_{k\to\infty, k\in K} x^k = x^*$$

and such that x^* satisfies a first order optimality condition.

Proof. Consider a sequence that satisfies the thesis of theorem 3.9, and let us extract a subsequence K_2 that converges to x^*. For k big enough, it must hold $W(x^k) \supseteq W(x^*)$ and $w(x^k) \supseteq w(x^*)$. Using the p^k defined by theorem 3.8 we can write, for k big enough

$$\nabla f_{M^k}(x^k) = \max_{i\in W(x^k)} \nabla f_i(x^k) \ge \max_{i\in W(x^*)} \nabla f_i(x^k) \ge \nabla f_{p^k}(x^k) \ge$$

$$\geq \min_{i \in w(x^*)} \nabla f_i(x^k) \geq \min_{i \in w(x^k)} \nabla f_i(x^k) = \nabla f_{m^k}(x^k)$$

By equation (3.12) and the squeeze theorem

$$\lim_{k \to \infty, k \in K_2} \max_{i \in W(x^*)} \nabla f_i(x^k) = \min_{i \in w(x^*)} \nabla f_i(x^k)$$

and, since $f \in C^1$

$$\max_{i \in W(x^*)} \nabla f_i(x^*) = \min_{i \in w(x^*)} \nabla f_i(x^*)$$

Then, for every $i \in W(x^*)$, $j \in w(x^*)$

$$\nabla f_i(x^*) \leq \nabla f_j(x^*)$$

which is a first order optimality condition. □

Theorem 3.11. *If f is convex, then every limit point of sequence $\{x^k\}$ solves the problem.*

Proof. By theorem 3.10 there exists one accumulation point x^* which satisfies a first order optimality condition. Since the Armijo linesearch enforces a strict decrease of the objective function value, any other accumulation point of the sequence must have the same objective function value of x^*, and then be a global minimizer of f. □

Chapter 4

Decomposition methods for network equilibrium problems

Network assignment problems are a widely studied subject in many research fields, as for instance transportation and data transmission. The aim of a network equilibrium model is to predict the link flows of a network which depend on the routes origin/destination chosen by the users (travellers or data package) of the network.

The network is modeled by a direct graph, whose nodes represent origins, destinations, and intersections, and arcs represent the transportation links. There is a set of node pairs, called Origin/Destination (OD) pair, and for each OD pair there is a known demand. For each link there is a user cost function depending, in general, on the arc flows of the whole graph. Every infinitesimal unit of flow travels from its origin to its destination along a path that minimizes its own travel cost, so the network will eventually reach stability in a Nash equilibrium point. The Wardrop's user-optimal principle states that, if the network is in equilibrium, then all the routes used by the users have a cost less or equal to that of any unused route. The Wardrop equilibrium conditions lead to solve a variational inequality which, under suitable assumptions on the cost functions, is equivalent to a convex optimization problem. For instance, the equivalence between the variational inequality and the convex optimization problem holds whenever the cost f_a of any given link a is non-decreasing and depends only on the flow y_a through the given link. We will refer to [34, 36, 72, 4] for the technical details, for the assumptions which lead to the convex optimization problems object of this work (see below), and for the wide literature on the topic.

Let P be the number of OD pairs, n the number of paths between all the origins and all the destinations, and n_p the number of paths between the origin and destination of the p-th OD pair, so that we have $n = n_1 + n_2 + \ldots + n_P$. We will not consider the cyclic paths, as they cannot appear in any optimal solution. We denote by $x \in \mathbb{R}^n$ the vector of path flows.

We will denote by bracketed subscripts the subvectors. For example with $x_{(h)}, h \in \{1, \ldots, P\}$ we will denote the h-th subvector of x. In the case of the network equilibrium problem, it will corrispond to the h-th OD pair variables.

We partition the vector of variables x as follows

$$x = \big(x_{(1)}, \ldots, x_{(p)}, \ldots, x_{(P)}\big)$$

where $x_{(p)} \in \mathbb{R}^{n_p}$, $p \in \{1, \ldots, P\}$. We introduce a convex minimization problem as formulation of symmetric network equilibrium problems. We remark that real network equilibrium problems are very large scale problems, and this is the main issue to be considered in the design of optimization algorithms. Under suitable assumptions, the optimization problem (object of this work) takes the structure

$$\begin{aligned} \min_x \quad & f(x) \\ & e'x_{(p)} = d_p \quad \forall p \in \{1, \ldots, P\} \\ & x_{(p)} \geq 0 \qquad \forall p \in \{1, \ldots, P\} \end{aligned} \tag{4.1}$$

where $f : R^n \to R$ is a convex continuously differentiable function; e is a vector of ones; d_p is the demand of the p-th OD pair.

An arc-based reformulation of problem (4.1) can be used under the assumption of additive path costs. Let m be the number of arcs of the graph. We denote by $y \in \mathbb{R}^m$ the arc flow vector. Arc and path flows are related by $y = Hx$, where $H \in \mathbb{R}^{m \times n}$ is the arc-path incidence matrix whose generic element h_{ij} is equal to 1 if arc i belongs to path j and is equal to 0 otherwise. Under the assumption of additive path costs, the optimization problem with arc variables takes the structure

$$\begin{aligned}
\min_{x,y} \quad & F(y) \\
& y = Hx \\
& e'x_{(p)} = d_p \quad \forall p \in \{1, \dots, P\} \\
& x_{(p)} \geq 0 \qquad \forall p \in \{1, \dots, P\}
\end{aligned} \tag{4.2}$$

where $F : R^m \to R$ is separable, i.e., $F(y) = \sum_{i=1}^m F_i(y_i)$, and each $F_i : \mathbb{R}^+ \to \mathbb{R}^+$, is a convex continuously differentiable function.

Arc-based algorithms using formulation (4.2), like the well-known Frank-Wolfe [35] method, are widely used since they require to store only arc flows. Recently, new arc flow-based algorithms have been proposed in [4, 30, 67, 5] and show impressive performance in terms of speed of convergence. However, when the classic additivity assumption on the path costs does not hold, it is necessary (see [36, 72]) to use the more general path formulation (4.1) and, as consequence, algorithms employing explicit path information, so that the efficient arc-based algorithms cited above cannot be adopted.

Then, for sake of generality, in this work we consider the general class of path-based optimization problems defined in (4.1).

The convex problem (4.1) has a very simple structure, as its feasible set $\mathcal{F}$ is the Cartesian product of simplices. However, problem (4.1) can be considered a "virtual" formulation: indeed, in any real application it is not reasonable to completely enumerate, a priori, all the paths, since this would be too expensive.

In order to take into account the difficulties related to the large dimension of problem (4.1) we adopt a standard column generation strategy (by iteratively adding only the variables, i.e., the paths, of the model needed to reach optimality) and we employ a gradient projection method within an inexact decomposition framework. Similar approaches have been proposed in [69] for solving the asymmetric traffic equilibrium problem formulated as a variational inequality, and in [33, 52] for solving the symmetric traffic equilibrium problem formulated as (4.1). In particular, in [33] an adaptation of the Rosen's Projected Gradient algorithm is used within a theoretically exact decomposition Gauss-Seidel scheme.

The aims of the work are mainly to study convergent inexact decomposition methods for problems defined on the Cartesian product of convex sets, and to derive specific algorithms for problems of the form (4.1).

The chapter is organized as follows: in section 4.1, with reference to a general problem defined on the Cartesian product of convex sets, we present an inexact decomposition algorithm using restricted feasible sets (belonging to lower dimensional spaces to possibly tackle large dimensional problems), and gradient projection iterations. Under suitable assumptions on the restricted feasible sets, we prove the global convergence of the presented algorithm. Due to the generality of the assumptions stated, the proposed decomposition algorithm can be the basic framework even to develop algorithms for classes of problems different from that object of the present work. The specific case of traffic equilibrium problem is the topic of section 4.2, in which different decomposition schemes are derived from the general framework previously defined. Computational results are presented in section 4.3.

4.1 Inexact decomposition algorithm using restricted feasible sets

Let us consider the problem

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x) \\ &x \in \mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \times \ldots \times \mathcal{F}_L \end{aligned} \tag{4.3}$$

where $f : R^n \to R$ is a convex continuously differentiable function, $\mathcal{F}_h \subseteq \mathbb{R}^{n_h}$, $h = 1, \ldots, L$, are compact convex sets, and $n_1 + \ldots + n_h + \ldots + n_L = n$. Taking into account the structure of the feasible set $\mathcal{F}$, we partition the vector of variables x as follows

$$x = \big(x_{(1)}, \ldots, x_{(h)}, \ldots, x_{(L)}\big)$$

where $x_{(h)} \in \mathbb{R}^{n_h}$, $h \in \{1, \ldots, L\}$.

Given a point $y \in \mathcal{F}$, for $h \in \{1, \ldots, L\}$, we denote by $\mathcal{F}_h(y)$ a closed convex set depending on the point y and such that $\mathcal{F}_h(y) \subseteq F_h$. Formally, we state the following assumption on the sets $F_h(\cdot)$ for $h \in \{1, \ldots, L\}$.

Assumption 4.1.

(i) *There is only a finite set of possible values for $\mathcal{F}_h(x)$, and such set does not depend on x.*

(ii) *Let $K \subseteq \{0, 1, \ldots, \}$ be an infinite subsequence such that $x^k \in \mathcal{F}$ for all $k \in K$, and assume that $x^k \to \bar{x}$ for $k \in K$ and $k \to \infty$. If for $k \in K$ and* *k sufficiently large we have that*

$$\bar{x}_{(h)} \in \arg \min_{x_{(h)} \in \mathcal{F}_h(x^k)} f(\bar{x}_{(1)}, \ldots x_{(h)}, \ldots, \bar{x}_{(L)})$$

then it holds

$$\bar{x}_{(h)} \in \arg \min_{x_{(h)} \in \mathcal{F}_h} f(\bar{x}_{(1)}, \ldots x_{(h)}, \ldots, \bar{x}_{(L)})$$

We observe that (i) of Assumption 4.1 is a technical condition needed to manage projection operations for infinite subsequences. Condition (ii) requires that the restricted sets $\mathcal{F}_h(x^k)$ capture, in the limit, the geometry of the set $\mathcal{F}_h$ in terms of optimality conditions. Just to have an idea how to satisfy condition (ii), we can consider the case (central in the present work) of $\mathcal{F}_h$ defined as a simplex, that is

$$\mathcal{F}_h = \left\{ x \in \mathbb{R}^{n_h} : e' x_{(h)} = d_h, \quad x_{(h)} \geq 0 \right\}$$

Given a feasible point x^k, the restricted set $\mathcal{F}_h(x^k)$ can be defined by considering as variables only the components $x^k_{(h),i} > 0$ and that component $x^k_{(h),i^\star} = 0$ that in some way mostly violates the optimality conditions, that is,

$$\frac{\partial f(x^k)}{x_{(h),i^\star}} \leq \frac{\partial f(x^k)}{x_{(h),j}} \qquad j \in \{1, \ldots, n_h\}$$

We will show later that the above definition of the restricted set $\mathcal{F}_h(x^k)$ allows us to satisfy Assumption 4.1.

In algorithm 10 we describe the inexact decomposition algorithm (IDA) based on the gradient projection and on the well-known Armijo-type line search, together with a theoretical result employed in our convergence analysis.

In order to ensure the global convergence of algorithm IDA we need to introduce the following assumption, which requires that each index $l \in \{1, \ldots, L\}$ (corresponding to the block component $x_{(l)}$) is periodically considered at step 3 within a prefixed maximum number of iterations.

Assumption 4.2. *There exists an integer $M > 0$ such that, for all $k \geq 0$ and for all $l \in \{1, \ldots, L\}$, we can find an index $l(k)$, with $0 \leq l(k) \leq M$, such that at step 4 we have $h^{k+l(k)} = l$.*

Note that setting, for instance, $h^k = (k \mod L) + 1$, algorithm IDA reduces to an inexact Gauss-Seidel algorithm (on restricted feasible component subsets) where a single iteration of the projection gradient method is performed for every

Algorithm 10: Inexact Decomposition Algorithm (IDA)

```
  Input: $x^0 \in \mathcal{F}$
1 $k \leftarrow 0$;
2 while stopping criterion not satisfied do
3     choose $h^k \in \{1, \dots, L\}$;
4     define $\mathcal{F}_{h^k}(x^k)$;
5     set
          $d^k_{(i)} = \begin{cases} 0_{(i)} & \text{if } i \neq h^k \\ \hat{x}^k_{(i)} - x^k_{(i)} & \text{if } i = h^k \end{cases}$
      where $\hat{x}^k_{(h^k)} = P_{\mathcal{F}_{h^k}(x^k)}[x^k_{(h)} - \nabla_{(h)} f(x^k)]$;
6     $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
7     $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8     $k \leftarrow k + 1$;
9 end
```

block-component. The literature on the convergence of exact decomposition algorithms is wide (see, eg., [6], [43], [57], [60]). A recent study on inexact Gauss-Seidel algorithms based on gradient projection mappings has been performed in [11]. We remark that the convergence analysis of algorithm IDA can not be derived from results stated in preceding works, since algorithm IDA involves restricted feasible subsets $\mathcal{F}_h(x^k)$ instead of the prefixed subsets $\mathcal{F}_h$.

We are ready to state the following convergence result.

Theorem 4.3. *Let $\{x^k\}$ be the sequence generated by the algorithm* IDA. *Suppose that for all k the sets $\mathcal{F}_{h^k}(x^k)$ defined at step 4 satisfy Assumption* 4.1, *and that the sequence $\{h^k\}$ is such that Assumption* 4.2 *holds. Then $\{x^k\}$ admits limit points and each limit point is a solution of problem* (4.3).

Proof. The points of the sequence $\{x^k\}$ belong to the feasible compact set, so $\{x^k\}$ admits limit points.

Let $x^\star$ be a limit point of $\{x^k\}$, i.e., there exists an infinite subset $K \subseteq \mathbb{N}$ such that

$$\lim_{k \in K, k \to \infty} x^k = x^\star \tag{4.4}$$

The instructions of the algorithm imply

$$f(x^{k+1}) \leq f(x^k)$$

so that, as f is bounded below, we can write

$$\lim_{k \to \infty} f(x^{k+1}) - f(x^k) = 0 \tag{4.5}$$

From the properties of the projection mapping we get

$$\nabla f(x^k)'d^k = \nabla_{(h^k)} f(x^k)' d^k_{(h^k)} \le -\|\hat{x}^k_{(h^k)} - x^k_{(h^k)}\|^2 = -\|\hat{x}^{k+1} - x^k\|^2 \tag{4.6}$$

being

$$\hat{x}^k_{(h^k)} = P_{\mathcal{F}_{h^k}(x^k)}[x^k_{(h^k)} - \nabla_{(h^k)} f(x^k)] \tag{4.7}$$

Furthermore, for all $k \in K$ we have

$$f(x^{k+1}) \le f(x^k) + \gamma \alpha^k \nabla f(x^k)' d^k$$

where α^k is determined by means of the Armijo line search along the search direction d^k. Note that, due to the convexity of $\mathcal{F}_{h^k}(x^k)$, the maximum feasible step length β^k along d^k is greater than or equal to 1. Furthermore, as the closed convex set $\mathcal{F}_{h^k}(x^k)$ belongs, by assumption, to the compact set $\mathcal{F}$, we have that the search direction d^k is bounded.

Then using (4.5) and assertion *(ii)* of theorem A.2 we obtain

$$\lim_{k\to\infty} \nabla f(x^k)' d^k = 0 \tag{4.8}$$

From (4.8) and (4.6) it follows

$$\lim_{k\to\infty} \|x^{k+1} - x^k\| = \lim_{k\to\infty} \|\hat{x}^k_{(h^k)} - x^k_{(h^k)}\| = \lim_{k\to\infty} \|d^k\| = 0 \tag{4.9}$$

From (i) of Assumption 4.1 it follows that, for every $j \in \{1, \dots, L\}$, we can find an infinite subset $K_1 \subseteq K$ and an $\mathcal{F}^\star_j$ such that

$$\mathcal{F}_j(x^k) = \mathcal{F}^\star_j \qquad \forall k \in K_1$$

Recalling Assumption 4.2 we have that $h^{k+j(k)} = j$, with $0 \le j(k) \le M$, and hence, using (4.9) we can write

$$\lim_{k\in K_1, k\to\infty} x^{k+j(k)} = x^\star \tag{4.10}$$

From (4.7), (4.9) and (4.10), recalling the continuity of the projection mapping, we obtain

$$x^\star_{(j)} = P_{\mathcal{F}^\star_j}[x^\star_{(j)} - \nabla_{(j)} f(x^\star)] \tag{4.11}$$

which implies that

$$x^\star_{(j)} \in \arg\min_{x_{(j)} \in \mathcal{F}^\star_j} f(x^\star_{(1)}, \dots, x_{(j)}, \dots, x^\star_{(L)}) \tag{4.12}$$

Taking into account (4.12) and recalling (ii) of Assumption 4.1 we have

$$x^\star_{(j)} \in \arg \min_{x_{(j)} \in \mathcal{F}_j} f(x^\star_{(1)}, \dots, x_{(j)}, \dots, x^\star_{(L)})$$

This equation holds for every $j \in \{1, \dots, L\}$, and hence the theorem is proved.
$\square$

4.2 Decomposition algorithms for NEP

In this section we present two decomposition schemes for the special case of the network equilibrium problem defined by (4.1) and (4.13), as well as the case with no decomposition at all. We define the feasible set as the Cartesian product of

$$\mathcal{F}_p = \left\{ x \in \mathbb{R}^{n_p} : e' x_{(p)} = d_p, \quad x_{(p)} \geq 0 \right\} \tag{4.13}$$

We recall that

- given $x \in \mathcal{F}$, $x_{(h),i}$ is the flow of the i-th path between the h-th OD pair;
- $f : R^n \to R$ is a convex continuously differentiable function;
- the partial derivative
$$\frac{\partial f(x)}{\partial x_{(h),i}}$$
is the cost of the i-th path.

We exploit the special structure of the feasible set to derive two algorithms from the general framework depicted in algorithm 10. To this aim we need to specify:

(a) the rule for constructing the restricted feasible set $\mathcal{F}_h(x)$;

(b) the rule for selecting the block variables $x_{(h^k)}$ to be updated at each iteration k.

4.2.1 OD-pairs based decomposition algorithm

Problem (4.1) has the same form of problem (4.3), being $L = P$ the number of convex compact subsets $\mathcal{F}_h$ whose Cartesian product defines the feasible set $\mathcal{F}$. Formulation (4.1) naturally leads to an *OD-pair based decomposition* scheme.

Concerning point (a), since each subproblem has a huge number of variables (i.e., there exists a huge number of paths between each OD pair), the aim is to avoid enumerating them a priori. Thus, the basic idea is to consider, for each subproblem, only the variables whose current value is strictly positive (corresponding to paths carrying non-zero flows), and the variable that corresponds to the cheaper path to the destination. In this way we consider restricted feasible sets belonging to lower dimensional subspaces.

