Chapter 7
Introduction to Optimisation

Annalisa Riccardi, Edmondo Minisci, Kerem Akartunali, Cristian Greco, Naomi Rutledge, Alexander Kershaw, and Aymen Hashim

Abstract This chapter gives a brief introduction to the formulation of optimisation problems and solving algorithms. After mentioning the different classes of problems, such as continuous/discrete, local/global and single-/multi-objective, and introducing some of the useful terminology, the chapter is split in two main parts: (1) formulations and algorithms for continuous problems, including optimal control, and (2) formulations and algorithms for integer and mixed-integer problems. Both sections first consider standard deterministic methods that have been derived starting by optimality criteria, then more recent heuristics derived by experience and sometimes inspired by nature. This gives the basis to better read and understand some of the following chapters on more advanced topics.

Keywords Optimisation · Optimal control · Network optimisation · Multi-objective · Continuous variables · Combinatorial variables

7.1 Introduction

Optimisation, derived from the Latin word optimus meaning 'the best', is the general name used to characterise the process of finding the best possible solution for a problem given a measure of 'goodness'. This is, for example, the problem of finding the shortest or fastest route between two points or the best investment in the stock market that minimises risk and maximises return.

People have been optimising since the beginning of the humankind era, but the roots for modern-day mathematical and engineering optimisation can be traced to the Second World War, where optimisation processes were formalised, implemented and applied to practical operational problems. The term operational research (OR)

A. Riccardi (✉) · E. Minisci · K. Akartunali · C. Greco · N. Rutledge · A. Kershaw · A. Hashim
University of Strathclyde, Glasgow, UK
e-mail: annalisa.riccardi@strath.ac.uk; edmondo.minisci@strath.ac.uk; kerem.akartunali@strath.ac.uk; c.greco@strath.ac.uk; naomi.rutledge.2013@uni.strath.ac.uk; alexander.kershaw.2013@uni.strath.ac.uk; aymen.hashim.2013@uni.strath.ac.uk

M. Vasile (ed.), *Optimization Under Uncertainty with Applications to Aerospace Engineering*, https://doi.org/10.1007/978-3-030-60166-9_7

originated from the activities performed by teams of multidisciplinary experts in the armed forces that were using advanced analytical methods to devise better decisions. Applications in the service industries did not begin until the mid-1960s, where the knowledge generated during the war was applied to logistic-related problems.

The term 'programming' is often used in relation to optimisation: mathematical programming, linear programming, non-linear programming, mixed-integer programming, etc. In principal, the original use of the word 'programming' has little to do with modern-day computer programming. Before the days of computing, a set of values which represented a solution to a problem was referred to as a programme. Nowadays, software is programmed to find a set of optimal values (or 'programme') for your problem. The intention of optimisation in modern-day programming is to maximise or minimise an objective function (performance measure indicator) with respect to a set of variables (optimisation variables) subject to one or more constraints. Modern mathematical optimisation can be used in a wide array of fields and disciplines, ranging from the design of aircrafts, the planning of routes and schedules, to the design of a control profile for an operating machine. In any optimisation problem, there are formulation and programming challenges that must be overcome to find an optimal solution. Some of them are discussed in the next section.

7.1.1 Solving an Optimisation Problem

There are three main challenges, or steps, to be addressed when facing a general optimisation problem: problem formulation, problem characteristics and algorithm selection.

- Problem formulation: the problem, originally described in general terms, needs to be translated into its mathematical formulation, including the identification of the set of optimisation variables and constant problem parameters, definition of objectives and constraints.
- Problem characteristics: the dimension of the design vector space (number of optimisation variables) and its nature (continuous or discrete), dimension of the objectives and constraints space (number of performance measures and constraints functions), their degree of non-linearity, their smoothness, their landscape as well as their computational cost.
- Algorithm selection: from the pool of available algorithms the most suitable algorithm needs to be selected to solve the formulated problem.