Formally, given $\bar{x} \in \mathcal{F}$, we set

$$I_h^+(\bar{x}) = \{i \in \{1, \dots, n_h\} : \ \bar{x}_{(h),i} > 0\}$$

and

$$\pi_h(\bar{x}) \in \arg \min_{j=1,\dots,n_h} \frac{\partial f(\bar{x})}{\partial x_{(h),j}}$$

Then we define the index set identifying the variables to be updated

$$I_h(\bar{x}) = I_h^+(\bar{x}) \cup \{\pi_h(\bar{x})\}$$

For each $h \in \{1, \dots, P\}$ we introduce the restricted feasible set $\mathcal{F}_h(\bar{x})$ defined as follows

$$\mathcal{F}_h(\bar{x}) = \{x_{(h)} \in \mathbb{R}^{n_h} : x_{(h)} \in \mathcal{F}_h, \ x_{(h),i} = 0 \ \forall i \notin I_h(\bar{x})\} \tag{4.14}$$

Assumption 4.1 holds for the restricted feasible set defined by (4.14), as stated in the next theorem

Theorem 4.4. *For any point $x \in \mathcal{F}$ and for $h = 1, \dots, P$ let $\mathcal{F}_h(x)$ be the restricted feasible set defined as in* (4.14)*. Then Assumption* 4.1 *holds.*

Proof. **(i)** Let $x \in \mathcal{F}$ and $h \in \{1, \dots, P\}$. Note that $\mathcal{F}_h(x)$ only depends from $I_h(x)$, and $I_h(x) \subseteq \{1, \dots, n_h\}$, with $I_h(x) \neq \emptyset$. Then we have

$$|\cup_{x \in \mathcal{F}} \{\mathcal{F}_h(x)\}| = 2^{n_h} - 1$$

(ii) If

$$\bar{x}_{(h)} \in \arg \min_{x_{(h)} \in \mathcal{F}_h(x^k)} f(\bar{x}_{(1)}, \dots x_{(h)}, \dots, \bar{x}_{(P)})$$

then, using the optimality conditions, for any $i \in \{1, \dots, n_h\}$ such that $\bar{x}_{(h),i} > 0$ we can write

$$\frac{\partial f(\bar{x})}{\partial x_{(h),i}} \leq \frac{\partial f(\bar{x})}{\partial x_{(h),j}} \qquad \forall j \in I_h(x^k) \tag{4.15}$$

By relabelling, if necessary, the infinite subset K, we have $\pi_h(x^k) = i^\star$ for all

$k \in K$, and by definition of $\pi_h(x^k)$, we have

$$\frac{\partial f(x^k)}{\partial x_{(h),i^\star}} \le \frac{\partial f(x^k)}{\partial x_{(h),j}} \qquad \forall j \in \{1,\dots,n_h\} \tag{4.16}$$

From condition (4.15), as $i^\star \in I_h(x^k)$, it follows

$$\frac{\partial f(\bar{x})}{\partial x_{(h),i^\star}} \ge \frac{\partial f(\bar{x})}{\partial x_{(h),i}} = \frac{\partial f(\bar{x})}{\partial x_{(h),j}} \qquad \forall i,j \in \{1,\dots,n_h\} \text{ s.t. } x^k_{(h),i},\ x^k_{(h),j} > 0 \tag{4.17}$$

Now by contradiction assume that

$$\bar{x}_{(h)} \notin \arg\min_{x_{(h)}\in\mathcal{F}_h} f(\bar{x}_{(1)},\dots x_{(h)},\dots,\bar{x}_{(P)}) \tag{4.18}$$

Then, there exists a pair $(\hat{\imath},\hat{\jmath})$ of indexes in $\{1,\dots,n_h\}$ such that $x_{(h),\hat{\jmath}} > 0$ and

$$\frac{\partial f(\bar{x})}{\partial x_{(h),\hat{\imath}}} < \frac{\partial f(\bar{x})}{\partial x_{(h),\hat{\jmath}}} \tag{4.19}$$

Note that $\bar{x}_{(h),i} > 0$ implies $x^k_{(h),i} > 0$ for $k \in K$ and k sufficiently large. Hence, from (4.19) and (4.17), we get that $\bar{x}_{(h),\hat{\imath}} = 0$ and

$$\frac{\partial f(\bar{x})}{\partial x_{(h),\hat{\imath}}} < \frac{\partial f(\bar{x})}{\partial x_{(h),j}} \qquad \forall j \in \{1,\dots,n_h\} \text{ s.t. } \bar{x}_{(h),j} > 0 \tag{4.20}$$

Taking the limits in (4.16) for $k \in K$ e $k \to \infty$, and recalling (4.20) we obtain

$$\frac{\partial f(\bar{x})}{\partial x_{(h),i^\star}} \le \frac{\partial f(\bar{x})}{\partial x_{(h),\hat{\imath}}} < \frac{\partial f(\bar{x})}{\partial x_{(h),j}} \qquad \forall j \in \{1,\dots,n_h\} \text{ s.t. } \bar{x}_{(h),j} > 0$$

which contradicts (4.17). □

Note that the definition of the set $\mathcal{F}_h(\bar{x})$ requires to determine the index $\pi_h(\bar{x})$, and this can be done by computing the shortest path between the considered origin and destination.

As regards point (b), we keep h_k constant for a fixed number of iterations $n_{iter} \ge 1$, that is, each block of variables $x_{(h)}$ is sequentially selected and n_{iter} iterations of the gradient projection method are performed for its updating.

The formal description of the algorithm, which is a specific realization of algorithm IDA, named IDA-OD, is reported in algorithm 11.

The global convergence of the algorithm follows from theorem 4.3 and is stated in the following theorem.

Algorithm 11: Inexact Decomposition Algorithm for Origin-Destination pairs (IDA-OD)

```
   Input: $x^0 \in \mathcal{F}$, $n_{iter} \geq 1$
1  $k \leftarrow 0$, $l \leftarrow 1$, $count \leftarrow 0$;
2  while stopping criterion not satisfied do
3      $h^k \leftarrow l$ ;
4      define $\mathcal{F}_{h^k}(x^k)$ as in (4.14) replacing $\bar{x}$ with $x^k$;
5      set
           $d^k_{(i)} = \begin{cases} 0_{(i)} & \text{if } i \neq h^k \\ \hat{x}^k_{(i)} - x^k_{(i)} & \text{if } i = h^k \end{cases}$
       where $\hat{x}^k_{(h^k)} = P_{\mathcal{F}_{h^k}(x^k)}[x^k_{(h)} - \nabla_{(h)} f(x^k)]$;
6      $\alpha^k \leftarrow$ Armijo linesearch along $d^k$;
7      $x^{k+1} \leftarrow x^k + \alpha^k d^k$;
8      if $count < n_{iter}$ then
9          $count \leftarrow count + 1$;
10     else
11         $count \leftarrow 0$;
12         $l \leftarrow (l \mod P) + 1$;
13     end
14     $k \leftarrow k + 1$;
15 end
```

Theorem 4.5. *Let $\{x^k\}$ be the sequence generated by the algorithm* IDA-OD*. Then $\{x^k\}$ admits limit points and each limit point is a solution of problem* (4.1)*.*

Proof. The rule for defining the restricted feasible set $\mathcal{F}_{h^k}(x^k)$ at step 4 is such that Assumption 4.1 holds (see theorem 4.4), and the sequence $\{h^k\}$ is generated in such a way that Assumption 4.2 is satisfied with $M = L \cdot n_{iter}$. Then the thesis follows from theorem 4.3. □

4.2.2 Origin based decomposition algorithm

The network equilibrium problem formulated in (4.1) can be rewritten in an equivalent form in order to define another decomposition scheme.

More specifically, denoting by O the set of the origins, with $|O| = R$, and denoting with $D(h)$, for each $h \in O$, the set of destinations associated to the origin h, we can equivalently write the problem as follows

$$\begin{aligned} &\min_x f(x) \\ &x \in \mathcal{F} = \Omega_1 \times \Omega_2 \times \dots \Omega_R \end{aligned} \tag{4.21}$$

where

$$\Omega_h = \prod_{l \in D(h)} \mathcal{F}_l$$

Formulation (4.21) induces an *Origin-based decomposition* scheme.

Again, concerning point (a), following the strategy previously described for the OD-pair based decomposition algorithm, given $\bar{x} \in \mathcal{F}$, for each $h \in O$ we introduce the restricted feasible set $\Omega_h(\bar{x})$ defined as follows

$$\Omega_h(\bar{x}) = \prod_{l \in D(h)} \mathcal{F}_l(\bar{x}) \tag{4.22}$$

where the restricted feasible subset $\mathcal{F}_l(\bar{x})$ is defined in (4.14). We can prove that the restricted feasible set so defined is such that Assumption 4.1 holds as stated in the next theorem

Theorem 4.6. *For any point $x \in \mathcal{F}$ and for $h = 1, \dots, R$ let $\Omega_h(x)$ be the restricted feasible set defined as in* (4.22)*. Then Assumption* 4.1 *holds.*

Proof. **(i)** Similarly to theorem 4.4, the set $\Omega_h(x)$ only depends on $I_p(x)$, with $p \in D(h)$. Every $I_p(x)$ is a non-empty subset of $\{1, \dots, n_p\}$. Then

$$|\cup_{x \in \mathcal{F}} \{\Omega_h(x)\}| = \prod_{p \in D(h)} (2^{n_p} - 1)$$

(ii) By point (ii) of theorem 4.4, for every $p \in D(h)$, it holds

$$\bar{x}_{(p)} \in \arg \min_{x_{(p)} \in \mathcal{F}_p} f(\bar{x}_{(1)}, \dots x_{(p)}, \dots, \bar{x}_{(P)})$$

Then, since the set Ω_h is separable, it also holds

$$\bar{x}_{(h)} \in \arg \min_{x_{(h)} \in \Omega_h} f(\bar{x}_{(1)}, \dots x_{(h)}, \dots, \bar{x}_{(R)})$$ □

Note that the definition of the set $\Omega_h(\bar{x})$ requires to determine the indexes $\pi_l(\bar{x})$, for $l \in D(h)$ and this can be done by computing the shortest paths tree between the considered origin and all its destinations.

The rule for selecting the block variables $x_{(h^k)}$ to be updated at each iteration k is the same adopted for algorithm IDA-OD.

We do not yield the formal description the algorithm, called IDA-O, since it can be immediately derived from the previously described algorithm IDA-OD by replacing the restricted feasible set $\mathcal{F}_{h^k}(x^k)$ with $\Omega_{h^k}(x^k)$.

The global convergence of the algorithm follows from theorem 4.3 and is stated in the following theorem

Theorem 4.7. *Let $\{x^k\}$ be the sequence generated by the algorithm* IDA-O*. Then $\{x^k\}$ admits limit points and each limit point is a solution of problem* (4.21)*.*

Proof. The rule for defining the restricted feasible set $\Omega_{h^k}(x^k)$ at step 4 is such that Assumption 4.1 holds (see theorem 4.6), and the sequence $\{h^k\}$ is generated in such a way that Assumption 4.2 is satisfied with $M = R * n_{iter}$. Then the thesis follows from theorem 4.3. □

4.2.3 Column-generation based Projected Gradient

The network equilibrium problem formulated in (4.1) can be also tackled without using a decomposition scheme at all. Indeed, the Projected Gradient framework based on the column-generation approach directly applies when no decomposition is used, i.e. a special case in which we select all OD pairs at each iteration.

For what concern point (a), following the strategy previously described for the OD-pair based decomposition algorithm, given $\bar{x} \in \mathcal{F}$, the restricted feasible set $\Gamma(\bar{x})$ is defined as follows

$$\Gamma(\bar{x}) = \prod_{p \in P} \mathcal{F}_p(\bar{x}) \tag{4.23}$$

where the restricted feasible subset $\mathcal{F}_p(\bar{x})$ is defined in (4.14). Again, we can prove that $\Gamma(\bar{x})$ is such that Assumption 4.1 holds as stated in the next theorem

Theorem 4.8. *For any point $x \in \mathcal{F}$ let $\Gamma(x)$ be the restricted feasible set defined as in* (4.23)*. Then Assumption* 4.1 *holds.*

Proof. **(i)** Trivially, in this case

$$|\cup_{x \in \mathcal{F}} \Gamma(x)| = 2^n - 1$$

(ii) Again by point (ii) of theorem 4.4, for every $p \in D(h)$, it holds

$$\bar{x}_{(p)} \in \arg \min_{x_{(p)} \in \mathcal{F}_p} f(\bar{x}_{(1)}, \ldots x_{(p)}, \ldots, \bar{x}_{(P)})$$

Then, since the set Γ is separable, it also holds

$$\bar{x}_{(h)} \in \arg \min_{x_{(h)} \in \Gamma} f(\bar{x}_{(1)}, \ldots x_{(h)}, \ldots, \bar{x}_{(R)})$$

□

Note that the definition of the set $\Gamma(\bar{x})$ requires to determine the indexes $\pi_p(\bar{x})$, for $p \in P$ and this can be done by computing the shortest paths tree between all origins to all their destinations.

The resulting algorithm, denoted as PG, can be obtained from algorithm IDA-OD by replacing the restricted feasible set $\mathcal{F}_{h^k}(x^k)$ with $\Gamma(x^k)$.

The global convergence of the algorithm follows from theorem 4.3 and is stated in the following theorem.

Theorem 4.9. *Let $\{x^k\}$ be the sequence generated by the algorithm* PG*. Then $\{x^k\}$ admits limit points and each limit point is a solution of problem* (4.21).

Proof. The rule for defining the restricted feasible set $\Gamma(x^k)$ is such that Assumption 4.1 holds (see theorem 4.6). Assumption 4.2 trivially holds, since we select all OD pairs at each iteration. Then the thesis follows from theorem 4.3. □

4.3 Numerical experiments

In this section we show the results of the computational experiments performed on a widely used data set by the proposed inexact decomposition algorithms based on a column generation strategy. The aims of the experimentation were mainly the following:

(a) to verify the applicability of the column generation-based strategy (characterizing the new methods) for solving real large dimensional problems;

(b) to compare the new methods with the baseline algorithm for network equilibrium problem, that is, the Frank-Wolfe method;

(c) to evaluate the possible advantages of the inexact decomposition algorithms compared with (approximately) exact decomposition methods and with methods not using a decomposition strategy.

Concerning point (a), we remark that both algorithm IDA-OD and algorithm IDA-O require to store path variables that are positive. As shown in Table 4.7, the number of paths stored at each iteration is very small, and this confirms that the adopted column generation strategy is a viable technique for tackling large dimensional problems.

We observe that the experiments have been performed on data sets (described in section 4.3.1) concerning traffic equilibrium problems with additive path costs, although the proposed algorithms can be applied, in principle, even to problems with nonadditive path costs. In this latter case, the standard Dijkstra algorithm for the shortest path computation can not be applied, but (slightly) more complex multi-label shortest route methods can be used (see

[55]). In order to yield answers to the computational issues discussed above (which do not depend on the assumption of either additive or nonadditive path costs), for sake of simplicity we have preferred to limit our computational experience to traffic equilibrium problems with additive path costs, so that several efficient implementations of the Dijkstra algorithm are easily available and can be used to implement the algorithms to be tested. Future computational work will be devoted to intensive experiments on problems with nonadditive path costs.

4.3.1 Test problems

We performed numerical experiments using the freely available data sets from the repository of Hillel Bar-Gera[1] and listed in Table 4.1. This data set refers to traffic equilibrium problems with additive path costs, so to problems of the form (4.2), which leads to problems of the form (4.1) by simple substitution. The arc cost is expressed by the BPR function (see for instance [46, 68]) defined, for each link $i \in \{1, \dots, m\}$, as

$$s_i(y_i) = \phi_i + \phi_i \beta_i \left(\frac{y_i}{c_i} \right)^{\alpha_i} \tag{4.24}$$

where $C_i, \beta_i, \alpha_i, \phi_i$ are arc dependent parameters, that characterize the traffic network, and

$$y_i = \sum_{j=1}^{n} h_{ij} x_j$$

is the arc flow being h_{ij} elements of the arc-path incidence matrix. Then, according to the adopted notation, we have

$$F_i(y_i) = \int_0^{y_i} \left[\phi_i + \phi_i \beta_i \left(\frac{z}{c_i} \right)^{\alpha_i} \right] dz$$

name	#nodes	#links	#zones	# OD-pair
Barcelona	1020	2522	110	7922
Winnipeg	1067	2975	154	4344
Berlin Central	12981	28376	865	49688
Chicago Sketch	933	2950	387	93135

Table 4.1: Data set information.

[1]`http://www.bgu.ac.il/~bargera/tntp/`

Performance evaluation

For all tests we report the so-called *relative gap* (see for instance [4] for more details), a widely used quality function defined as

$$r_{gap}(x) = 1 - \frac{\sum_{p=1}^{P} \pi_p(x) d_p}{\sum_{i=1}^{m} s_i(y_i) y_i} = 1 - \frac{\sum_{p=1}^{P} \pi_p(x) d_p}{\sum_{i=1}^{m} s_i(\sum_{j=1}^{n} h_{ij} x_j) \sum_{j=1}^{n} h_{ij} x_j} \tag{4.25}$$

We recall that π_p is the shortest path cost for the p-th OD pair. The denominator represents the sum, for each arc, of the arc cost weighted by the flow that moves through that arc. It is easy to see that, by the Wardrop equilibrium conditions, $r_{gap}(x^k)$ converges to zero if and only if x^k converges to a solution of the network equilibrium problem.

Implementation details: initial solution, projection over a simplex and scaling

An initial feasible solution, needed to initialize all the implemented algorithms, has been obtained by computing the shortest path tree from each origin (following the origin id order) to all destinations and loading on each OD pair shortest path the entire demand. We have adopted the strategy *once-at-a-time* proposed in [69]. More in particular, the origins are sequentially considered, once that a single origin has been processed, the flow vector and the corresponding path cost vector are reevaluated. In this way, when computing the shortest paths tree for a given origin (with the exception of the first processed origin), we are not considering the graph with an empty flow, so that a better starting point can be obtained at no additional cost. Formally, the vector x is initialized using algorithm 12, where O is the set of origins.
The Projected Gradient method requires to project a point over a simplex. Such operation is performed by a very simple algorithm (see [65] for the details), which finds the projection of a point in at most n iterations.

We have also implemented a scaled version of IDA-O algorithm (named IDA-SO, see later). We recall that in IDA-O algorithm each block component refers to several OD pairs having the same origin. Then, in order to take into account that different OD pairs may have different scales and may generate very different steps, we have scaled the search direction by premultiplying it for a diagonal matrix having on the diagonal, for each OD pair, the maximum steplength (greater than or equal to one) preserving the non-negativity of the

Algorithm 12: Finds a feasible flow x on the graph G for a given demand d.

```
Data: A flow-dependent graph $G$, a set of $P$ OD pairs with
      corresponding demand vector $d$.
1  set $x = 0$;
2  foreach $i \in O$ do
3      let $t_i$ be the shortest path tree from $i$ on $G$;
4      foreach $p \in P$ with origin $i$ do
5          let $h_w \in t_i$ be the shortest path between the pair $p$;
6          let $d_w$ the demand of the pair $p$;
7          set $x_{h_p} = d_p$;
8      end
9      update arc costs with the current path flow $x$;
10 end
```

corresponding used variables. This should promote a significant updating for all blocks of variables related to different OD pairs. By imposing suitable bounds on the elements of the scaling matrix convergence properties of the algorithm are obviously guaranteed.

4.3.2 Results

In this section we show the results obtained on the four test problems by the algorithms listed in Table 4.2. The first three are the gradient projection decomposition algorithms presented in section 4.2.1 (the third one is the scaled version), the fourth one is the gradient projection method without decomposition as introduced in section 4.2.1. All these algorithms are path-based methods for the solution of formulation (4.1). The fifth tested algorithm is the Frank-Wolfe algorithm, which is applied to the arc-based formulation (4.2), and is the most used baseline algorithm.

For what concerns the number n_{iter} of projected-gradient iterations for IDA-like algorithms, no significant differences have been observed when performing more than one iteration (say $n_{iter} = 5, 10, 20$). Thus, we only present results in the case of $n_{iter} = 1$.

In order to assess the usefulness of the inexact decomposition strategy characterizing IDA-like algorithms, we have also implemented two corresponding (approximately) exact versions of the same algorithms. More in particular, the two versions have been realized by setting algorithms IDA-OD and IDA-O n_{iter} sufficiently high (say $n_{iter} = 100$), and by introducing an inner stopping criterion for the solution of each subproblem ($\|d^k\|_\infty \leq 10^{-8}$). These two versions have been named EDA-OD and EDA-O respectively.

algorithm	name
Inexact Decomposition Algorithm-OD	IDA-OD
Inexact Decomposition Algorithm-O	IDA-O
Inexact Decomposition Algorithm-O with Scaling	IDA-SO
Projected Gradient	PG
Frank-Wolfe	FW
Exact Decomposition Algorithm-OD	EDA-OD
Exact Decomposition Algorithm-O	EDA-O

Table 4.2: Algorithms tested.

All tests have been performed on a Intel Core i7 2.93GHz standard desktop machine with 3GByte of RAM. The algorithms have been coded in C++ using the Boost Graph Library[2] implementation of the Dijkstra algorithm for the shortest path computation, as well as the graph representation.

For each test problem and for each algorithm we report in Tables 4.3-4.6 the CPU time required to satisfy the stopping criterion, i.e., for attaining a value of the relative gap (see (4.25)) less than or equal to the tolerance ϵ, which has been fixed to different values. The symbol $*$ indicates that the algorithm was not able to satisfy the stopping criterion within $5 \cdot 10^3$ seconds for Winnipeg, 10^4 seconds for Barcelona and $2 \cdot 10^4$ seconds for Berlin Central and Chicago Sketch.

algorithm	$\epsilon = 10^{-4}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$
IDA-OD	11.23	21.43	37.33	109.20
IDA-O	26.93	37.52	60.19	108.96
IDA-SO	1.81	3.46	16.19	40.10
PG	245.45	732.25	818.66	1004.70
EDA-OD	291.81	674.12	805.12	1375.88
EDA-O	107.05	243.54	291.94	411.98
FW	4.09	27.75	*	*

Table 4.3: Barcelona road network: CPU time (seconds) required to attain $r_{gap}(x^k) \leq \epsilon$.

From the results reported in Tables 4.3-4.6, we can first of all observe that the proposed approach outperforms the standard Frank-Wolfe algorithm both in speed and accuracy in most of the proposed variants. In particular, it can be seen that, despite its ability to quickly reach low accuracy solutions, Frank-Wolfe method can not usually make any significant progress towards high quality ones. However, the worst convergence characteristics of the Frank-Wolfe method were well-known.

For what concerns the different proposed algorithms that have been tested,

[2]http://www.boost.org/doc/libs/1_46_1/libs/graph/doc/index.html

algorithm	$\epsilon = 10^{-4}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$
IDA-OD	10.13	23.09	71.60	94.35
IDA-O	14.46	25.74	47.77	88.38
IDA-SO	3.22	18.79	57.11	106.23
PG	64.51	126.09	293.06	469.27
EDA-OD	292.80	755.86	3157.87	4245.42
EDA-O	154.03	478.03	1596.67	2140.79
FW	21.67	147.16	*	*

Table 4.4: Winnipeg road network: CPU time (seconds) required to attain $r_{gap}(x^k) \leq \epsilon$.

algorithm	$\epsilon = 10^{-4}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$
IDA-OD	73.17	167.33	426.13	657.70
IDA-O	39.48	106.27	249.10	809.90
IDA-SO	32.78	106.57	175.14	259.80
PG	154.60	2029.64	10082.73	*
EDA-OD	687.38	1979.10	6809.64	13199.06
EDA-O	397.08	1014.59	2828.35	4444.50
FW	114.96	1360.90	*	*

Table 4.5: Berlin Central road network: CPU time (seconds) required to attain $r_{gap}(x^k) \leq \epsilon$.

we observe that any inexact decomposition-based algorithm outperforms the PG algorithm. Moreover, exact versions are far less efficient than the inexact ones: as conjectured, the effort to exactly optimize a subproblem is not balanced by a substantial improve in the overall convergence speed.

Comparing origin-based and OD-based is tricky: the former is much faster in achieving the same accuracy especially when the network dimension grows. For Barcelona and Winnipeg IDA-O outperforms IDA-OD only when very high accuracy is required; for Berlin Central IDA-OD is better for low and high accuracy, while for the Chicago Sketch network the difference is of one order of magnitude for IDA-OD.

algorithm	$\epsilon = 10^{-4}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-7}$
IDA-OD	108.07	183.40	261.88	548.67
IDA-O	1095.46	5023.69	8741.1	15640.47
IDA-SO	25.90	384.12	611.04	668.17
PG	9283.45	*	*	*
EDA-OD	2557.88	4142.85	6811.27	14175.96
EDA-O	1076.44	3343.16	8969.27	16363.40
FW	34.72	258.70	*	*

Table 4.6: Chicago Sketch road network: CPU time (seconds) required to attain $r_{gap}(x^k) \leq \epsilon$.

network	PG	IDA-O	IDA-OD
Barcelona	1.68	1.47	1.49
Winnipeg	1.68	2.08	2.00
Chicago Sketch	2.57	1.38	1.41
Berlin Central	1.13	1.18	1.11

Table 4.7: Average number of active paths for each OD pair at the best solution found by each algorithm.

The differences in performance between IDA-O and IDA-OD are mainly due to two factors: the usage of the Dijkstra algorithm to find either single shortest paths or shortest path trees; the effectiveness of the search directions generated by the projected gradient inner step of the decomposition scheme. For what concern the former, IDA-O can compute shortest path trees from each origin, with a great saving of time. This advantage can be easily verified looking at the time per iteration (i.e. the time to process all OD pairs) which is much lower for IDA-O. On the other hand, search directions generated by IDA-OD seem to be more effective, allowing a greater reduction of the objective function at each iteration.

The previous observations are confirmed by the substantial improvement achieved by the IDA-SO algorithm. The introduction of the scaling matrix seems to substantially improve the practical properties of the scaled descent direction. We observe that in many cases (with different levels of accuracy required) IDA-SO algorithm outperforms all the other tested algorithms. Then, on the basis of the experimentation performed on the four test problems, IDA-SO algorithm seems to be the preferable algorithm among those presented in this work.

It is also worth to be noticed that there is room for improvements tuning the gradient projection: the gradient can be scaled by any factor $\lambda > 0$ (which has been set to one in our work) without loosing he convergence properties or any significant computational burden. Scaling the gradient affects the search direction: extensive tests not here reported have shown that tuning λ can improve the algorithms performance.

Being a critical issues for path-based algorithms, we report in Table 4.7 the average number of active paths for each OD pair recorded at the best solution found by the algorithms IDA-O, IDA-OD and PG.

Values reported in Table 4.7 clearly show that high quality solutions are characterized by a number of active paths of the same magnitude of the number of OD pairs. Unless the latter became huge, the proposed algorithms represent then a viable strategy also in terms of memory consumption.

Chapter 5

Global optimization methods for nonlinear mixed integer portfolio selection problems

Portfolio selection theory studies how to allocate an investor's available capital into a prefixed set of assets with the aims of maximizing the expected return and minimizing the investment risk.

We denote by n the number of available assets, by $\mu \in \mathbb{R}^n$ the vector of expected returns of the assets, and by $Q \in \mathbb{R}^{n\times n}$ the symmetric positive semidefinite matrix whose generic element q_{ij} is the variance of returns of asset i and asset j. Usually both the vector μ and the matrix Q are not known analytically but can be estimated using historical data.

Let us assume that one unit of capital is available and that we want it to be fully invested. Then, let $x \in \mathbb{R}^n$ be the vector of decision variables, where x_i is the fraction of the available capital to be invested into asset i, with $i = 1, \dots, n$. Since the available capital is to be entirely used for investment and no short positions are allowed, vector x must satisfy the constraints

$$e'x = 1 \quad x \geq 0$$

where $e \in \mathbb{R}^n$ denotes the column vector of all ones. Then, by this notation, $\mu' x$ is the expected return of the portfolio and $x'Qx$ is the variance of the portfolio which can be used as a measure of the risk connected with the investment [62]. Hence, the classical Markowitz portfolio selection model [62] seeks for solutions that minimize the risk $(x''Qx)$ while maximizing the expected return $(\mu'x)$ of the portfolio.

In the traditional Markowitz portfolio selection model [62] this bi-objective problem is transformed into a single-objective optimization problem, where the objective function is the risk of the portfolio $(x'Qx)$, and the expected return $(\mu'x)$ of the assets is fixed to a chosen value. Formally, the optimization problem is stated as the following convex quadratic program

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & x'Qx \\ & \mu'x = \beta \\ & e'x = 1 \\ & x \geq 0 \end{aligned} \tag{5.1}$$

where β is the desired expected return of the portfolio. The main drawback of problem (5.1) is that means and covariances of the assets are not sufficiently accurate since they have to be estimated from historical (and typically) noisy data. Indeed, it is extremely difficult to estimate the mean returns to working precision and this is a known phenomenon referred to as *mean blur* [59, 63]. Besides, the mean-variance model (5.1) is very sensitive to distributional input parameters. As a result, the model amplifies any estimation error thus yielding extreme portfolios which, as it can be seen, perform poorly in out-of-sample tests [14, 22, 27, 64]. Several variants of the Markowitz model have been proposed and many attempts have been undertaken to ease the mentioned amplification of estimation errors and yield more stable portfolios. In [10] Bayesian estimation of means and covariances has been employed. In [51, 20] additional portfolio constraints have been imposed in the model in order to guide the optimization process towards more diversified portfolios. In [21] the use of a James-Steiner estimator for the means have been proposed which steers the optimal allocations towards the minimum-variance portfolio. The employment of robust estimators has been investigated in [27]. In [9, 32, 15, 26] an important class of portfolio selection problems has been defined by limiting the number of assets to be held in the portfolio so as to reduce both the transaction costs and the complexity of portfolio management. Such a constraint, as argued in [15], helps inducing sparsity of the selected portfolio and can be a remedy to the high instability of classic methods for portfolio selection.

In particular, in [8, 9, 18] the Markowitz model has been modified by adding to problem (5.1) a constraint on the number of assets that can be held in the portfolio. This kind of problems, usually called *cardinality-constrained portfolio selection problems*, are stated as follows

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & x'Qx \\ & \mu'x = \beta \\ & e'x = 1 \\ & x \geq 0 \\ & \|x\|_0 \leq K \end{aligned} \tag{5.2}$$

where the parameter K is the chosen limit of assets to be held in the portfolio. Optimal portfolios with a limited number of assets can be achieved by fixing parameter K in (5.2) to a sufficiently small value.
A different approach for the search of sparse portfolios consists in replacing the l_0 norm in (5.2) with the more tractable l_1 norm [26], or (equivalently) in adding, as a tunable penalty term, the l_1 norm of x to the objective function (5.1), see e.g. [15].

Summarizing, we can say that modern portfolio selection problems involve three partially conflicting objectives:

(a) the risk connected with the portfolio selection ($x'Qx$), to be minimized,

(b) the expected return of the portfolio ($\mu'x$), to be maximized,

(c) the number of assets ($\|x\|_0$) held in the portfolio, to be minimized.

As such, we have different alternatives to model the portfolio selection problem as a single-objective optimization problem.

In this work we focus on the *sparsest portfolio*, that is, on the following nonsmooth optimization problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_0 \\ & \mu'x = \beta \\ & x'Qx \leq \alpha \\ & e'x = 1 \\ & x \geq 0 \end{aligned} \tag{5.3}$$

where β is the desired expected return of the portfolio, and α is the maximum acceptable level of risk.

As shown in [9], problem (5.3) is a difficult, in fact NP-hard, combinatorial problem. Following [61] and [76], we choose to tackle it by replacing the non-smooth objective function $\|x\|_0$ with a suitable smooth concave approximating function. This leads to an "equivalent" (in a sense to be specified later) concave optimization problem. In this way, we move the difficulty of solving (5.3) to that of solving a concave global minimization problem. As a global optimization strategy we adopt the Monotonic Basin Hopping (MBH) method (see, e.g., [56], [45]) employing as local minimization procedure an efficient version of the Frank-Wolfe method [35], useful to solve large dimensional problems.

We observe that the use of the l_1 norm in place of the the objective function $\|x\|_0$ can guarantee the recovery of sparse solutions. However, as shown in [13] and [76], the solutions which are obtained this way, although easily obtainable thanks to the convex nature of the problem, are far less sparse than those obtainable via concave approximations.

In this chapter, we will perform a theoretical analysis aimed to prove the equivalence between a general class of zero norm minimization problems (including (3)) and smooth concave minimization problems; in particular we will prove a new equivalence result which extends previous results to non polyhedral feasible sets. We will also describe the design and implementation of a specific version of the (MBH) global optimization method that can be applied to efficiently solve, as pointed out by the numerical experiments, the class of portfolio selection problems here considered; in particular this version of the MBH method is different from previously known ones as it is specifically tailored to this class of problems: we will design for this problem a specific perturbation operator, which is the crucial part of any MBH algorithm. We will see that this will greatly improve the performance of the method.

5.1 Concave formulations of zero norm minimization problems

In this section we describe an approach for transforming a zero norm minimization problem, which is nonsmooth, into an equivalent (in some sense) smooth concave optimization problem. The approach used here is very general and can be applied not only to the portfolio selection problem, but to any optimization problem which involves the minimization of the zero norm function over a compact set.

Hence, let us consider problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_0 \\ & x \in S \\ & x \geq 0 \end{aligned} \tag{5.4}$$

where we assume that $S \subset \mathbb{R}^n$ is a compact set.

In order to illustrate the idea underlying the concave approach, we observe that the objective function of problem (5.4) can be written as follows

$$\|x\|_0 = \sum_{i=1}^{n} s(|x_i|)$$

where $s : \mathbb{R} \to \mathbb{R}^+$ is the *step function* such that $s(t) = 1$ for $t > 0$ and $s(t) = 0$ for $t \leq 0$. The approach was originally proposed in [61], and is based on the idea of replacing the discontinuous step function by a continuously differentiable concave function $1 - e^{-\alpha t}$, with $\alpha > 0$, thus obtaining a problem of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} (1 - e^{-\alpha x_i}) \\ & x \in S \\ & x \geq 0 \end{aligned} \tag{5.5}$$

It has been shown in [61] that, by assuming that S is a *polyhedral set*, the approximating problem (5.5) is equivalent to the given nonsmooth problem (5.4), that is, for α sufficiently large, there exists a solution of (5.5) which yields a solution of the original problem (5.4).

A similar concave optimization-based approach has been proposed in [90], where the idea is that of using the logarithm function instead of the step function. The adoption of the logarithm function is practically motivated by the fact that, due to the form of the logarithm function, it is better to increase one variable while setting to zero another one rather than doing some compromise between both, and this should facilitate the computation of a sparse solution. This leads to a concave smooth problem of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} \log(\epsilon + x_i) \\ & x \in S \\ & x \geq 0 \end{aligned} \tag{5.6}$$

The equivalence of (5.6) with (5.4), namely that for ϵ sufficiently small there exists a solution of (5.6) which yields a solution of the original problem (5.4), has been proved in [76] under the assumption that S is a polyhedral set.

Here we remove the assumption that S is a polyhedral set, and we study the equivalence between problem (5.4) and a problem of the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} f^u(x_i) \\ & x \in S \\ & x \geq 0 \end{aligned} \tag{5.7}$$

where $f^u : \mathbb{R}^+ \to \mathbb{R}$ is a smooth function depending on a parameter $u \in U \subseteq R$.

To this aim we introduce the following assumptions on the parametrized function f^u. There exists $\bar{u} \in U$ such that, for any infinite sequence $\{u_k\} \to \bar{u}$, we have that:

(i) for each $x_i \geq 0$, $\lim_{k\to\infty} f^{u_k}(x_i)$ is well-defined;

(ii) for each $x_i > 0$, it follows $f^{u_k}(0) < f^{u_k}(x_i)$ and

$$\lim_{k\to\infty} f^{u_k}(0) < \lim_{k\to\infty} f^{u_k}(x_i) < \infty;$$

(iii) for any $\bar{x}_i > 0$, and for any sequence $\{x_i^k\} \to \bar{x}_i$ we have

$$\lim_{k\to\infty} f^{u_k}(x_i^k) = \lim_{k\to\infty} f^{u_k}(\bar{x}_i);$$

(iv) for each $x_i \geq 0$, one of the following conditions holds: either

$$\lim_{k\to\infty} f^{u_k}(x_i) = \begin{cases} 1 & \text{if} \quad x_i > 0 \\ 0 & \text{if} \quad x_i = 0 \end{cases} \tag{5.8}$$

or

$$\lim_{k\to\infty} f^{u_k}(0) = -\infty \tag{5.9}$$

It can be shown that, setting $U = R^+$, we have that assumptions *(i)–(iv)* above are satisfied, for instance:

- by the function $f^u(x_i) = (1 - e^{-ux_i})$, with $\bar{u} = +\infty$;
- by the function $f^u(x_i) = \log(u + x_i)$, with $\bar{u} = 0$.

In particular, the function $f^u(x_i) = (1 - e^{-ux_i})$ satisfies condition (5.8), and function $f^u(x_i) = \log(u + x_i)$ satisfies condition (5.9). Note that whenever condition (5.8) holds we have

$$\lim_{k\to\infty} \sum_{i=1}^{n} f^{u_k}(x_i) = \|x\|_0 \tag{5.10}$$

Let $\{u^k\}$ be any sequence convergent to $\bar{u}$. For each k, let x^k be a solution of (5.7) with $u = u^k$. Thus, by definition, we have for all k and for each $x \in S, x \geq 0$

$$\sum_{i=1}^{n} f^{u^k}(x_i^k) \leq \sum_{i=1}^{n} f^{u^k}(x_i) \tag{5.11}$$

We prove that any limit point of $\{x^k\}$ is a solution of the original problem (5.4). In this way we provide a theoretical justification regarding the transformation of (5.4) into the class of smooth problems defined by (5.7).

Theorem 5.1. *Let $\{u^k\}$ be a sequence such that $\lim_{k\to\infty} u^k = \bar{u}$. Let $\{x^k\}$ be a sequence such that x^k solves problem* (5.7) *with $u = u^k$. Then, the sequence $\{x^k\}$ admits accumulation points, and all of them solve problem* (5.4).

Proof. For all k, x^k solves problem (5.7). Then it follows that $x^k \in S$ and $x^k \geq 0$. Thus, compactness of S implies that $\{x^k\}$ admits accumulation points. Now, let $\bar{x}$ be a limit point of $\{x^k\}$ and $x^\star$ be a solution of (5.4). By compactness of S, we have that $\bar{x} \in S, \bar{x} \geq 0$.

Assume by contradiction that $\bar{x}$ is not a solution of (5.4), that is

$$\|\bar{x}\|_0 \geq \|x^\star\|_0 + 1 \tag{5.12}$$

Consider any $i \in \{1, \dots, n\}$ such that $\bar{x}_i > 0$. From assumption *(iv)* it follows

$$\lim_{k\to\infty} f^{u_k}(\bar{x}_i) = \lim_{k\to\infty} f^{u_k}(x_i^k) = l_i \tag{5.13}$$

where l_i is defined as such the limit value. Then, given any positive ϵ such that $n\epsilon < 1$, two positive integers $k_1(\epsilon)$, $k_2(\epsilon)$ exists such that

$$\begin{aligned} f^{u_k}(\bar{x}_i) &\leq l_i + \frac{\epsilon}{2} && \text{for all } k \geq k_1(\epsilon) \\ f^{u_k}(x_i^k) &\geq l_i - \frac{\epsilon}{2} && \text{for all } k \geq k_2(\epsilon) \end{aligned}$$

Thus, for k sufficiently large, we obtain

$$f^{u_k}(\bar{x}_i) \leq f^{u_k}(x_i^k) + \epsilon \tag{5.14}$$

Now, let us consider any index $i \in \{1, \dots, n\}$ such that $\bar{x}_i = 0$. Using assumption *(ii)* we have, for all k

$$f^{u^k}(\bar{x}_i) \leq f^{u^k}(x_i^k) \tag{5.15}$$

From (5.14) and (5.15) we get that for k sufficiently large we can write

$$\sum_{i=1}^{n} f^{u^k}(\bar{x}_i) \le \sum_{i=1}^{n} f^{u^k}(x_i^k) + n\epsilon \tag{5.16}$$

Conditions (5.11) and (5.16) imply, as $x^\star \in S, x^\star \ge 0$,

$$\sum_{i=1}^{n} f^{u^k}(\bar{x}_i) \le \sum_{i=1}^{n} f^{u^k}(x_i^\star) + n\epsilon \tag{5.17}$$

Now let us distinguish the two cases.

Case I: suppose that condition (5.8) holds. Using (5.10) we have

$$\lim_{k\to\infty} \sum_{i=1}^{n} f^{u^k}(\bar{x}_i) = \|\bar{x}\|_0$$

$$\lim_{k\to\infty} \sum_{i=1}^{n} f^{u^k}(x_i^\star) = \|x^\star\|_0$$

Hence, taking the limits for $k \to \infty$ in (5.17) we obtain

$$\|\bar{x}\|_0 \le \|x^\star\|_0 + n\epsilon$$

From the above relation and (5.12) it follows

$$\|x^\star\|_0 + 1 \le \|x^\star\|_0 + n\epsilon$$

which contradicts the fact that $n\epsilon < 1$.

Case II: suppose that condition (5.9) holds. First we rewrite relation (5.17) as follows

$$\sum_{i:\bar{x}_i>0} f^{u_k}(\bar{x}_i) + (n - \|\bar{x}\|_0) f^{u_k}(0) \le \sum_{i:x_i^\star>0} f^{u_k}(x_i^\star) + (n - \|x^\star\|_0) f^{u_k}(0) + n\epsilon$$

from which we obtain

$$(\|x^\star\|_0 - \|\bar{x}\|_0) f^{u_k}(0) \le \sum_{i:x_i^\star>0} f^{u_k}(x_i^\star) - \sum_{i:\bar{x}_i>0} f^{u_k}(\bar{x}_i) + n\epsilon$$

Taking limits for $k \to \infty$, using (5.12) and condition (5.9) we get that the left member of the above relation tends to $+\infty$, while the right member tends to a finite value (see assumption *(ii)*), thus yielding a contradiction. $\square$

5.1.1 Frank-Wolfe method as local optimizer

In this section we describe an efficient version of the Frank-Wolfe algorithm for minimizing a concave function over a compact convex set and recall some theoretical results about its global convergence (see [74] for further details and proofs). The main motivations for using the Frank-Wolfe algorithm as a local minimizer are the following:

- no need to make a line search when minimizing a concave function over a compact convex set (see theorem 5.2 below);
- possibility to reduce the problem dimension at each step of the algorithm, which leads to significant savings in the computational time (see theorems 5.3 and 5.4 below). This can be seen as a decomposition scheme where, at every iteration, a number of variables are fixed to zero. However, contrary to many classical decomposition schemes, we can prove that there is no need to release the variables that have been fixed in the previous iterations. The particular structure of the problem guaranties that, once a variable has been constrained to zero, the convergence result holds if such variable remains fixed for the rest of the algorithm.

Let us consider the problem

$$\begin{aligned} \min \quad & f(x) \\ & x \in S \end{aligned} \tag{5.18}$$

where $S \subset R^n$ is a non empty compact convex set having the following form

$$S = \{x \in R^n : \mu' x = \beta,\ x'Qx \leq \alpha,\ e'x = 1, x \geq 0\} \tag{5.19}$$

and $f : R^n \to R$ is a concave, continuously differentiable function, bounded below on S. A version of the Frank-Wolfe algorithm with unitary stepsize is described in algorithm 13. The global convergence property of the FW1 algorithm is stated in the following theorem. Proofs of the theorems in this section can be found in [76] [61].