Without loss of generalisation we can restrict ourselves to discuss only the case of minimisation: find $\mathbf{x}^* \in \Omega \subseteq \mathbb{R}^{n_x}$

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \Omega} f(\mathbf{x}) \quad \text{subject to } c(\mathbf{x}) \leq 0,$$

where $f : \Omega \rightarrow \mathbb{R}^{n_{\text{obj}}}$ is the objective function and $c : \Omega \rightarrow \mathbb{R}^m$ the constraints function.

The set of points satisfying the constraints is called the feasible region

$$D = \{\mathbf{x} \in \Omega \mid c(\mathbf{x}) \leq 0\}.$$

The problem can be rewritten as

$$\min_{\mathbf{x} \in D} f(\mathbf{x}).$$

Depending on the nature of the objective function and constraints (linear or non-linear, single or multi-objective) of the search space (continuous or discrete), the optimisation problems can be divided in different classes

- **Continuous** or **discrete**: the optimisation variables belong to a feasible set that is a subset of the real space

  $$\mathbf{x} \in D \subseteq \mathbb{R}^n.$$

  In some problems the variable $\mathbf{x}$ represents integer values. Such problems are defined as **integer programming problems**, and the variables are in a feasible set such that

  $$\mathbf{x} \in D \subseteq \mathbb{Z}^n.$$

  A subset of integer programming problems is the **binary programming problems** where

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

  If some of the variables in the problem are not restricted to be integer variables, the problem is called **mixed-integer programming problem**

  $$\mathbf{x} = (\mathbf{x}_r, \mathbf{x}_d) \in D \subseteq \mathbb{R}^{n_r} \times \mathbb{Z}^{n_d}, \quad \text{with } n_r + n_d = n.$$

- **Constrained** or **unconstrained**: if there are no constraints on the design variables ($m = 0$), the problem is unconstrained. For constrained optimisation, instead $m > 0$. Unconstrained problems arise also as reformulations of constrained optimisation problems, in which the constraints are added to the objective function as penalisation terms.
- **Linear** or **non-linear**: if the objective function and all the constraints are linear functions of $\mathbf{x}$, the problem is called **linear programming** problem. Otherwise if some of the constraints or the objectives are non-linear functions, the problem is a **non-linear programming** problem.

- **Global** or **local**: many algorithms for non-linear optimisation problems find only a **local solution**, i.e. a point at which the objective function is smaller than all the other feasible points in a neighbourhood. They do not always find the **global solution**, which is the point that has the lowest function value among all the points of the feasible region. Only for linear programming problems and convex programming problems, the local solution is also the global one. An objective function that presents a large number of local optima is called *multimodal* function.
- **Single**- or **multi**- or **many**-objective: if the objective function is a scalar function, that is

$$n_{\text{obj}} = 1$$

the problem is said to be **single-objective**. In many engineering applications, one is seeking a trade-off between different objectives; f is in this case a vectorial function with

$$n_{\text{obj}} > 1$$

and the problem is called a **multi-objective** optimisation problem ($1 < n_{\text{obj}} \leq 3$) or **many-objective** optimisation problem ($n_{\text{obj}} > 3$). Multi-objective optimisation problems can be transformed into single-objective problems, for example, by means of aggregating functions, condensing all objectives in a single-cost function with the use of weights coefficients, or by using alternatives such as the $\epsilon-$constrained, and the goal-attainment methods. More details are given in Sect. 7.2.3.

To apply the most suitable algorithm, the problem must first be understood and categorised. An algorithm suitable for linear problems may not be suitable for non-linear problems, and vice versa. By incorrectly categorising a problem, an unsuitable optimisation category can be chosen, leading to invalid results, for example, a convex problem. This is a problem where the constraint functions are all convex, all minimising objectives are convex, and all maximising objectives are concave. These problems typically have only one optimal solution, and so every local solution is also a global solution. Using a global algorithm on a convex problem is generally computationally more expensive than a local one while still leading to the correct solution.

When selecting an algorithm, it should be noted also that there is not a single most effective algorithm that can be applied to all optimisation problems. Each algorithm has benefits and drawbacks. The main theorem of optimisation, the *no free lunch theorem* (NFL) [1]), states: if any algorithm A outperforms another algorithm B in the search for an extreme of an objective function, then algorithm B will outperform A over some other desired trait such as computational cost, accuracy or complexity. The NFL theorem suggests that the average performance overall possible objective functions is the same for all search algorithms. All algorithms

for optimisation will give the same average performance when averaged overall possible functions, which means that the universally best method does not exist for all optimisation problems. This theorem proves the importance of applying problem-specific information when deciding upon an appropriate algorithm to achieve better than average results.