Theorem 5.2. *Let $\{x^k\}$ be a sequence generated by algorithm 13. Then every limit point $\bar{x}$ of $\{x^k\}$ is a stationary point of problem* (5.18).

Now consider the problem

$$\begin{aligned} \min \quad & f(x) = \sum_{j=1}^{n} \phi_j(x_j) \\ & x \in S \end{aligned} \tag{5.20}$$

Algorithm 13: Frank-Wolfe - Unitary Stepsize (**FW1**) Algorithm

```
  Data: A feasible point $x^1$
1 $k \leftarrow 1$;
2 while stopping criterion not satisfied do
3     if $x^k \in \arg\min_{x \in S} \nabla f(x^k)'x$ then
4         STOP;
5     else
6         compute $x^{k+1} \in \min_{x \in S} \nabla f(x^k)'x$;
7         $k \leftarrow k+1$;
8     end
9 end
```

where $\phi_j : \mathbb{R} \to \mathbb{R}$, for $j \in \{1, \ldots, n\}$ are concave, continuously differentiable functions. We assume that f is bounded below on S.

We observe that problem (5.20) includes as special cases the concave programming problems presented in the previous section.

The next theorem shows that, under suitable conditions on the concave functions ϕ_j, the algorithm does not change a variable once that it has been fixed to zero.

Theorem 5.3. *Let $\{x^k\}$ be any sequence generated by algorithm 13. There exists a value M such that, if*

$$\nabla \phi_i(0) \geq M$$

then we have that for $k \geq 1$

$$x_i^k = 0 \quad \text{implies} \quad x_i^{k+1} = 0$$

On the basis of theorem 5.3 it is possible to define algorithm 14, a version of the Frank-Wolfe algorithm with unitary stepsize, where the problems to be solved are of reduced dimension. In particular, whenever a variable is set to zero at an iteration, the method removes this variable for all the following ones. Note that the problem in line (8) is equivalent to a problem of dimension $n - |I^k|$, and that $I^k \subseteq I^{k+1}$, so that the problems to be solved are of nonincreasing dimensions. This yields obvious advantages in terms of computational time. Since algorithm 14 is different from the standard Frank-Wolfe method, its convergence properties cannot be derived from the known result given by theorem 5.2. Next theorem shows the convergence of the algorithm to a stationary point.

Algorithm 14: Frank-Wolfe - Unitary Stepsize - Reduced Dimension (**FW1-RD**) Algorithm

```
Data: A feasible point $x^1$
1  $k \leftarrow 1$;
2  while stopping criterion not satisfied do
3      $I^k \leftarrow \{i \ : x_i^k = 0\}$;
4      $S^k \leftarrow \{x \in S : x_i = 0 \ \forall i \in I^k\}$;
5      if $x^k \in \arg\min_{x \in S} \ \nabla f(x^k)'x$ then
6          STOP;
7      else
8          compute $x^{k+1} \in \min_{x \in S^k} \ \nabla f(x^k)'x$;
9          $k \leftarrow k + 1$;
10     end
11 end
```

Theorem 5.4. *Let $\{x^k\}$ be a sequence generated by algorithm 14. Suppose there exists a value M such that $\nabla \phi_j(0) \geq M$ for $j \in \{1, \dots, n\}$. Then every limit point $\bar{x}$ of $\{x^k\}$ is a stationary point.*

Concerning the separable concave objective functions of problems (5.5), (5.6), we have, for $j \in \{1, \dots, n\}$ and $\alpha, \epsilon > 0$,

- $\phi_j(x_j) = f^\alpha(x_j) = 1 - e^{-\alpha x_j}$ and $\nabla \phi_j(0) = \alpha$
- $\phi_j(x_j) = f^\epsilon(x_j) = \ln(\epsilon + x_j)$ and $\nabla \phi_j(0) = 1/\epsilon$

Therefore, the assumption of theorem 5.4 holds for suitable values of the parameters of the above concave functions, so that algorithm 14 can be applied to solve problems (5.5) and (5.6).

5.2 Global optimization using the Basin Hopping method

In this section we present the global optimization method we used to solve the concave optimization problem. The problem introduced in the previous sections is the minimization of a concave objective function over a compact convex region, that is

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) = \sum_{i=1}^{n} f^u(x_i) \\ s.t. \quad & \mu' x = \beta \\ & x'Qx \le \alpha \\ & e'x = 1 \\ & x \ge 0 \end{aligned} \tag{5.21}$$

where $f^u : \mathbb{R}^+ \to \mathbb{R}$ is a concave function depending on a parameter u and satisfying assumptions *(i)–(iv)* of section 5.1.

Although concave minimization is quite a special case of global optimization, with many important properties which may guide towards designing good optimization algorithm, it remains nonetheless an NP–hard problem. Concavity in particular implies that the global optimum will be located at an extreme point of the feasible region; as we saw, Frank-Wolfe algorithm can be employed to find a stationary point which surely belongs to the frontier of the feasible set. Note that each iteration of Frank-Wolfe algorithm applied to (5.21) requires to solve a problem with linear objective function, one convex quadratic constraint, and linear constraints, and this can be efficiently performed by using modern solvers.

However, local optimization is not enough, as the problem under consideration has, in general, many local optima which are not global. Thus there is a need to employ some global optimization strategy in order to fully exploit the interesting properties of the proposed Frank-Wolfe local method. The most elementary strategy for global optimization is Multistart, which merely consists of repeatedly running a local optimization method from randomly chosen starting points and retaining the best local optimum found. A few experiments within the context of the problem described in this chapter quickly showed that Multistart is too inefficient, leading to an extremely slow and computationally expensive convergence to the global optimum. Our choice was to try a slightly more elaborated method, Monotonic Basin Hopping (MBH) ([56],[45]). This is a simple iterated local search algorithm which consists in applying a perturbation to the current locally optimal solution and starting a local search from the perturbed point. If the local search leads to an improvement, then the current local optimum is updated; otherwise, it is left unchanged and the procedure is repeated until, for a prefixed number of iterations, no improvement is observed. This MBH procedure might be considered as a refined local search method and included in a Multistart framework. Thus, several runs of MBH are performed starting from randomly selected points. In order to define a MBH–based method some procedures have to be defined and some parameters chosen. In particular, we need:

- a procedure $\mathcal{G}()$, which generates a random starting point
- a procedure $\mathcal{P}(x)$, which generates a perturbed solution in a prescribed neighborhood of the current point x
- a procedure $\mathcal{L}(f, X, x)$, which, starting from a point x, produces a local minimum of the objective function f over the feasible set X

Among the parameters to be chosen, most relevant are the number N of Multistart trials to be performed and the number MNI of iterations without improvement after which the current MBH run is stopped. A simplified scheme for Multistart/MBH is described in algorithm 15. The choice of the perturbation

Algorithm 15: Multistart/MBH algorithm

```
1  $f_{best} \leftarrow \infty$;
2  while stopping criterion not satisfied do
3      $x \leftarrow \mathcal{G}()$;
4      $x^\star \leftarrow \mathcal{L}(x, X, f)$;
5      $k \leftarrow 0$;
6      while $k < MNI$ do
7          $y \leftarrow \mathcal{P}(x^\star)$;
8          $y^\star \leftarrow \mathcal{L}(y, X, f)$;
9          if $f(y^\star) < f(x^\star)$ then
10             $x^\star \leftarrow y^\star$;
11             $k \leftarrow 0$;
12         else
13             $k \leftarrow k + 1$;
14         end
15     end
16     if $f(x^\star) < f_{best}$ then
17         $f_{best} \leftarrow f(x^\star)$;
18         $x_{best} \leftarrow x^\star$;
19     end
20 end
```

procedure $\mathcal{P}$ is one of the keys for the success of MBH; in fact choosing too small a neighborhood makes MBH get stuck at a local optimum, wasting local searches. On the other hand, choosing too large a neighborhood makes MBH behave like pure Multistart, thus loosing efficiency.

The most frequently used perturbation consists in choosing a radius r and uniformly generating the perturbed point y in a ball of radius r centered at $x^\star$. In this case it is crucial to choose a radius r which is neither too small nor too large, the ideal being somewhat larger than the radius (i.e., half the diameter) of the region of attraction of the current local optimum $x^\star$. This strategy, with many different choices for the radius r, has been attempted for the concave portfolio optimization problem, but only with very limited success. A possible explanation might be that the local optima obtained through the Frank-Wolfe procedure belong to quite a large basin of attraction which includes large portions of the boundary of the feasible region; thus, usually, a new local optimization from a nearby point will most likely lead to the same local optimum. More refined generation procedures might be tried, exploiting the fact that perturbed points should be preferably located on the boundary but not too close to the current one. However, we chose a more combinatorial perturbation mechanism which turned out to be quite efficient for the problem under consideration. At each iteration we choose to "swap" the values of some variables of the current solution $x^\star$ which are non zero, with other variables which are currently null. This way we performed a perturbation, in some sense, in a space more closely related to the zero–norm. In the experiments we choose to swap $\min\{20, 0.5\|x^\star\|_0\}$ pairs.

For what concerns the other procedures, we choose as $\mathcal{L}$ algorithm 14 described before. Finally, some care had to be taken also for the initial random generation $\mathcal{G}$. In fact the feasible set is the intersection of a simplex with an ellipsoid and with an half–space: generating a uniform point in such a set might not be a trivial task. In our experiments we choose to uniformly generate a point in the unit simplex surface by means of a standard procedure which consists in the uniform generation of $n-1$ points in the interval $[0,1]$: the vector composed of the n lengths of the partition of the unit interval induced by these points turns out to be a uniform point in the n–dimensional simplex surface. After this, feasibility with respect to the other constraints was checked and the whole procedure repeated until a feasible point was eventually produced. Thus the generation tool was a mix of a rigorous generator in the unit simplex coupled with an acceptance/rejection method for the remaining constraints. Apart from pathological cases, this procedure turned out to be quite efficient.

It might be observed that in the generation phase we choose to generate feasible points (and this is motivated from some experiments which showed a significant improvement obtained thanks to feasible point initialization). However, in the perturbation phase we usually obtain an unfeasible starting point (which, in any case, satisfies the unit simplex constraints). This was considered not to be too harmful as, after a perturbation, usually the local optimization algorithm was able to restore feasibility by moving in the neighborhood of the current point.

For what concerns other algorithmic choices, we choose a relatively small value for the MNI parameter (equal to 10) and ran Multistart until either 500 calls to algorithm 14 procedure have been made, or until 1 hour of CPU time has passed.

5.3 Numerical results

The aims of the experiments are the following:

(a) to evaluate the efficiency of the proposed algorithm (called MBH/FW1-RD algorithm) in terms of global optimization and of computational time; to this aim we use a state-of-the-art solver which is able to produce a certificate of optimality.

(b) to assess, on out-of-sample data, the performance of the portfolios obtained from our model and to compare them to classical Markowitz model (5.1).

The results concerning point (a) will be presented in section 5.3, while those regarding point (b) will be shown in section 5.3.

As a concluding remark, we highlight that the original problem (5.3) can be easily formulated as a mixed-integer quadratically constrained programming problem (MIQCP), which can be solved exactly by a number of solvers. Let us consider the following formulation:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, y\in\{0,1\}^n} \quad & \sum_{i=1}^{n} y_i \\ & \mu' x = \beta \\ & x'Qx \le \alpha \\ & e'x = 1 \\ & y_i \ge x_i \quad \forall i \\ & x \ge 0 \end{aligned} \tag{5.22}$$

It can be easily shown that problem (5.3) is equivalent to problem (5.22).

Implementation details and test problems

We considered formulation (5.21) with logarithmic concave function, namely the problem

$$\begin{aligned}\min_{x\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} log(\epsilon + x_i) \\ & \mu' x = \beta \\ & x' Q x \leq \alpha \\ & e' x = 1 \\ & x \geq 0\end{aligned} \tag{5.23}$$

with $\epsilon = 10^{-6}$.

We employed CPLEX 12.0 to solve the subproblem defined at each iteration of the Frank-Wolfe method.

In the computational experiments, we used seven publicly available data sets ([18] and [26]) for mean-variance portfolio optimization. These test problems refer to the following capital market indexes:

- **6FF** [26] ($n = 6$): Six Fama and French (1992) portfolios of firms sorted by size and book-to-market;
- **10IND** [26] ($n = 10$): Ten industry portfolios representing the U.S. stock market;
- **25FF** [26] ($n = 25$): Twenty-five Fama and French (1992) portfolios of firms sorted by size and book-to-market;
- **48IND** [26] ($n = 48$): Forty-eight industry portfolios representing the U.S. stock market.
- **FTSE 100** [18] ($n = 79$): it is a share index published since 1984 of the 100 most highly capitalized UK companies listed on the London Stock Exchange;
- **S&P 500** [18] ($n = 476$): it is a free-float capitalization-weighted index published since 1957 of the prices of 500 large-cap common stocks actively traded in the United States;
- **NASDAQ** [18] ($n = 2196$): it is a stock market index published since 1971 of all of the common stocks and similar securities listed on the NASDAQ stock market (about 3000 components).

For each test problem, we collected the following data:

- Expected returns;

- Covariance matrix.

For the three biggest problems we also have

- Name list of stocks;

- Weekly stock price data from March 2003 to March 2008;

- Weekly stock return data from March 2003 to March 2008;

Note that, expected returns and covariance matrices for problems **FTSE 100**, **S&P 500** and **NASDAQ** are calculated using the stock data from March 2003 to March 2007. The remaining data, for the period April 2007-March 2008, are used as out-of-sample data to evaluate the performance of the portfolios obtained with our method (see below for further details).

For each problem we generated different instances by varying the parameters α and β as follows. For what concerns parameter β, following [18], we define an interval $[\beta_{min}, \beta_{max}]$ and then select N_β values equally spaced in the interval, there including the extreme values. The following table reports, for each problem, the values β_{min}, β_{max} and N_β.

dataset	β_{min}	β_{max}	N_β
6FF	1.0425	1.5693	100
10IND	0.9254	1.1574	100
25FF	1.0466	1.6805	100
48IND	0.9217	1.4463	100
FTSE	$3.4691 \cdot 10^{-3}$	$1.6146 \cdot 10^{-2}$	245
S&P	$2.4970 \cdot 10^{-3}$	$1.6721 \cdot 10^{-2}$	314
NASDAQ	$2.4970 \cdot 10^{-3}$	$3.7239 \cdot 10^{-3}$	22

As regards the choice of parameter α, for each value of β, we select the corresponding value of α as the solution of problem (5.2), where we fix $K = 2$ for dataset **6FF**, $K = 3$ for datasets **10IND** and **25FF**, $K = 8$ for dataset **48IND**. In this way 100 instances were generated for each of these datasets.

For the remaining three problems **FTSE 100**, **S&P 500** and **NASDAQ**, used also to evaluate the out-of-sample performance, for each value of β we select five equally spaced values of α in the interval $[\alpha_\beta^{min}, \alpha_\beta^{max}]$ where:

- α_β^{min} is the minimum risk value, according to the classical Markowitz model, when the expected return is equal to β;

- α_β^{max} is the solution reported in [18] when the expected return is equal to β and $K = 10$, that is a value that allows to obtain solutions with no more than 10 active assets.

This choice seems reasonable since, on the one hand, we are interested in sparse portfolios with a limited risk (as near as possible to the Markowitz lower bound) and, on the other hand, we are interested in comparing the sparsity of our solutions to that of the Markowitz ones. Summarizing, for each of portfolio optimization problems **FTSE 100**, **S&P 500** and **NASDAQ**, a grid of $5N_\beta$ (α, β) pairs was used to generate the test instances of the experimentation.

Global optimization results and comparison with MIQC formulation

In this subsection we compare our method, in term of optimal solution value and of computational time, with a state-of-the art exact solver for MIQCP problems, that is CPLEX 12.0.

First we solved the small dimensional portfolio problems described in [26] and compared the solutions obtained by algorithm MBH/FW1-RD with those obtained by means of the MIQCP solver CPLEX.

Comparing the results, it turned out that the algorithm was generally able to find the global optima. More specifically,

- for three problems over four (namely 6FF, 10IND and 48IND), we obtained the certified optimal solution in all the instances;

- for problem 25FF, we obtained the certified optimal solution in the 80% of the instances, and a slightly less sparse solution than the one found by CPLEX in all the other cases.

These results confirm the good ability of our method in finding the sparsest portfolio. The computational time spent for solving these small problems was very low both for algorithm MBH/FW1-RD and for CPLEX.

Then we considered the difficult portfolio problems described in [18]. For each of the three test problems, we attempted to solve all the instances of (5.22) obtained with the values of parameters α and β as described in subsection 5.3.

CPLEX was able to exactly solve in a reasonable amount of time only the instances of FTSE problem. We have verified that in all instances of FTSE problem the solutions found by CPLEX and MBH/FW1-RD algorithm were the same. This behaviour was also quite robust, as only 4.2% of the local minimizations in the multistart framework returned a solution with a zero norm value worse than the optimal one.

In order to evaluate the efficiency of our solver, we choose to compare the time our algorithm needs to compute the solution with the time spent by CPLEX to solve problem (5.22). However a straightforward comparison would not produce meaningful results, as CPLEX is designed to prove the optimality of the solution, a task which is computationally very expensive and which our algorithm is unable to perform. In order to overcome such difficulty, we adopted the following procedure. For every chosen value of α and β, we execute the algorithm described in section 5.2, and we record the best solution $x^*_{\alpha,\beta}$ found and the time spent to find it. Then we execute the CPLEX solver on the same problem using the formulation described in (5.22). However, we terminate the program execution as soon as the current objective function value becomes equal or less than $\left\|x^*_{\alpha,\beta}\right\|_0$, and we record again the time spent by the algorithm to find such objective function value. We execute this procedure on the FTSE, S&P and NASDAQ problems. At the end of such runs we obtain the time needed by both algorithms in order to find the same objective function value $\left\|x^*_{\alpha,\beta}\right\|_0$ on the same problems. Then we can finally compare the average time spent by the two algorithms which are displayed, in seconds, on the following table

Dataset	Dimension	MBH/FW1-RD	CPLEX integer formulation
FTSE	79	0.24	6.1
S&P	476	80.1	> 1 day
NASDAQ	2196	2860	> 1 day

Clearly MBH/FW1-RD algorithm outperforms CPLEX in terms of computational time. We did not have any exact time for integer formulation on the S&P and NASDAQ problems as, for every tested value of α and β, after 1 day of execution CPLEX was not able to reach the same $K_{\alpha,\beta}$ found by MBH/FW1-RD, so we decided to interrupt its execution. For S&P and NASDAQ problems, we also tried to warm-start CPLEX by supplying it with the best solution found by our method, however it turns out that this technique does not help CPLEX in solving the problem to global optimality.

Result validation on out-of-sample data

In this subsection we evaluate the out-of-sample performance of the sparse portfolios obtained by using the proposed concave programming strategy. Moreover, we compare these solutions with reference to portfolio solutions obtained by using the classical Markowitz model.

First, let us briefly recall how the historical data have been used in the experimentation. For a given set of assets, available historical data refer to a period of 264 weeks extending from March 2003 to March 2008. The covariance matrix Q and the vector of expected returns μ are computed by using the first 212 weeks of the data (the training period) thus leaving out 52 weeks (from April 2007 to March 2008, the so-called out-of-sample data that we reference to as the testing period) that we use for testing. In particular, the out-of-sample data are used to evaluate the obtained sparse portfolios and to compare them against other reasonable ones (e.g. the Markowitz solutions). We assess the quality of a portfolio solution by means of a one-parameter performance measure. More specifically, let $1, \dots, P$ be the weeks of the testing period, and let $\mu_1' x, \dots, \mu_P' x$ be the actual returns obtained by portfolio x. We indicate by $\hat{\mu}$ the average of the returns of the testing period, i.e.,

$$\hat{\mu}(x) = \frac{1}{P} \sum_{k=1}^{P} \mu_k' x$$

and by σ the standard deviation, i.e.,

$$\sigma(x) = \sqrt{\frac{1}{P-1} \sum_{k=1}^{P} (\mu_k' x - \hat{\mu})^2}$$

Then, to evaluate investment x, we employ the Sharpe ratio

$$SR(x) = \hat{\mu}(x)/\sigma(x)$$

which yields a measure of the performance of the portfolio compared to the risk taken (i.e. the higher the Sharpe ratio, the better the performance and the greater the profits for taking on additional risk).

Let us denote, for a given value of β such that (5.1) is feasible, by x^β the solution of problem (5.1), i.e. the classical Markowitz problem. Furthermore, for given values of the parameters α and β defining the risk and return constraints, respectively, let us denote by $x^*_{\alpha,\beta}$ the approximate solution of problem (5.4) determined by the proposed method. In the following we make a comparison between the solutions x^β of the Markowitz problem for varying values of parameter β and our corresponding sparse solutions $x^*_{\alpha,\beta}$ for varying values of the risk parameter α.

In Figure 5.1, for each dataset and for each $\left\|x^\beta\right\|_0$, we report by a squared marker the Markowitz solution which yields the highest SR on the testing period. Similarly, in Figure 5.1 we report, for each dataset and for each value of $\left\|x^*_{\alpha,\beta}\right\|_0$ obtained by our method, the solution which yields the highest SR on the testing period. The figures show that for small and medium size datasets, i.e. datasets FTSE and S&P, our solutions and those provided by the Markowitz model are comparable in terms of number of active assets. As for the quality of the solutions, measured by the Sharpe ratio, we note that our solutions are comparable with the Markowitz ones on the small dataset but perform better on the medium one. On the large dataset considered, NASDAQ, our solutions clearly outperforms the Markowitz ones both in terms of achieved sparsity and of portfolio quality as measured by the SR coefficient.

The computational results show that the proposed approach provides sparse and efficient portfolios for suitable values of the expected return and allowable risk. Therefore, it may be a useful tool for financial experts, who are responsible for selecting appropriate values of β and α to obtain a sparse. portfolio yielding good performance on out-of-sample data.

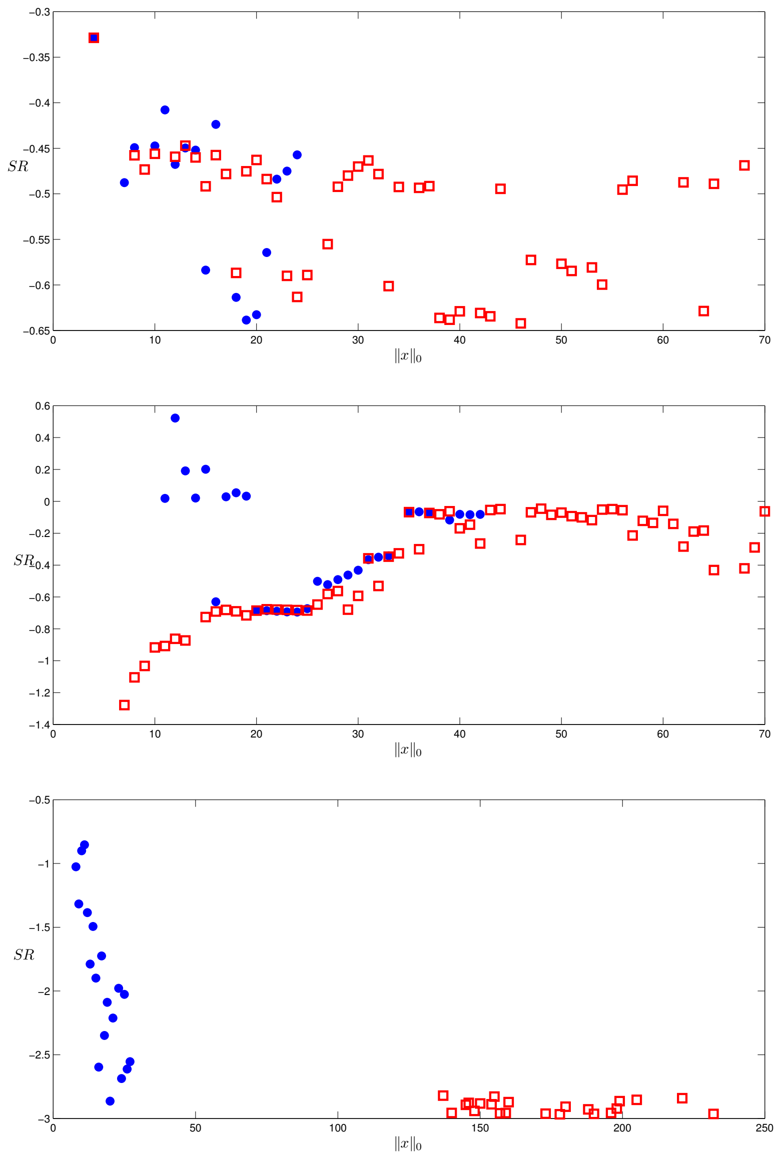

Figure 5.1: Out-of-sample comparison on the datasets (top to bottom) FTSE, S&P and NASDAQ between our solutions, denoted by dotted markers, and those provided by the classical Markowitz model, denoted by squared markers.

Chapter 6

Low-thrust space trajectory optimization

The application of global optimization techniques to the design of interplanetary trajectories has received quite some attention in the last years as it becomes increasingly evident that such a framework introduces a high level of automation in a process that is otherwise still heavily relying on expert engineering knowledge. The systematic study of global optimization algorithms in relation to chemically propelled spacecrafts [28, 66, 49, 87, 85, 3] has proved that efficient computer algorithms are able to produce, for these types of spacecrafts, competitive trajectory designs. Thanks to initiatives such as the Global Trajectory Optimization Competition (GTOC) [48] and the Global Trajectory Problem database (GTOP) [50] of the European Space Agency, the attention of communities not traditionally linked to aerospace engineering research [1, 17, 78] has increased bringing a beneficial influx of new ideas and solutions thus advancing the field considerably. While for problem formalizations such as the MGA (Multiple Gravity Assist) and the MGA-DSM (Multiple Gravity Assist with Deep Space Maneuver) [87], the advantages of using these techniques has been proved, no convincing results have been produced so far [2] in the case of the LT-MGA problem (Low-Thrust Multiple Gravity Assist). The optimization problem of simple low-thrust trajectories can be solved efficiently by local optimization methods. However, on a large design space, local methods converge to suboptimal solutions or sometimes fail to converge if a good starting guess is not provided. On the other hand, global methods fail to provide a good solution because of the complexity of the resulting nonlinear programming problem that, unlike the box-constrained MGA and MGA-DSM, has to deal with a high number of nonlinear constraints if an accurate spacecraft dynamics has to be accounted for. Constraints in the optimization problem can be handled as an extra penalty term in the objective function [92]. However a suitable value of the weighting factor on the penalty is unknown beforehand. A bad choice on the weighting factor leads to premature convergence on the objective or to infeasible solutions. In this chapter we build on some recent work [92, 86] and we present a global optimization framework for the LT-MGA problem where nonlinear constraints handling is incorporated in the main global optimization loop via the algorithm hybridization with a local method. The resulting algorithm explore efficiently the vast solution space of low-thrust trajectories thus producing a convincing case for using global optimization techniques also in relation to the LT-MGA problem.

6.1 Single leg problem

Before deriving any model, we first need to consider the problem of Keplerian orbital propagation. In other words we need to compute, given a starting position and velocity, the position and velocity of a point mass subject to a central force field describing the gravitational attraction to a massive object located in the origin of the coordinate system. Although the shape of the resulting trajectory in known to be a conic, the analytical relation between position, velocity and time is more complex and requires the solution of a transcendental equation equation called the Kepler's equation. Different methods exist, and a description of those is outside the scope of this thesis. Many known algorithms first compute if the orbit is elliptic or hyperbolic, and then execute slightly different instructions to handle the two cases. The parabolic trajectory is usually not handled explicitly and is considered a degenerate case. However, if the starting conditions are such that the trajectory is nearly parabolic, such methods often present numerical difficulties, and the loss of accuracy strongly affects the behaviour of the solvers. Here we will briefly describe here a technique which uses a reformulation of the problem, called the universal variable formulation, that allows us to treat in the same way the parabolic, hyperbolic and elliptic cases, and prevents such numerical problems from arising in the first place. For more information about the method, we refer to [24]. The algorithm is also efficient, but somewhat at the expense of clarity, as many of the variables used inside the method do not represent any obvious physical property.

Let $r_0 \in \mathbb{R}^3$ and $v_0 \in \mathbb{R}^3$ be the starting position and velocity of the orbiting body. Let μ be the gravitational parameter of the central body

$$\mu = G\,M$$

where $G = 6.67384(80)\,10^{-11}\,\mathrm{m}^3\mathrm{kg}^{-1}\mathrm{s}^{-2}$ is the gravitational constant, and M is the mass of such body. Our goal is to compute the position $r_t \in \mathbb{R}^3$ and the velocity $v_t \in \mathbb{R}^3$ of the orbiting body after t units of time have passed. First, we compute α as

$$\alpha = \frac{2}{\|r_o\|} - \frac{\|v_0\|^2}{\mu}$$

The sign of α gives the shape of the trajectory, which is hyperbolic if $\alpha < 0$, parabolic if $\alpha = 0$ and elliptic if $\alpha > 0$. In the latter case, the ellipsis semimajor axis is the inverse of α.
Then we find a value of θ such that

$$\frac{r_0' v_0 \theta^2 C(\alpha\theta^2)}{\sqrt{\mu}} + (1 - \alpha\|r_0\|)\,\theta^3 S(\alpha\theta^2) + \|r_0\|\theta = \sqrt{\mu}t \tag{6.1}$$

where $S(z)$ and $C(z)$ are the Stumpff functions, defined as

$$S(z) = \begin{cases} (\sqrt{z} - \sin(\sqrt{z}))\,z^{-\frac{3}{2}} & \text{if } z > 0 \\ \left(\sinh(\sqrt{-z}) - \sqrt{-z}\right)(-z)^{-\frac{3}{2}} & \text{if } z < 0 \\ \frac{1}{6} & \text{if } z = 0 \end{cases}$$

$$C(z) = \begin{cases} \frac{1-\cos(\sqrt{z})}{z} & \text{if } z > 0 \\ \frac{\cosh(\sqrt{-z})-1}{z} & \text{if } z < 0 \\ \frac{1}{2} & \text{if } z = 0 \end{cases}$$

Next, we compute the coefficients F and G as

$$F = 1 - \frac{\theta^2 C(\alpha\theta^2)}{\|r_0\|} \qquad G = t - \frac{\theta^3 S(\alpha\theta^2)}{\sqrt{\mu}}$$

and find r_t as

$$r_t = F r_0 + G v_0$$

Similarly, we compute coefficients F_t and G_t as

$$F_t = \sqrt{\mu}\theta \frac{\alpha\theta^2 S(\alpha\theta^2) - 1}{\|r_0\|\|r_t\|} \qquad G_t = 1 - \frac{\theta^2 C(\alpha\theta^2)}{\|r_t\|}$$

to finally express v_t as

$$v_t = F_t r_0 + G_t v_0$$

By using the equations above, we can define the propagation function $\mathrm{Prop}_\mu : \mathbb{R}^7 \to \mathbb{R}^6$ that, given the initial conditions, propagates a body for a given period of time under the influence of a mass with gravitational parameter μ. Then, $[r_t, v_t] = \mathrm{Prop}_\mu(r_0, v_0, t)$. Such function cannot be computed analytically because neither equation (6.1) can. However, equation (6.1) is numerically well behaved and can easily be solved to machine precision in a few steps of the Newton algorithm. Apart from the degenerate points where $r_0 = 0$ or $r_t = 0$, function $\mathrm{Prop}_\mu(r_0, v_0, t)$ is smooth. When such degenerate cases occur, the function is undefined. However this function will be used to propagate a vehicle in the solar system, and if either $r_0 = 0$ or $r_t = 0$, the spacecraft must be located at the center of the Sun, so we will treat those cases with generous disregard. Finally, we note that this function can be used to propagate any body in the solar system, be it a spacecraft or a planet.

To show a simple usage of the Prop_μ function, we can write a very simple model, in which a spacecraft travels from the Earth to Mars, and is allowed to thrust the engines at departure and at arrival. A typical objective function we wish to minimize is the total fuel usage, or some function which is correlated with fuel consumption. Let r_0^E, v_0^E, r_0^M and v_0^M be the Earth and Mars position and velocities at a given epoch t_0. We introduce the variables

- $t_1 \in \mathbb{R}$, the time of departure as the number of units of time from t_0
- r_1^E, v_1^E, r_1^S and v_1^S, all belonging to $\mathbb{R}^3$, the positions and velocities of the Earth and the spacecraft at time t_1
- $t_2 \in \mathbb{R}$, the time of arrival as the number of units of time from t_0
- r_2^M, v_2^M, r_2^S and v_2^S, all belonging to $\mathbb{R}^3$, the positions and velocities of Mars and the spacecraft at time t_2

The model can be written as

$$\begin{aligned}
\min\ & \|v_1^S - v_1^E\| + \|v_2^S - v_2^M\| \\
& [r_1^E, v_1^E] = \mathrm{Prop}_\mu(r_0^E, v_0^E, t_1) \\
& [r_2^M, v_2^M] = \mathrm{Prop}_\mu(r_0^M, v_0^M, t_2) \\
& [r_2^S, v_2^S] = \mathrm{Prop}_\mu(r_1^S, v_1^S, t_2 - t_1) \\
& r_1^S = r_1^E \\
& r_2^S = r_2^M \\
& t_2 \geq t_1
\end{aligned} \tag{6.2}$$

The first three equations enforce the gravitational laws on the Earth, Mars, and on the spacecraft. The other two equalities constrain the spacecraft to start at the Earth and end on Mars, while the last one prevents from doing the trip backwards, from Mars to the Earth. We minimize the total difference between the spacecraft and planets velocity, because we are assuming that the fuel usage is proportional to the change in velocity of the body. This assumption does not always holds. For example in solar powered engines the efficiency depends also from the distance to the Sun. On the other hand some thrusters are powered by the heat produced from radioactive materials, so the thrust depends on the time elapsed from the start of the mission, since such materials slowly decay over time. In any case, it is straightforward to write the fuel usage as a function of the change of velocity and, possibly, other variables, and plug the correct equations into the objective function. Finally, we note that for a working implementation, many variables might be eliminated by substitution to reduce the problem size, in particular the position and velocity of the planets could be computed and substituted from the first two constraints.

6.2 Extensions

Model (6.2) can easily be extended. It is trivial to enforce simple constraints like bounds on the departure or arrival times. Additional constraints on the departure can be easily imposed. For example the angle between v_1^S and the Earth rotation plane could be required to be small, as the launch systems always exploit the increased initial speed given for free by the Earth's rotation.

6.2.1 Multiple legs and deep space maneuvers

The main limitation of model (6.2) is that it forbids the engine from thrusting during the flight. To allow this kind of deep space maneuvers, additional variables must be introduced, regarding the position and velocities of the spacecraft when lighting the engines. We will call a leg the fraction of the trajectory between two planets, and a segment a fraction of the leg, during which the spacecraft can perform a space maneuver. A leg can be composed of more than one segment. Inside a leg, between two deep space maneuvers, no force, apart from the gravitational pull of the Sun, affects the spacecraft. Rewriting the previous model to account for this case we obtain

$$\begin{aligned}
&\min \left\|v_1^S - v_{1^+}^E\right\| + \sum_{i=1}^{n} \left\|v_{i^+}^S - v_{i^-}^S\right\| + \left\|v_{(n+1)^-}^S - v_{n+1}^M\right\| \\
&\quad [r_1^E, v_1^E] = \mathrm{Prop}_\mu(r_0^E, v_0^E, t_1) \\
&\quad [r_{n+1}^M, v_{n+1}^M] = \mathrm{Prop}_\mu(r_0^M, v_0^M, t_{n+1}) \\
&\quad [r_{i+1}^S, v_{(i+1)^-}^S] = \mathrm{Prop}_\mu(r_i^S, v_{i^+}^S, t_{i+1} - t_i) \quad \forall i \in \{1, \dots, n\} \\
&\quad r_1^S = r_1^E \\
&\quad r_{n+1}^S = r_{n+1}^M \\
&\quad t_{i+1} \geq t_i \quad \forall i \in \{1, \dots, n\}
\end{aligned} \tag{6.3}$$

For every deep space i maneuver we introduce variables

- $t_i \in \mathbb{R}$ is the time the deep space maneuver is performed
- $r_i \in \mathbb{R}^3$ is the position of the spacecraft at the time the deep space maneuver is performed
- $v_{i^-} \in \mathbb{R}^3$ is the velocity of the spacecraft just before the deep space maneuver is performed
- $v_{i^+} \in \mathbb{R}^3$ is the velocity of the spacecraft immediately after the deep space maneuver is performed

6.2.2 Planetary swingby

A planetary swingby happens when the spacecraft gets so close to a planet that the planet's gravitational pull becomes much stronger than the Sun's, and such force can be used to alter the speed direction. Since the amount of time spent in the neighbourhood of a planet is so small with respect to the total time of a mission, we can consider that in our model swingbys happen instantly. At a given time t, let r^P and v^P be the planet position and velocity, r^S be the spacecraft position, v_-^S be the spacecraft velocity just before the swingby and v_+^S be the spacecraft velocity just after the swingby. The following constraints must be imposed

$$r^P = r^S \tag{6.4}$$

$$\left\|v^P - v_-^S\right\| = \left\|v_P - v_+^S\right\| \tag{6.5}$$

Equation (6.4) is obvious and forces the spacecraft and planet positions to coincide at the moment of the swingby. Equation (6.5) derives form the fact that gravity is a conservation law, and it cannot be used to gain energy with respect to the attractor body. The only reason why planets can actually be used to change the velocity of the spacecraft with respect to the Sun is that planets are not stationary. Under the assumption that planets are point mass objects, such equations describe all the possible swingbys that can happen. In practice however many of such orbits must pass so close to the center of the planet that the spacecraft trajectory would intersect the planet sphere. Assuming that equation (6.5) holds, the angle θ between v_-^S and v_+^S and the minimum distance from the center of the planet $r^{\min}$ are related by the following equation

$$\theta = 2 \arcsin \left(\frac{1}{1 + r^{\min}\left\|v_P - v_-^S\right\|^2 / \mu^P} \right) \tag{6.6}$$

where μ^P is the gravitational constant of the planet. $r^{\min}$ must obviously be greater than the radius of the planet, plus a safety margin.

6.3 Low thrust engine

The trajectory model that is to be used to transcribe an LT-MGA trajectory optimization into a nonlinear programming problem (to be solved by global optimization methods) is crucial to the success of the overall algorithm one wants to produce. Criteria to be accounted for include accuracy in the description of the spacecraft dynamics, computational efficiency in the objective function and constraints evaluation, problem dimension and number of nonlinear constraints produced. Bearing these issues in mind, we propose to use a version of the trajectory model proposed by Sims and Flanagan [79]. Figure 6.1 briefly illustrates such a trajectory model. A trajectory is divided into legs which begin and end with a planet. Low-thrust arcs on each leg are modeled as sequences of impulsive maneuvers $\Delta \mathrm{V}_i$ connected, using the Prop_μ function, by conic arcs.

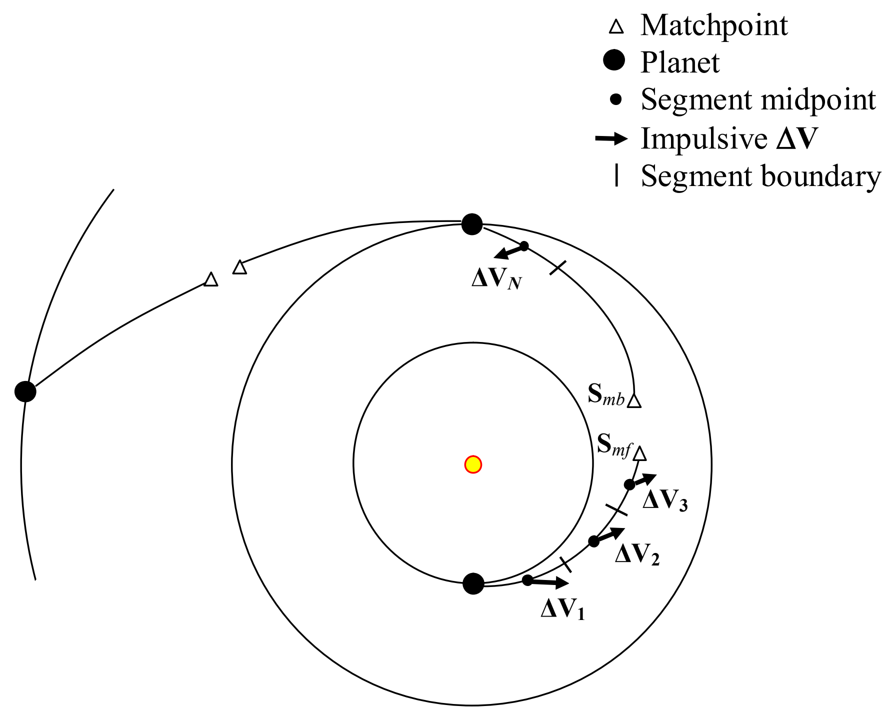

Figure 6.1: Impulsive ΔV transcription of a low-thrust trajectory, after Sims and Flanagan [79]

We denote the number of impulses for a given arc (which is the same as the number of segments) with N. The ΔV_i at each segment, equal to $\left\| v^S_{i^-} - v^S_{i^+} \right\|$ should not exceed a maximum magnitude, ΔV_{max}, where ΔV_{max} is the velocity change accumulated by the spacecraft when it is operated at full thrust during that segment

$$\Delta V_{max} = (T_{max}/m)(t_f - t_0)/N \tag{6.7}$$

where T_{max} is the maximum thrust of the low-thrust engine, m is the mass of the spacecraft, t_0 and t_f is the initial and final time of a leg.