7.1.2 Local vs Global Optimisation

There are two categories of optimal solutions that can be found as a result of an optimisation process: local solutions and global solutions. Mathematically, a local solution is a solution for which an optimal solution $\mathbf{x}^*_{local}$ is better than all other values of $\mathbf{x}$ in its neighbourhood. A global solution describes an optimal solution $\mathbf{x}^*_{global}$ which is better than all other values of $\mathbf{x}$ across the whole search space. As a result, all global minima are also local minima. This distinction highlights the importance of a correct problem formulation. For a linear, convex problem, a local solution is a global solution. In the case of a complex, non-convex problem, a local solution is not necessarily a global one. In this case the choice of the initial guess, from which the optimisation algorithm performs the search, can be crucial for the performance of the algorithm itself because of the possibility of converging into one of the local optima close to the initial guess rather than the global one. Hence global optimisation algorithms are designed with particular strategies that are aiming at avoiding being trapped in local optima.

7.1.3 Single- vs Multi-Objective

The objective functions drive the optimisation algorithm to find an optimum value, depending on whether the result has to be minimised or maximised. In single-objective optimisation, the main goal is to find the 'optimal' solution for only one objective function.

For a problem with more than one objective, there is rarely one solution that is the optimal solution for each of the objective functions. In this case, a set of optimal solutions is found. Finding the optimum solution for multiple objective functions can be difficult and computationally expensive. One method of simplification is to reduce the number of objective functions. Multiple objective functions can be lumped into one objective functions through a weighted sum approach, where the function outputs are scaled then multiplied a constant representing its importance relative to the other objectives. It should be noted that, although conceptually easy, the weighted sum approach only finds solution on the convex regions of the Pareto front and are difficult to implement when the objective functions have different orders of magnitude. Another method is the ϵ-constraint one, which considers all objectives except one, as constraints in the optimisation process.

These constraints are assigned different constants based on the importance of their respective objective functions (e.g. minimum reliability levels, maximum price), and multiple solution of a single-objective problem are found for different satisfaction levels of each constraint. A deeper discussion into multi-objective strategies is provided in Sect. 7.2.3.

7.2 Continuous Optimisation

7.2.1 Local Optimisation

Local optimisation algorithms are exact methods that guarantee the convergence to the local optimum in a neighbourhood of search. They are the most investigated optimisation techniques and have their roots in the calculus of variations and the work of Euler and Lagrange. The development of linear programming falls back to the 1940s, and it was the base of the modern optimisation theory that rapidly grew and then was developed in the last 70 years.

As already defined in the previous section, the general optimisation problem is defined as

$$\min_{\mathbf{x}\in D} f(\mathbf{x})$$

where $D = \{\mathbf{x} \in \Omega \mid c(\mathbf{x}) \leq 0\}$, $f : \Omega \to \mathbb{R}$ and $c : \Omega \to \mathbb{R}^m$ are sufficiently smooth functions. It must be pointed out that local optimisation techniques restrict their field of application to single-objective optimisation problems with continuous variables. To extend the use to multi-objective optimisation problems, one of the aggregate techniques presented above must be taken into consideration.

Before introducing the optimality results, some definitions need to be stated.

Definition 7.2.1 The real function $\mathscr{L} : \Omega \times \mathbb{R}^m \to \mathbb{R}$ defined as

$$\mathscr{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda^T c(\mathbf{x})$$

is the *Lagrangian*, and the coefficients $\lambda \in \mathbb{R}^m$ are called *Lagrange multipliers*.

Definition 7.2.2 Given a point $\mathbf{x}$ in the feasible region, the *active set* $\mathscr{A}(\mathbf{x})$ is defined as

$$\mathscr{A}(\mathbf{x}) = \{i \in \mathscr{I} \mid c_i(\mathbf{x}) = 0\},$$

where $\mathscr{I} = \{1, \ldots, m\}$ is the index set of the constraint.