The spacecraft mass is propagated using the Tsiolkovsky rocket equation [84]

$$m_{i+1} = m_i e^{(-\Delta V_i/(g_0 I_{sp}))} \tag{6.8}$$

where the subscript i is used to refer to the mass and ΔV on the i-th segment, g_0 is the standard gravity (9.80665 m/s^2), and I_{sp} is the specific impulse of the low-thrust engine.

At each segment, trajectory is propagated using the Prop_μ equation as seen in section 6.2.1. Swingbys maneuvers are modeled using the equations in section

6.2.2.

6.3.1 Optimization problem

Using all the elements previously introduced, we can finally write an optimization model for a low thrust trajectory with multiple swingbys. Let L be the number of legs, and let S_l be the number of segments of the l-th leg. For every leg, the following variables are used

- $r_l^i \in \mathbb{R}^3$, $r_l^f \in \mathbb{R}^3$ are the Cartesian positions of the spacecraft at the beginning and at the end of the leg.
- $v_l^i \in \mathbb{R}^3$, $v_l^f \in \mathbb{R}^3$ are the spacecraft velocity at the beginning and at the end of the leg.
- $m_l^i \in \mathbb{R}$, $m_l^f \in \mathbb{R}$ are the spacecraft mass at the beginning and at the end of the leg.
- t_l is the time spent in leg l.
- $\Delta v_{l,s} \in \mathbb{R}^3$, for every segment s in the leg, is the change in velocity provided by the spacecraft engine.
- Except for the last leg, where no swingby is executed, r_l^{swingby} is the minimum distance from the spacecraft to the center of the planet during the maneuver.

For convenience in the model, we define a function Propleg_μ that propagates, in sequence, a spacecraft through all the segments which compose a leg. Propleg_μ takes as arguments the starting position, velocity and mass, the total time t of the leg, one $\Delta v \in \mathbb{R}^3$ vector for each segment, and computes the final position, velocity and mass of the spacecraft after the leg. Starting from the initial position, for every segment we must propagate the body up to the middle, then apply a DSM and propagate for the remaining time. Let r, v, m denote the current position, velocity and mass. Those variables are initially set to the starting values. For every segment, which lasts for an amount of time equal to t/S_l

1. We propagate the spacecraft up to half the segment time, using $[r, v] = \text{Prop}_\mu(r, v, t/(2S_l))$. The mass m is kept constant.
2. We instantaneously apply the DSM. This does not change the spacecraft position r. The velocity is modified by $v = v + \Delta v_i$, where Δv_i is the

change in velocity for segment i. The mass is updated using equation (6.8) to account for the spent fuel.

3. We propagate the spacecraft to the end of the segment, using again equation $[r, v] = \text{Prop}_\mu(r, v, t/(2S_l))$.

Finally, the updated values of r, v, m are returned. As an implementation related note, the Prop_μ function satisfies property 6.1, which can be used to reduce the number of times the Prop_μ function is computed from $2S_l$ to $S_l + 1$ times, by computing simultaneously the last half of every segment with the first half of the following one.

Property 6.1. *Let $r_1, r_2, r_3, v_1, v_2, v_3$ all belonging to $\mathbb{R}^3$, and let $t_1, t_2 \in \mathbb{R}^+$. Then if*

$$[r_2, v_2] = \text{Prop}_\mu(r_1, v_1, t_1)$$

$$[r_3, v_3] = \text{Prop}_\mu(r_2, v_2, t_2)$$

it holds

$$[r_3, v_3] = \text{Prop}_\mu(r_1, v_1, t_1 + t_2)$$

Note that the Propleg_μ function does not limit the thrust of the spacecraft. For low thrust engines this is a very unrealistic assumption, so the following additional constraint must be imposed for every segment

$$\|\Delta v_i\| \leq \frac{T_{\max} t}{m_i N} \tag{6.9}$$

where $T_{\max}$ is the maximum power of the engine, t is the total time of the leg, m_i is the current mass of the spacecraft during the i-th segment and N is the number of segments for the leg. Such constraints are cheaper to compute while executing the Propleg_μ function, as this avoids to evaluate twice the values of m_i.

Our objective is to minimize the total fuel consumption or, equivalently, to maximize the final mass of the spacecraft.

$$\max m_f \tag{6.10}$$

subject to, for every leg l

$$[r_l^f, v_l^f, m_l^f] = \text{Propleg}_\mu(r_l^i, v_l^i, m_l^i, t_l, \Delta v_{l,1}, \dots, \Delta v_{l,S_l}) \tag{6.11}$$

$$\|\Delta v_{l,s}\| \leq \frac{T_{\max} t_l}{m_{l,s} S_l} \qquad \forall s \in \{1, \dots S_l\} \tag{6.12}$$

$$r_{l+1}^i = r_l^f = r_l^P \tag{6.13}$$

$$m_{l+1}^i = m_l^f \tag{6.14}$$

$$\left\|v_l^f - v_l^P\right\| = \|v_{l+1}^i - v_l^P\| \tag{6.15}$$

$$\arccos\left(\frac{(v_l^f - v_l^P)'(v_{l+1}^i - v_l^P)}{\left\|v_l^f - v_l^P\right\|\|v_{l+1}^i - v_l^P\|}\right) = 2\arcsin\left(\frac{1}{1 + r_l^{\text{swingby}}\left\|v_l^P - v_l^f\right\|^2/\mu_l^P}\right) \tag{6.16}$$

$$r_l^{\text{swingby}} \geq r_l^{\min} \tag{6.17}$$

Equation (6.11) imposes the trajectory constraints from the beginning to the end of every leg, equation (6.12) forbids the engine to thrust more than its maximum power allows. Equations (6.13) and (6.14) simply state that at every swingby the position of the spacecraft must coincide with the one of the planet, and that the spacecraft mass is not affected by the maneuver. Equations (6.15)–(6.17) put a constraint of the swingby, and can be disregarded for the last leg, if no swingby is performed. Additional mission dependant constraints must be usually imposed, such as limits on the times, and constraining the spacecraft starting velocity with respect to the Earth to the maximum escape velocity that the launcher is able to provide.

6.4 Global optimization algorithms

The proposed transcription of the LT-MGA problem is a continuous, constrained, nonlinear optimization problem. Since such class of problems usually present local minimizers that are not global, they are often unsolvable using only local optimization algorithms. Practical experience shows that this is often the case with trajectory optimization problems, regardless of the propulsion type. Thus, local solvers must be used inside a global optimization strategy in order to achieve solutions which are as close as possible to the optimal ones. Here we describe three different approaches that have been tried on such problem, one of which, the Basin Hopping algorithm, have already been used in section 5.2. Let us first define some procedures that the algorithms will use.

- $\mathcal{G}()$ is a procedure that randomly generates a starting point. Ideally, we would like the point to be uniformly distributed on the feasible region, but since our problem's feasible set is of a very small size, and generating a point inside such region is a hard problem by itself, we used a procedure that uniformly generates points into a reasonable box containing the feasible region.

- $\mathcal{S}(x)$ is a procedure that, given a point x, computes a local minimizer of the objective function, taking x as an initial guess. For our tests we will use a software package called SNOPT [39] [38] for our tests, which is based on sequential quadratic programming (SQP).

- $Best(x, y)$ is a procedure that, given two solutions x and y, returns the best one according to a fixed rule. Since we are dealing with a constrained optimization problem, $Best(x, y)$ chooses the point with the lower constraint violation norm value. In case both points are feasible, then the point with the lower objective function value is chosen instead.

The first and most simple algorithm is called Multistart (MS), which directly optimizes the points obtained by a generator. Algorithm 16 describes Multistart. Although for N big enough and reasonable choices of $\mathcal{G}$, $\mathcal{S}$ and

Algorithm 16: Multistart

```
1 $x^\star \leftarrow \mathcal{G}()$;
2 for $i \in \{1, \ldots, N\}$ do
3     $x \leftarrow \mathcal{G}()$;
4     $y^\star \leftarrow \mathcal{S}(x)$;
5     $x^\star \leftarrow Best(x^\star, y^\star)$;
6 end
```

$Best$, Multistart will eventually converge to the global solution, the extremely slow convergence rate renders this algorithm unfit for the vast majority of global optimization problems. However, because of its simplicity, this algorithm can be effectively used as a baseline to compare other solvers to.

The second algorithm is called Basin Hopping (BH), or Iterated Local Search [58]. In addition to the procedures previously used by Multistart, Basin Hopping needs a procedure $\mathcal{P}(x)$ that, given a point x, returns another point randomly generated in a conveniently defined neighborhood of x. BH is described in algorithm 17. In other words, after a local optimization, instead of generating

Algorithm 17: Multistart Basin Hopping

```
1  $x_{best} \leftarrow \mathcal{G}()$;
2  for $i \in \{1, \dots, N\}$ do
3      $x \leftarrow \mathcal{G}()$;
4      $x^\star \leftarrow \mathcal{S}(x)$;
5      $k \leftarrow 0$;
6      while $k < MNI$ do
7          $y \leftarrow \mathcal{P}(x^\star)$;
8          $y^\star \leftarrow \mathcal{S}(y)$;
9          if $Best(y^\star, x^\star) = y^\star$ then
10             $x^\star \leftarrow y^\star$;
11             $k \leftarrow 0$;
12         end
13         else
14             $k \leftarrow k + 1$;
15         end
16     end
17     $x_{best} \leftarrow Best(x_{best}, x^\star)$;
18 end
```

a new point inside the whole feasible region like in Multistart, we restrict the generation inside a small region centered on the current best point. If, as it happens with real life problems, the good solutions are clustered together, there is a good chance that the series of perturbations and re-optimizations will lead to the best solution contained inside the cluster [88] [89] [56] [44]. A new point is then generated from the whole feasible set only when no improvement has been made for a number of times equal to a fixed parameter called Max No Improve (MNI), which has been set to 500 during our runs. The choice of the perturbation function usually determines the Basin Hopping performance. A typical rule is to choose a new point inside a small box or sphere centered on the current point, but problem knowledge, if available, can be used to better tune the perturbation. As an example, when only two different planets are considered, we can expect that by shifting a solution in time by a period equal to the synodic period of such two planets, solutions that are similar (and maybe better) than the current one could be found. So the perturbation rule we used is composed by the following two steps

1. for each variable x_i and its lower and upper bounds l_i and u_i, add to x_i a value uniformly chosen in the interval $[-r(u_i - l_i),\, r(u_i - l_i)]$ with a small given value of r.

2. with a low probability p, shift the solution in time, either forward or

backward with equal probability, by a time length equal to the synodic period.

In our experiments, we used $r = 0.05$ and $p = 0.1$.

Finally, the Simulated Annealing (SA) with Adaptive Neighborhood has been used. The algorithm structure is similar to Basin Hopping, with some important differences in key parts of the algorithm, which are quickly described here. For a more complete description we refer to [54] [23].

First of all, the comparison in line 9 is modified to allow for non monotonicity. The problem is transformed to an unconstrained optimization problem by using a penalty function to account for constraints violations. Then, $Best(x, y)$ returns x with probability 1 if $f(x) \leq f(y)$, or with probability $e^{(f(y)-f(x))/T}$ if $f(x) > f(y)$. T is the temperature parameter, which is exponentially decreased every fixed number of iterations.

Furthermore, the perturbation procedure $\mathcal{P}$ is adaptive, meaning that the radius of the perturbation is adjusted at every iteration. Such r value is increased each time the generated point is accepted, and decreased each time the point is refused.

Finally, the local optimization $\mathcal{S}$ is not executed at each step of the algorithm, but just once at the end, starting from the point returned by the Simulated Annealing with Adaptive Neighborhood.

In our tests, the temperature parameter T starts at 10 and is multiplied by 0.8 every 100 iterations. The starting perturbation radius r^0 is equal for each variable to 0.05 times the width of the box, and every K iterations a new value r^{k+1} is obtained from r^k using the following rule. Let ρ be the number of accepted steps in the last K iterations, divided by K. Then

$$r^{k+1} = \begin{cases} r^k \left(1 + \dfrac{\rho - 0.6}{2}\right) & \texttt{if } \rho > 0.6 \\ \dfrac{r^k}{1 + \frac{0.4-\rho}{2}} & \texttt{if } \rho < 0.4 \\ r^k & \texttt{otherwise} \end{cases}$$

We used $K = 10$, and the maximum number of iterations was set equal to 2000.

All the code was written in C++, and compiled with gcc on a GNU/Linux environment.

6.5 Numerical results

6.5.1 Nuclear electric propulsion mission to Jupiter

We demonstrate the application of our global optimization framework to perform preliminary design of trajectories for a mission that employs nuclear electric propulsion (see Table 6.1). The spacecraft is assumed to have a thruster

Table 6.1: Parameters for a Nuclear Electric Propulsion Mission

Parameters	Values
Initial mass of the spacecraft	$20,000$ kg
Maximum thrust	2.26 N
Specific impulse	$6,000$ s
Launch date	Jan. 1, 2020 – Jan. 1, 2030
Launch V_∞	≤ 2.0 km/s
Maximum time of flight	10 years
Minimum flyby radius	$7,000$ km

with a constant maximum thrust and a constant specific impulse, with similar hardware parameters to the Jupiter Icy Moons Orbiter [41, 91, 96, 95, 70] (a canceled mission originally proposed by NASA in 2003). In our example scenario, we consider a planetary encounter sequences of Earth-Earth-Jupiter (i.e., one Earth flyby) which rendezvous at Jupiter.

Our approach begins with the three global optimization algorithms solving an Earth-Earth-Jupiter rendezvous problem which maximizes the final mass (see equations (6.10)–(6.17)). We use ten segments ($N = 10$) for each leg and the dimension of this problem is 75 with 35 nonlinear constraints (as we choose Cartesian coordinates to encode the velocities, the constraints on the ΔV magnitude are quadratic and thus increase the number of nonlinear constraints). We first let the global optimizers run for a fixed time (∼5 hours for each algorithm) on a PC (AMD Turion@2.1 GHz with 3GB of RAM), which produced hundreds of trajectories. Then we select feasible solutions for which the norm of the constraint violation vector is less than 10^{-6}. Table 6.2 summarizes the results found by the three global optimizers. We notice that many solutions found by Simulated Annealing and Multistart fail to converge and therefore the number of solutions is less than Basin Hopping. Figure 6.2 plots the comparison of the three algorithms, which shows that results found by Basin Hopping always have higher final mass than the other two methods, while Simulated Annealing and Multistart have similar performance.

Figure 6.3 plots the x-y projection of a trajectory found by Basin Hopping

Table 6.2: Algorithm Statistics for the E-E-J Mission

Algorithm	Basin Hopping	Simulated Annealing	Multistart
Best (kg)	17,102	17,019	16,961
Worst (kg)	10,913	11,924	11,900
Mean (kg)	16,235	16,302	15,930
Mean best 10 (kg)	17,039	16,912	16,715
Std (kg)	1,345	1,160	1,320
No. of sol. above 95% best (16,246 kg)	137	19	11
No. of sol. above 90% best (15,391 kg)	155	22	14
Total no. of solutions	183	24	18

with the highest final mass. On the plot, the solid and dash curves represent thrusting and coasting segments, respectively; while a ΔV is shown as an arrow in the midpoint of a segment. In this example, the spacecraft leaves the Earth on November 12, 2021 with a V_∞ of 2 km/s. It enters an 4:3 resonance orbit with a period of $\sim$ 1.3 years and goes around the Sun for 3 revs, before it flybys the Earth after 3.8 years with an increased V_∞ of 8.5 km/s. After the gravity assist at the Earth, its aphelion increases for the transfer to Jupiter. After 7.4 years of interplanetary flight, the spacecraft rendezvous at Jupiter on March 24, 2029 with a final mass of 17,102 kg.

Besides the value of the objective function (final mass), it is also interesting from a mission design point-of-view that the optimization process is able to find trajectories that launch on different dates. In our example in Fig. 6.4, the difference in the final mass is less than 200 kg (or 1% of the initial mass) for most launch periods. The 1% penalty of the final mass gives flexibility to the mission designer to choose a different date in case there is a change in the mission. The process is also able to locate various locally optimal trajectory families, which is interesting from an astrodynamics point-of-view. From Fig. 6.5, we note that the 'clusters' of solutions belong to different Earth-Earth resonance transfer orbit. For example, 1:1 resonance with flight time $\sim$400 days, 2:3 resonance with flight time $\sim$800 days, and 3:2 resonance with flight time $\sim$1100 days. Unlike the case in the ballistic transfer, in the low-thrust transfer case, the Earth-Earth flight time does not exactly equal to some integer multiple of Earth's orbital period and the spacecraft does not encounter the Earth at the same position from launch. However the mechanism in the low-thrust case is similar to the chemical case [81], where the V_∞ at the second Earth encounter

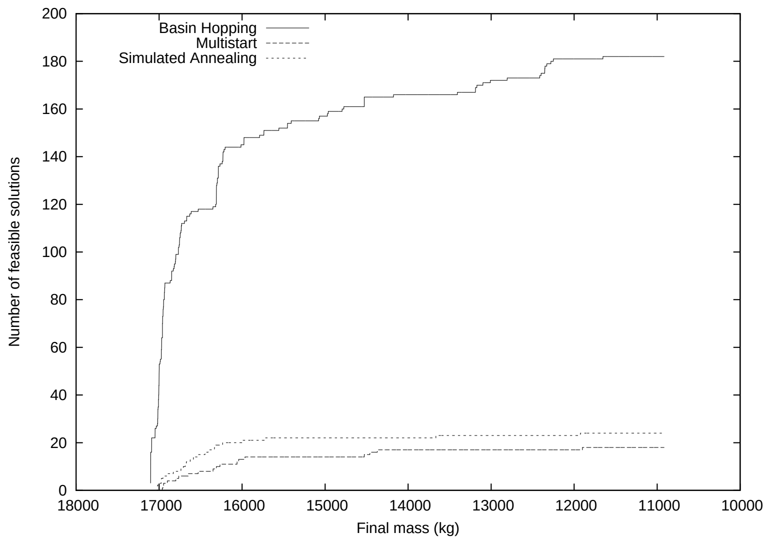

Figure 6.2: Cumulative number of solutions for the E-E-J mission

is increased through some small maneuvers.

6.5.2 Nuclear electric propulsion mission to Mercury

Our second test case is a mission to Mercury, inspired from the BepiColombo mission [97]. The planetary encounter sequence is Earth-Venus-Venus-Mercury-Mercury-Mercury (EVVMMM) and the mission parameters are given in Table 6.3. The mission parameters have been subject to many changes since the

Table 6.3: Parameters for a Mission to Mercury

Parameters	Values
Initial mass of the spacecraft	$1,300$ kg
Maximum thrust	0.34 N
Specific impulse	$3,200$ s
Launch date	Aug. 1, 2009 – Apr. 27, 2012
Launch V_∞	≤ 1.925 km/s
Arrival V_∞	≤ 0.5017 km/s
Arrival date	No later than Nov. 26, 2021
Minimum flyby radius	1.1 R_{planet}

original concept. Here we consider the mission as was described in a 2002 re-

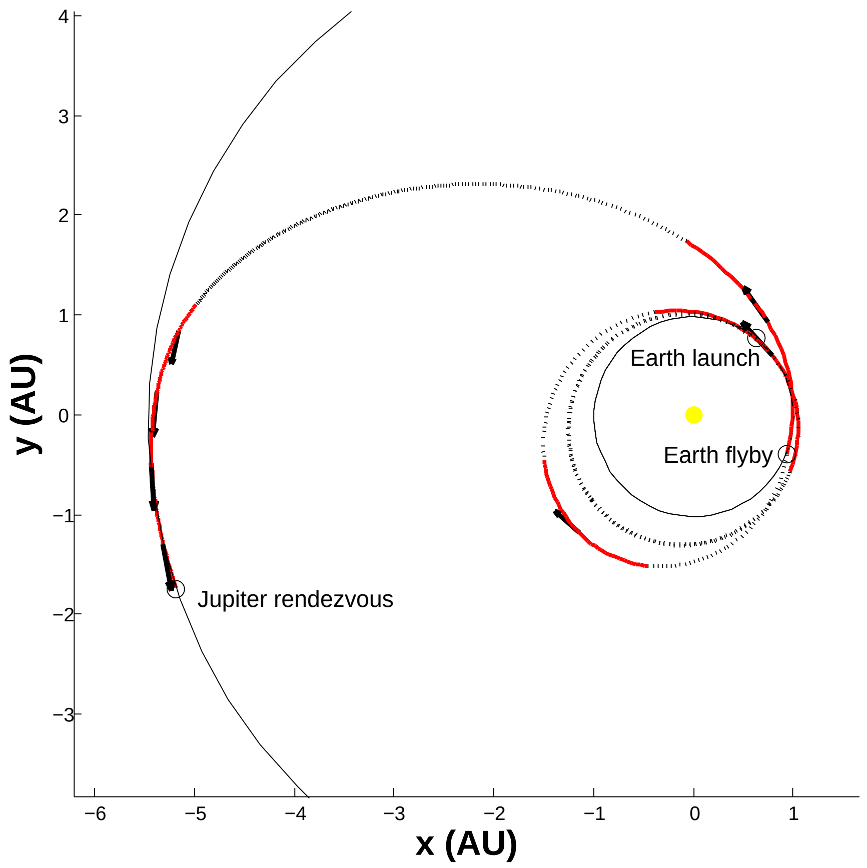

Figure 6.3: Trajectory plot of an Earth-Earth-Jupiter rendezvous mission

port from the European Space Operations Centre [53]. Unlike the solar electric propulsion (SEP) used in the BepiColombo mission, we do not model the SEP engine in our test but rather assume the thrust and specific impulse to be constant. In comparison with the first case, the number of flybys increases from 1 to 4 and the dimension of the problem increases to 222 with 99 nonlinear constraints. Also, we do not strictly require a rendezvous with Mercury with a V_∞ of zero, but the spacecraft is allowed to have a 0.5 km/s speed difference with respect to the planet.

Because of the added complexity, we allowed the global solvers to run for a longer time (∼48 hours), and we used a more performant machine (Intel Xeon@2.66 GHz with 8 GB of RAM). The test results are summarized in Table 6.4. Basin Hopping found more than three hundred locally optimal solutions, of which more than 80 are above 90% of the best solution; while Multistart found double the number of solutions than Basin Hopping, but with fewer high final

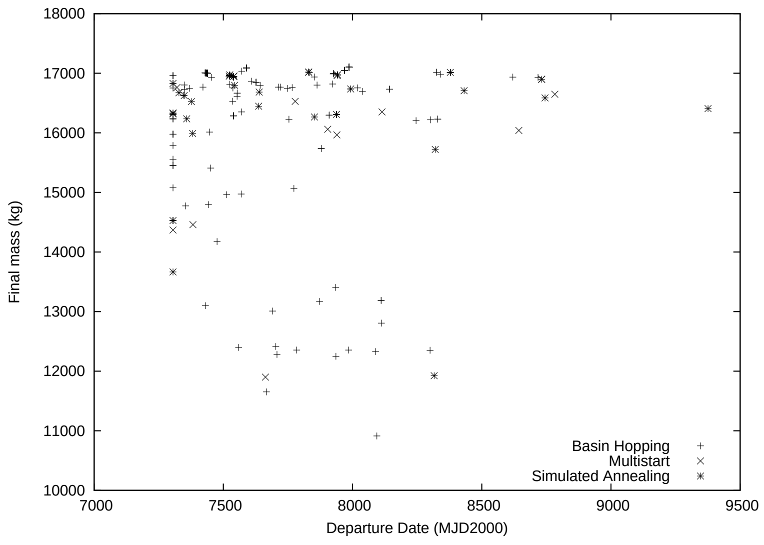

Figure 6.4: E-E-J solutions with various launch dates

mass trajectories (e.g. only 30 solutions are above the best 90%). Simulated Annealing only found about one hundred locally optimal solutions and with less good quality solutions than the other two algorithms.

From Figs. 6.6 and 6.7, we note that the first one third of the solutions have a high final mass (over 900 kg) and it covers many launch opportunities within the search space. If we do not restrict the analysis to the solutions that are also locally optimal, then the range of launch opportunities increases even more. The trajectory of the 'best' solution (with the highest final mass) is plotted in Fig. 6.8. Here the spacecraft leaves Earth on April 13, 2010, performs two flybys at Venus to lower its orbit, then encounters Mercury twice to lower its V_∞ to 0.5 km/s before it arrives on Sept 5, 2015. Comparing such trajectory with the one described in the BepiColombo early mission planning stage, we note that our results are comparable with those described in [53] (as cited in [37]) with regard to the overall shape of the optimal trajectory, which includes the planetary resonances, the times of flight, and the engine thrust periods, but have been obtained using a completely automated process.

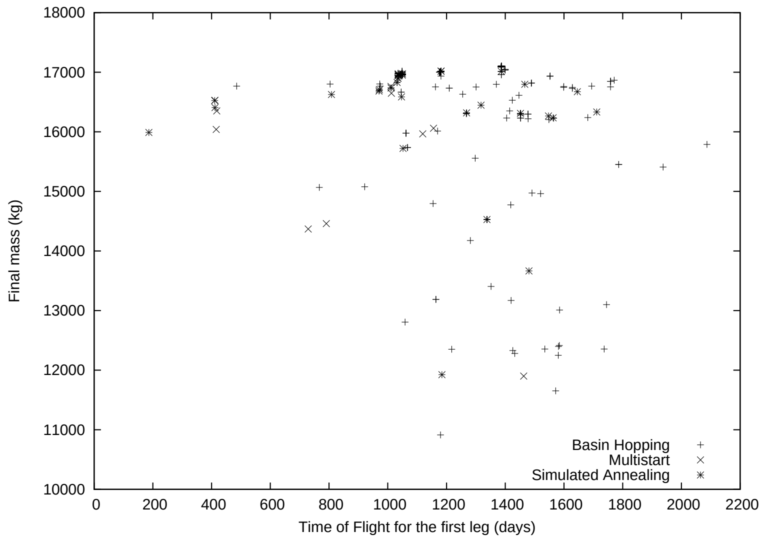

Figure 6.5: E-E-J solutions with various Earth-Earth transfer times

Table 6.4: Algorithm Statistics for a Mission to Mercury

Algorithm	Basin Hopping	Simulated Annealing	Multistart
Best (kg)	1,064	988	1,052
Worst (kg)	101	111	100
Mean (kg)	724	522	530
Mean best 10 (kg)	1,051	917	1,032
Std (kg)	277	253	261
No. of sol. above 95% best (1,011 kg)	35	0	1
No. of sol. above 90% best (958 kg)	88	14	31
Total no. of solutions	345	94	756

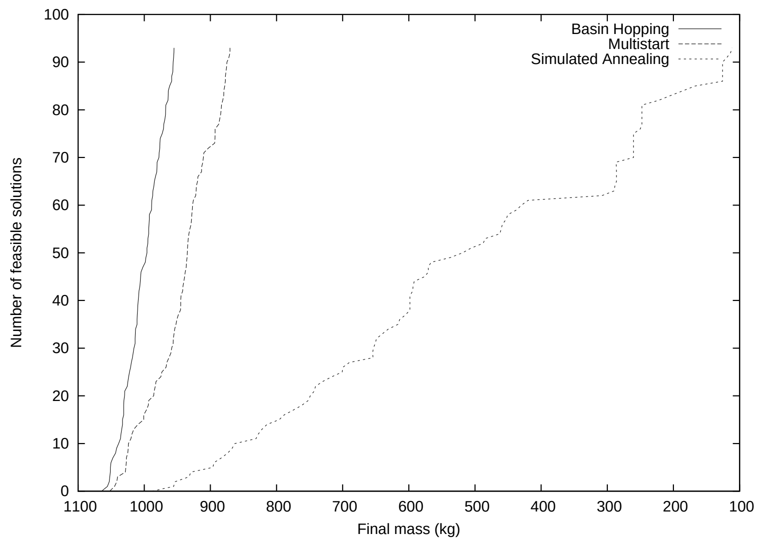

Figure 6.6: Cumulative number of solutions for the EVVMMM mission

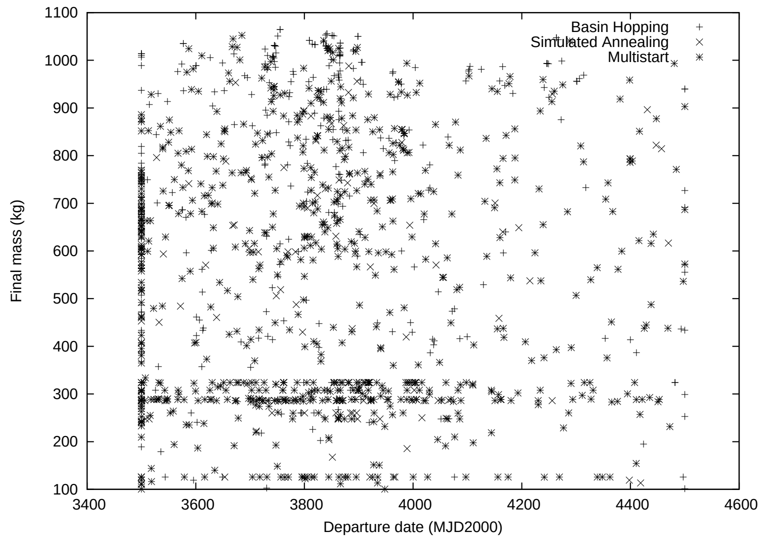

Figure 6.7: EVVMMM solutions with various launch dates

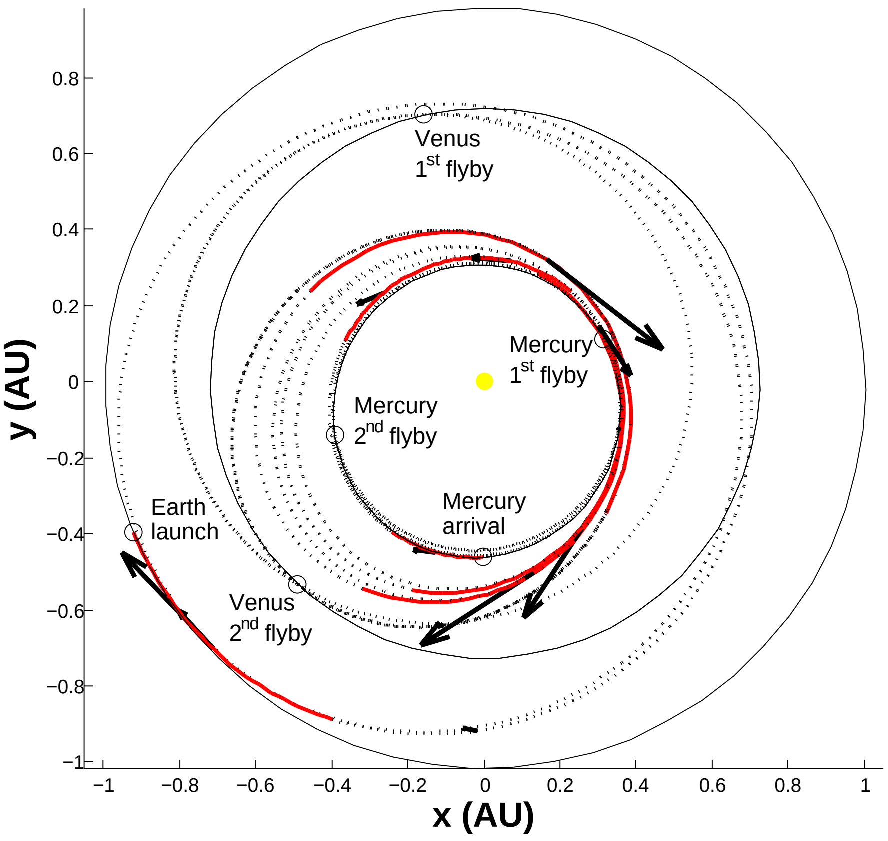

Figure 6.8: Trajectory plot of a mission to Mercury

Chapter 7

Conclusions

This thesis presented a set of techniques, mainly focusing on decomposition methods, to handle large scale problems. We focused our attention both on the theoretical aspects of the algorithms, and on the application to real world problems.

Decomposition methods

Regarding decomposition methods, we described and presented convergence proofs to a wide range of possible decomposition strategies. Such possibilities are actually exploited in many algorithms but, in most cases, no rigorous convergence analysis was performed. In chapter 3 we contributed in filling such gap by presenting a set of theorems which prove convergence for those algorithms. Furthermore, in section 3.2 we partially settled an open problem about the convergence of some methods that can be used for SVM training. The results here presented have also been successfully used in the later chapters to solve problems regarding network equilibrium on graphs.

Network equilibrium problems

In chapter 4 of this thesis we have considered the class of path-based optimization problems, motivated by the fact that this formulation allows us to deal with both additive and nonadditive path costs. We have proposed convergent inexact decomposition algorithms for a more general class of convex problems defined on the Cartesian product of convex sets. On the basis of the general framework studied, specific algorithms for network equilibrium problems have been presented. As well as the widely used Frank-Wolfe method, the proposed algorithms have global convergence properties, are simple to implement, represent a viable strategy in terms of memory consumption (at least for medium/large networks), and show a good efficiency in comparison with the Frank-Wolfe method. This latter fact is not surprising, however, due to their comparable simplicity, the presented algorithms can be easily implemented, and this represents a valuable characteristic for practitioners interested to manage their own code.

Concerning mathematical programming issues, we remark that most of the existing algorithms are tailored on the specific application of network equilibrium problems, and possible extensions to deal with similar classes of problems are not immediate. Thanks to the general basic framework developed in section 4.1, the approach proposed in this work can be extended, with a reasonable effort, to design optimization algorithms for a general class of large scale problems of the form

$$\begin{aligned} \min_x \quad & f(x) \\ & a'x = b \\ & l \le x \le u \end{aligned}$$

This class of problems includes important applications, like Support Vector Machine training problems, portfolio selection problems, a continuous formulation of the maximum clique problem, and is a current important topic in the field of mathematical programming as pointed out in a very recent paper [40].

Portfolio selection

In chapter 5 we have shown how to design a novel algorithm to solve, in a quite efficient way, the problem of choosing the sparsest portfolio which guarantees prefixed expected return and risk. From the point of view of optimization algorithms, the proposed approach can be seen as a general method to deal with sparse optimization, not necessarily related to portfolio optimization. Starting from a continuous equivalent formulation of the original discrete problem, the tools employed are Frank-Wolfe method as a local optimizer and iterated local search as a global method. We have seen that an efficient algorithm can be defined by exploiting a concave formulation. Applying this computational scheme to the sparsest portfolio problem, it turns out that the achieved computational efficiency is orders of magnitude higher than the one obtained by using a state of the art solver for an equivalent mixed integer formulation. So the first result regarding sparse portfolio optimization is having introduced a new optimization technique whose efficiency is significant for the class of sparse optimization problems over the intersection of a polytope with an ellipsoid. Moreover, we have shown that, when applied to optimal portfolio problems, this approach is capable of producing high quality portfolios, achieving a very good Sharpe Ratio, in particular when applied to datasets consisting of a large number of assets.

Space trajectory optimization

In chapter 6 we successfully applied global optimization techniques to achieve the automated design of two instances of multiple gravity assist low-thrust interplanetary trajectories within a ten year wide launch window. The first mission is the Jupiter Icy Moons Orbiter, later canceled by NASA, planned for the insertion of a spacecraft into an orbit around Jupiter, with the goal of collecting data about some about the planetary moons. The second one is BepiColombo mission to Mercury, scheduled to launch in 2014 which, in addition to performing measurements on the planet itself, will also be used to test some aspects of general relativity. The use of our technique is not limited to the two particular problems here studied as it rests upon a general interface between the Sims-Flanagan trajectory model and a global optimization layer hybridized with a local search as to deal efficiently with the nonlinear constraints. The resulting method makes no use of expert knowledge and starts from randomly generated trajectories, thus achieving a completely automated design process.

Appendix A

Linesearches

A.1 Armijo linesearch

In this section, for sake of completeness and to ease the reader, we recall some known results about Armijo-type line search algorithm.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function, and let $x, d \in \mathbb{R}^n$. An Armijo-type line search algorithm is described below. We report, without proof, the

Algorithm 18: Armijo linesearch

```
   Data: $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, $d \in \mathbb{R}^n$, $\Delta > 0$, $\delta \in (0,1)$, $\gamma \in (0,1)$
1  $j \leftarrow 1$;
2  $\alpha^j \leftarrow \Delta$;
3  while $f(x + \alpha^j d) > f(x) + \gamma \alpha^j \nabla f(x)' d$ do
4      $\alpha^{j+1} \leftarrow \delta \alpha^j$;
5      $j \leftarrow j + 1$;
6  end
7  return $\alpha^j$;
```

following theorem

Theorem A.1. *If d is a descent direction at x, then the Armijo linesearch algorithm terminates in a finite number of iterations, returning a value α such that*

$$f(x + \alpha d) \le f(x) + \gamma \alpha \nabla f(x)' d \tag{A.1}$$

and such that either $\alpha = \Delta$ or

$$f(x + \frac{\alpha}{\delta} d) > f(x) + \gamma \frac{\alpha}{\delta} \nabla f(x)' d$$

Finally, we report in theorem A.2 the widely known convergence result about the repeated use of the Armijo linesearch

Theorem A.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function. Let $\{x^k\} \subset \mathbb{R}^n$, $\{d^k\} \subset \mathbb{R}^n$, $\{\Delta^k\} \subset \mathbb{R}$ and $\{\alpha^k\} \subset \mathbb{R}$ be sequences such that*

(a) $f(x^{k+1}) \leq f(x^k + \alpha^k d^k)$

(b) $\nabla f(x^k)' d^k < 0$

(c) *There exists a forcing function σ such that*

$$\Delta^k \geq \frac{1}{\|d^k\|} \sigma \left(\frac{|\nabla f(x^k)' d^k|}{\|d^k\|} \right)$$

(d) *α^k is computed by the Armijo linesearch algorithm on f, x^k, d^k, Δ^k and some fixed δ and γ.*

Suppose also that the level set $\mathcal{L}(x^1)$ is a compact. Then it holds

(i) $f(x^{k+1}) < f(x^k)$

(ii) $\lim_{k\to\infty} \frac{\nabla f(x^k)' d^k}{\|d^k\|} = 0$

Proof. Point *(i)* immediately follows from equation (A.1), and from hypothesis *(a)*, *(b)* and *(d)*. Let us prove thesis *(ii)*. From equation (A.1) it holds

$$f(x^k) - f(x^{k+1}) \geq \gamma \alpha^k \|d^k\| \frac{|\nabla f(x^k)' d^k|}{\|d^k\|}$$

Since f is continuous and $\mathcal{L}(x^1)$ is compact, f must be bounded below. Then from point *(i)* it must hold

$$\lim_{k\to\infty} f(x^k) - f(x^{k+1}) = 0$$

which implies

$$\lim_{k\to\infty} \gamma \alpha^k \|d^k\| \frac{|\nabla f(x^k)' d^k|}{\|d^k\|} = 0 \tag{A.2}$$

Suppose by contradiction that *(ii)* is false. Then it must exist $\eta > 0$ such that

$$\lim_{k\to\infty} \frac{\nabla f(x^k)' d^k}{\|d^k\|} = -\eta \tag{A.3}$$

Then, by equation (A.2) we can write

$$\lim_{k\to\infty} \alpha^k \|d^k\| = 0 \tag{A.4}$$

Suppose that, for k big enough, it always holds $\alpha^k = \Delta^k$. Then

$$\Delta^k \|d^k\| \geq \sigma \left(\frac{|\nabla f(x^k)' d^k|}{\|d^k\|} \right) \tag{A.5}$$

which contradicts either equation (A.3) or equation (A.4). Then by theorem A.1, there must exist an infinite number of iterations where

$$f\left(x + \frac{\alpha}{\delta} d\right) > f(x) + \gamma \frac{\alpha}{\delta} \nabla f(x)' d \tag{A.6}$$

Since also for every k the value

$$\frac{\nabla f(x^k)' d^k}{\|d^k\|}$$

is bounded, there must exist an infinite subsequence $K \subseteq \mathbb{N}$, $\hat{x} \in \mathbb{R}^n$, $\hat{d} \in \mathbb{R}^n$ such that

$$\lim_{k \to \infty, k \in K} x^k = \hat{x} \qquad \lim_{k \to \infty, k \in K} \frac{\nabla f(x^k)' d^k}{\|d^k\|} = \hat{d}$$

and such that, for every $k \in K$, $\alpha^k < \Delta^k = 1$. By the mean value theorem we can write

$$f\left(x^k + \frac{\alpha^k}{\delta} d^k\right) = f(x^k) + \frac{\alpha^k}{\delta} \nabla f(z^k)' d^k \tag{A.7}$$

with

$$z^k = x^k + \theta^k \frac{\alpha^k}{\delta} d^k \quad \text{for some } \theta^k \in (0, 1)$$

Merging equations (A.6) and (A.7) we obtain

$$\nabla f(z^k)' d^k > \gamma \nabla f(x^k)' d^k$$

or

$$\frac{\nabla f(z^k)' d^k}{\|d^k\|} > \gamma \frac{\nabla f(x^k)' d^k}{\|d^k\|} \tag{A.8}$$

But from equation (A.4) it follows

$$\lim_{k \to \infty, k \in K} z^k = \lim_{k \to \infty, k \in K} \left(x^k + \theta^k \frac{\alpha^k}{\delta} d^k \right) = \hat{x}$$

Taking k to infinity in equation (A.8) we conclude that

$$\nabla f(\hat{x})' \hat{d} \geq \gamma \nabla f(\hat{x})' \hat{d}$$

which contradicts the fact that $\gamma < 1$ and proves the theorem. $\square$

A.2 Quadratic linesearch

The quadratic linesearch can be used instead of the Armijo linesearch along a descent direction. Although the algorithms are very similar, the convergence properties are different.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function, and let $x, d \in \mathbb{R}^n$. The quadratic

Algorithm 19: Quadratic linesearch

```
  Data: $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, $d \in \mathbb{R}^n$, $\Delta > 0$, $\delta \in (0,1)$, $\gamma > 0$
1 $j \leftarrow 1$;
2 $\alpha^j \leftarrow \Delta$;
3 while $f(x + \alpha^j d) > f(x) - \gamma \left(\alpha^j\right)^2 \|d\|^2$ do
4     $\alpha^{j+1} \leftarrow \delta\alpha^j$;
5     $j \leftarrow j + 1$;
6 end
7 return $\alpha^j$;
```

linesearch algorithm differs from algorithm 18 only for the condition in line A.2. It is trivial to prove that the stepsize will go to zero.

Theorem A.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth bounded function, and let $\{d^k\} \subset \mathbb{R}^n$, $x^0 \in \mathbb{R}^n$ and $x^{k+1} = x^k + \alpha^k d^k$, where α^k is computed through algorithm A.2. Then*

$$\lim_{k\to\infty} \alpha^k \left\|d^k\right\| = 0$$

Proof. The sequence $\{f(x^k)\}$ is decreasing and bounded, so it must hold

$$\lim_{k\to\infty} f(x + \alpha^j d) - f(x) = 0$$

The algorithm ensures that, for every k

$$f(x + \alpha^j d) - f(x) \le -\gamma \left(\alpha^j\right)^2 \|d\|^2$$

which implies the thesis. □

Theorem A.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth bounded function, and let $\{d^k\} \subset \mathbb{R}^n$, $x^0 \in \mathbb{R}^n$ and $x^{k+1} = x^k + \alpha^k d^k$, where α^k is computed through algorithm A.2. Then*

$$\lim_{k\to\infty} \nabla f(x^k)' d^k = 0 \tag{A.9}$$

Proof. Let us partition $\mathbb{N}$ into a possibly infinite number of infinite subsets such that, for every subset K, either

(i) for every $k \in K$, $\alpha^k = \Delta$

(ii) for every $k \in K$, $\alpha^k < \Delta$ and there exist $\bar{x}$ and $\bar{d}$ such that

$$\lim_{k\to\infty, k\in K} x^k = \bar{x} \qquad \lim_{k\to\infty, k\in K} d^k = \bar{d}$$

We will prove the theorem by showing that equation (A.9) holds for every such K.

Suppose case *(i)* holds. Then from theorem we can see that $\|d^k\|$ goes to zero, and this proves the thesis.

Suppose case holds. Then from line of algorithm we can deduce that

$$f\left(x^k + \frac{\alpha^k}{\delta} d^k\right) > f(x^k) - \gamma \left(\frac{\alpha^k}{\delta}\right)^2 \|d^k\|^2$$

or

$$\frac{f\left(x^k + \frac{\alpha^k}{\delta} d^k\right) - f(x^k)}{\frac{\alpha^k}{\delta}\|d^k\|} > -\gamma \frac{\alpha^k}{\delta}\|d^k\|$$

By the mean value theorem

$$\frac{f\left(x^k + \frac{\alpha^k}{\delta} d^k\right) - f(x^k)}{\frac{\alpha^k}{\delta}\|d^k\|} = \nabla f(z^k)' d^k \tag{A.10}$$

with

$$z^k = x^k + \theta^k \frac{\alpha^k}{\delta} d^k \quad \text{for some } \theta^k \in (0, 1)$$

Then

$$\nabla f(z^k)' d^k > -\gamma \frac{\alpha^k}{\delta}\|d^k\| \tag{A.11}$$

By the definition of z^k and theorem A.2 we can write

$$\lim_{k\to\infty, k\in K} z^k = \lim_{k\to\infty, k\in K} x^k = \bar{x}$$

so, taking the limit of equation (A.11) we obtain

$$\nabla f(\bar{x})' \bar{d} = 0 \tag{A.12}$$

Then, from theorem 2.3 we can infer that the function in equation (A.12) is continuous, and this implies the thesis. □

Bibliography

[1] Addis, B., Cassioli, A., Locatelli, M., and Schoen, F. A global optimization method for the design of space trajectories. *Computational Optimization and Applications 3* (Apr. 2011), 635–652.

[2] Alemany, K., and Braun, R. Survey of global optimization methods for low-thrust, multiple asteroid tour missions. In *AAS/AIAA Space Flight Mechanics Meeting* (Jan. 2007).

[3] Armellin, R., Di Lizia, P., Topputo, F., Lavagna, M., Bernelli-Zazzera, F., and Berz, M. Gravity assist space pruning based on differential algebra. *Celestial Mechanics and Dynamical Astronomy 106*, 1 (2010), 1–24.

[4] Bar-Gera, H. Origin-based algorithm for the traffic assignment problem. *Transportation Science 36* (2002), 398–417.

[5] Bar-Gera, H. Traffic assignment by paired alternative segments. *Transportation Science Part B 44* (2010), 1022–1046.

[6] Bertsekas, D., and Tsitsiklis, J. *Parallel and distributed computation: numerical methods.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[7] Bertsekas, D. P. *Nonlinear Programming*, second ed. Athena Scientific, Belmont, Massachussets, 1999.

[8] Bertsimas, D., and Shioda, R. Algorithm for cardinality-constrained quadratic optimization. *Computational Optimization and Applications 43*, 1 (May 2009), 1–22.

[9] Bienstock, D. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming 74* (1996), 121–140.

[10] Black, F., and Litterman, R. Asset allocation: combining investor views with market equilibrium. *The Journal of Fixed Income 1* (1991), 7–18.

[11] Bonettini, S. Inexact block coordinate descent methods with application to non-negative matrix factorization. *IMA Journal of Numerical Analysis 31*, 4 (2011), 1431–1452.

[12] Bonettini, S., Zanella, R., and Zanni, L. A scaled gradient projection method for constrained image deblurring. *Inverse Problems 25*, 1 (2009).

[13] Bradley, P. S., and Mangasarian, O. L. Feature selection via concave minimization and support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998* (1998), J. W. Shavlik, Ed., Morgan Kaufmann, pp. 82–90.

[14] Broadie, M. Computing efficient frontiers using estimated parameters. *Ann. Oper. Res. 45* (December 1993), 21–58.

[15] Brodie, J., Daubechies, I., Mol, C. D., Giannone, D., and Loris, I. Sparse and stable Markowitz portfolios. *European Central Bank Working Paper Series 36* (2008).

[16] Buzzi, C., Grippo, L., and Sciandrone, M. Convergent decomposition techniques for training RBF neural networks. *Neural Computation 13*, 8 (2001), 1891–1920.

[17] Cassioli, A., Di Lorenzo, D., Locatelli, M., Schoen, F., and Sciandrone, M. Machine learning for global optimization. *Computational Optimization and Applications 51* (2012), 279–303. 10.1007/s10589-010-9330-x.

[18] Cesarone, F., Scozzari, A., and Tardella, F. Efficient algorithms for mean-variance portfolio optimization with hard real-world constraints. *Giornale dell'Istituto Italiano degli Attuari, to appear*.

[19] Chen, G., and Teboulle, M. A proximal-based decomposition method for convex minimization problems. *Mathematical Programming 64* (1994), 81–101. 10.1007/BF01582566.

[20] Chopra, V. Improving optimization. *Journal of Investing 2* (1993), 51–59.

[21] Chopra, V., Hensel, C., and Turner, A. Massaging mean-variance inputs: return from alternative investment strategies in the 1980s. *Management Science 39* (1993), 845–855.

[22] Chopra, V., and Ziemba, W. The effect of errors in means, variances and covariances on optimal portfolio choice. *Journal of Portfolio Management 19* (1993), 6–11.

[23] Corana, A., Marchesi, M., Martini, C., and Ridella, S. Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm. *ACM Transactions on Mathematical Software (TOMS) 13*, 3 (1987), 280.

[24] Danby, J. *Fundamentals of Celestial Mechanics.* Willman-Bell, 1988.

[25] De Luca, A., and Di Pillo, G. Exact augmented lagrangian approach to multilevel optimization of large-scale systems. *International Journal of Systems Science 18*, 1 (1987), 157–176.

[26] DeMiguel, V., Garlappi, L., Nogales, F. J., and Uppal, R. A generalized approach to portfolio optimization: improving performance by constraining portfolio norms. *Management Science 55* (2009), 798–812.

[27] DeMiguel, V., and Nogales, F. J. Portfolio selection with robust estimation. *Operations Research 57*, 3 (2009), 560–577.

[28] Di Lizia, P., and Radice, G. Advanced global optimisation tools for mission analysis and design. Tech. Rep. 03-4101b, European Space Agency, the Advanced Concepts Team, 2004.

[29] Di Lorenzo, D., Liuzzi, G., Rinaldi, F., Schoen, F., and Sciandrone, M. A concave optimization-based approach for sparse portfolio selection. *Optimization Methods and Software* (2011).

[30] Dial, R. B. A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. *Transportation Research Part B: Methodological 40*, 10 (2006), 917 – 936.

[31] Du, D.-Z., and Zhang, X.-S. Global convergence of Rosen's gradient projection algorithm. *Mathematical Programming 44* (1989), 357–366.

[32] El Ghaoui, L., Oks, M., and Oustry, F. Worst-case value-at-risk and robust portfolio optimization: a conic programming approach. *Oper. Res. 51*, 4 (2003), 543–556.

[33] Florian, M., Constantin, I., and Florian, D. A new look at projected gradient method for equilibrium assignment. *Journal of the Transportation Research Record Board 2090*, 0361-1981 (2009), 10–16.

[34] Florian, M., and Hearn, D. W. *Hanbooks in OR & MS*, vol. 8. Elsevier Science B.V., Amsterdam, 1995, ch. Network Equilibrium Models, pp. 485–550.

[35] Frank, M., and Wolfe, P. An algorithm for quadratic programming. *Naval Research Logistics Quarterly 3*, 1-2 (1956), 95–110.

[36] Gabriel, S., and Bernstein, D. The traffic equilibrium problem with non-additive path costs. *Transportation Science 31* (1997), 337–348.

[37] Gil-Fernández, J., Graziano, M., Gomez-Tierno, M., and Milic, E. Autonomous low-thrust guidance: application to SMART-1 and Bepi-Colombo. *Annals of the New York Academy of Sciences 1017*, Astrodynamics, Space Missions, and Chaos (2004), 307–327.

[38] Gill, P. E., Murray, W., and Saunders, M. A. *User's Guide for SNOPT version 7, Software for Large-Scale Nonlinear Programming.* Stanford Business Software Inc., Feb 2006.

[39] Gill, P. E.and Murray, W., and Saunders, M. A. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization 12*, 4 (2002), 979–1006.

[40] Gonzalez-Limai, M. D., Hager, W. W., and Zhang, H. An affine-scaling interior point method for continuous knapsack constraints with applications to support vector machines. *Siam Journal on Optimization 21*, 1 (2011), 361–390.

[41] Greeley, R., and Johnson, T., E. Report of the NASA science definition team for the Jupiter icy moons orbiter. Tech. rep., NASA Science Definition Team, Feb. 2004.

[42] Grippo, L., and Sciandrone, M. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization Methods and Software 10*, 4 (1999), 587–637.

[43] Grippo, L., and Sciandrone, M. On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Operations Research Letters 26*, 3 (2000), 127 – 136.

[44] Grosso, A., Jamali, A., Locatelli, M., and Schoen, F. Solving the problem of packing equal and unequal circles in a circular container. *Journal of Global Optimization* (July 2009).

[45] Grosso, A., Locatelli, M., and Schoen, F. A population-based approach for hard global optimization problems based on dissimilarity measures. *Mathematical Programming 110* (2007), 373–404. 10.1007/s10107-006-0006-3.

[46] Horowitz, A. J., and Administration, United States. Federal Highway. *Delay-volume relations for travel forecasting: based on the 1985 Highway Capacity Manual.* US Dept. of Transportation, Federal Highway Administration, 1991.

[47] Izzo, D. Advances in global optimization for space trajectory design. In *25th International Symposium on Space Technology and Science* (2006), Japan Society for Aeronautical and Space Sciences and ISTS.

[48] Izzo, D. Global trajectory optimization competition portal. http://www.esa.int/gsp/ACT/mad/op/GTOC/index.htm, June 2009.

[49] Izzo, D., Becerra, V., Myatt, D., Nasuto, S., and Bishop, J. Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. *Journal of Global Optimization 38*, 2 (2007), 283–296.

[50] Izzo, D., Vinko, T., and Zapatero, M. Global trajectory optimization competition database. http://www.esa.int/gsp/ACT/inf/op/globopt.htm, June 2009.

[51] Jagannathan, R., and Ma, T. Risk reduction in large portfolios: why imposing the wrong constraints helps. *Journal of Finance 58* (2003), 1651–1684.

[52] Jayakrishnan, R., Tsai, W. K., Prashker, J. N., and Rajadhyaksha, S. A faster path-based algorithm for traffic assignment. *Transportation Research Record*, 1443 (1994), 75–75.

[53] Katzkowski, M., Corral, C., Jehn, R., Pellon, J.-L., Landgraf, M., Khan, M., Yanez, A., and Biesbroek, R. BepiColombo Mercury cornerstone mission analysis: input to definition study. Tech. rep., ESA European Space Operations Centre, Apr. 2002.

[54] Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[55] Larson, T., Lindberg, P.-O., Lundgren, J., and Patriksson, M. *Transportation Planning–State of the Art.* The Netherlands: Kluwer Academic Publishers, 2002, ch. On traffic quilibrium models with a nonlinear time/money relation, pp. 19–31.

[56] Leary, R. H. Global optimization on funneling landscapes. *J. of Global Optimization 18* (December 2000), 367–383.

[57] Lin, C.-J. On the convergence of the decomposition method for support vector machines. *Neural Networks, IEEE Transactions on 12*, 6 (2001), 1288–1298.

[58] Lourenço, H. R., Martin, O. C., and Stülze, T. Iterated local search. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003, pp. 321–353.

[59] Luenberger, D. Investment science, 1998.

[60] Luo, Z. Q., and Tseng, P. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications 72* (1992), 7–35. 10.1007/BF00939948.

[61] Mangasarian, O. Machine learning via polyhedral concave minimization. *Applied Mathematics and Parallel Computing – Festschrift for Klaus Ritter* (1996), 175–188.

[62] Markowitz, H. Portfolio selection. *The Journal of Finance 7*, 1 (1952), 77–91.

[63] Merton, R. C. On estimating the expected return on the market: an exploratory investigation. *Journal of Financial Economics 8*, 4 (December 1980), 323–361.

[64] Michaud, R. O. *Efficient Asset Management: a practical guide to stock portfolio management and asset allocation.* Oxford University Press, 2001.

[65] Michelot, C. A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. *Journal of Optimization Theory and Applications 50* (1986), 195–200. 10.1007/BF00938486.

[66] Myatt, D., Becerra, V., Nasuto, S., and Bishop, J. Advanced global optimisation tools for mission analysis and design. Tech. Rep. 03-4101a, European Space Agency, the Advanced Concepts Team, 2004.

[67] Nie, Y. A class of bush-based algorithms for the traffic assignment problem. *Transportation Research Part B 44* (2010), 73–89.

[68] Overgaard, K. R. Urban transportation planning: traffic estimation. *Traffic Quarterly 21*, 2 (1967).

[69] Panicucci, B., Pappalardo, M., and Passacantando, M. A path-based double projection method for solving the asymmetric traffic network equilibrium problem. *Optimization Letters 1*, 2 (2007), 171–185.

[70] Parcher, D. W., and Sims, J. A. Gravity-assist trajectories to Jupiter using nuclear electric propulsion. In *AAS/AIAA Astrodynamics Specialist Conference* (Aug. 2005).

[71] Patriksson, M. Decomposition methods for differentiable optimization problems over Cartesian product sets. *Computational Optimization and Applications 9* (1998), 5–42. 10.1023/A:1018358602892.

[72] Patriksson, M. Algorithms for computing traffic equilibria. *Networks and Spatial Economics 4* (2004), 23–34.

[73] Raghavachari, M. On connections between zero-one integer programming and concave programming under linear constraints. *Oper. Res. 17* (1969), 680–684.

[74] Rinaldi, F. Dipartimento di informatica e sistemistica, sapienza università di roma. Tech. rep., 2009.

[75] Rinaldi, F. New results on the equivalence between zero-one programming and continuous concave programming. *Optimization Letters 3* (2009), 377–386.

[76] Rinaldi, F., Schoen, F., and Sciandrone, M. Concave programming for minimizing the zero-norm over polyhedral sets. *Computational Optimization and Applications 46* (2010), 467–486.

[77] Rosen, J. B. The gradient projection method for nonlinear programming. Part I. Linear constraints. *Journal of the Society for Industrial and Applied Mathematics 8*, 1 (1960), pp. 181–217.

[78] Schütze, O., Vasile, M., Junge, O., Dellnitz, M., and Izzo, D. Designing optimal low-thrust gravity-assist trajectories using space pruning and a multi-objective approach. *Engineering Optimization 41*, 2 (2009), 155–181.

[79] Sims, J. A., and Flanagan, S. N. Preliminary design of low-thrust interplanetary missions. In *AAS/AIAA Astrodynamics Specialist Conference* (Aug. 1999).

[80] Solodov, M. V. A class of decomposition methods for convex optimization and monotone variational inclusions via the hybrid inexact proximal point framework. *Optimization Methods and Software 19*, 5 (2004), 557–575.

[81] Strange, N. J., and Sims, J. A. Methods for the design of v-infinity leveraging maneuvers. In *AAS/AIAA Astrodynamics Specialist Conference* (July/Aug. 2001).

[82] Tseng, P. Decomposition algorithm for convex differentiable minimization. *Journal of Optimization Theory and Applications 70* (1991), 109–135. 10.1007/BF00940507.

[83] Tseng, P. Alternating projection-proximal methods for convex programming and variational inequalities. *SIAM J. on Optimization 7* (April 1997), 951–965.

[84] Tsiolkovsky, K. E. Exploration of the universe with reaction machines (in Russian). *The Science Review 5* (1903).

[85] Vasile, M., and De Pascale, P. Preliminary design of multiple gravity-assist trajectories. *Journal of Spacecraft and Rockets 43*, 4 (2006), 794–805.

[86] Vavrina, M. A., and Howell, K. C. Global low-thrust trajectory optimization through hybridization of a genetic algorithm and a direct method. In *AIAA/AAS Astrodynamics Specialist Conference* (Aug. 2008).

[87] Vinko, T., and Izzo, D. Global optimisation heuristics and test problems for preliminary spacecraft trajectory design. Tech. Rep. GOHTPPSTD, European Space Agency, the Advanced Concepts Team, 2008.

[88] Wales, D. J., and Doye, J. P. K. Global optimization by basin-hopping and the lowest energy structures of Lennard–Jones clusters containing up to 110 atoms. *Journal of Physical Chemistry A 101*, 28 (1997), 5111–5116.

[89] Wales, D. J., and Scheraga, H. A. Global optimization of clusters, crystals, and biomolecules. *Science 285* (1999), 1368–1372.

[90] Weston, J., Elisseeff, A., Schölkopf, B., and Tipping, M. Use of the zero norm with linear models and kernel methods. *Journal of Machine Learning Research 3* (Mar. 2003), 1439–1461.

[91] Whiffen, G. J. An investigation of a Jupiter Galilean moon orbiter trajectory. In *AAS/AIAA Astrodynamics Specialist Conference* (Aug. 2003).

[92] Yam, C. H., Biscani, F., and Izzo, D. Global optimization of low-thrust trajectories via impulsive delta-v transcription. In *27th International Symposium on Space Technology and Science* (July 2009).

[93] Yam, C. H., Di Lorenzo, D., and Izzo, D. Constrained global optimization of low-thrust interplanetary trajectories. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (July 2010), pp. 1 –7.

[94] Yam, C. H., Di Lorenzo, D., and Izzo, D. Low-thrust trajectory design as a constrained global optimization problem. *Journal of Aerospace Engineering 225*, 11 (2011), 1243–1251.

[95] Yam, C. H., McConaghy, T. T., Chen, K. J., and Longuski, J. M. Design of low-thrust gravity-assist trajectories to the outer planets. In *55th International Astronautical Congress* (Oct. 2004).

[96] Yam, C. H., McConaghy, T. T., Chen, K. J., and Longuski, J. M. Preliminary design of nuclear electric propulsion missions to the outer planets. In *AIAA/AAS Astrodynamics Specialist Conference* (Aug. 2004).

[97] Yarnoz, D. G., Jehn, R., and Croon, M. Interplanetary navigation along the low-thrust trajectory of BepiColombo. *Acta Astronautica 59*, 1-5 (2006), 284 – 293.