Definition 7.2.3 The linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(\mathbf{x}), i \in \mathscr{A}(\mathbf{x})\}$ is linearly independent, that is,

$$\text{rank}(\nabla c_i(\mathbf{x}), i \in \mathscr{A}(\mathbf{x})) = |\mathscr{A}|.$$

Note that if this condition holds, none of the active constraint gradients can be zero.

7.2.1.1 Optimality Conditions

These definitions allow the statement of the following optimality conditions (refer to [2], for a proof of the Theorems).

Theorem 7.2.1 (First-order necessary condition) *Suppose that* $\mathbf{x}^*$ *is a local solution of the constrained non-linear programming (NLP) problem and that the LICQ holds at* $\mathbf{x}^*$. *Then a Lagrange multiplier vector* λ^* *exists such that the following conditions are satisfied at the point* $(\mathbf{x}^*, \lambda^*)$

$$\nabla_{\mathbf{x}} \mathscr{L}(\mathbf{x}^*, \lambda^*) = 0, \tag{7.1}$$

$$c(\mathbf{x}^*) \leq 0, \tag{7.2}$$

$$\lambda^* \geq 0, \tag{7.3}$$

$$(\lambda^*)^T c(\mathbf{x}^*) = 0. \tag{7.4}$$

These conditions are known as the Karush-Kuhn-Tucker (KKT) conditions.

Remark 7.2.1 The last condition implies that the Lagrange multipliers corresponding to inactive inequality constraints are zero; hence it is possible to rewrite the first equation as

$$0 = \nabla_x \mathscr{L}(\mathbf{x}^*, \lambda^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathscr{A}(\mathbf{x}^*)} \lambda_i^* \nabla c_i(\mathbf{x}^*).$$

The optimality condition presented above gives information on how the derivatives of objective and constraints are related at the minimum point $\mathbf{x}^*$. Another fundamental first-order necessary condition that gives additional information on the gradient of the objective function in the optimal point can be stated. For this an additional definition is needed.

Definition 7.2.4 Given a feasible point $\mathbf{x} \in D$, a sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ with $\mathbf{x}_k \in \Omega$ is a *feasible sequence* if, for all $k \in \mathbb{N}$, $\mathbf{x}_k \in D \backslash \{\mathbf{x}^*\}$ and

$$\lim_{k \to \infty} \mathbf{x}_k = \mathbf{x}.$$

Given a feasible sequence, the set of the limiting directions $w \in \Omega \backslash \{0\}$

$$\lim_{k \to \infty} \frac{\mathbf{x}_k - \mathbf{x}}{\|\mathbf{x}_k - \mathbf{x}\|_2} = \frac{w}{\|w\|_2}$$

is called the *cone of the feasible directions*, $C(\mathbf{x})$.

Moving along any vector of this cone (with vertex in a local minimum point $\mathbf{x}^*$) either increases the objective value or keeps it the same.

Theorem 7.2.2 (First-order necessary condition) *If $\mathbf{x}^*$ is a local solution of the optimisation problem and f is differentiable in $\mathbf{x}^*$, then*

$$\nabla f(\mathbf{x}^*) \cdot w \geq 0 \quad \forall w \in C(\mathbf{x}^*).$$

For the directions w for which $\nabla f(\mathbf{x}^*) \cdot w = 0$, it is not possible to determine, from first derivative information alone, whether a move along this direction will increase or decrease the objective function. It is necessary to examine the second derivatives of the objective function and constraints to see whether this extra information resolves the issue. The directions for which the behaviour of f is not clear from the first derivative form the following set:

Definition 7.2.5 Given a pair $(\mathbf{x}^*, \lambda^*)$ satisfying the KKT conditions

$$C(\lambda^*) = \{w \in C(\mathbf{x}^*) \mid \nabla c_i(\mathbf{x}^*) \cdot w = 0, \text{ for all } i \in \mathscr{A}(\mathbf{x}^*) \cap \mathscr{I}, \text{ with } \lambda_i^* > 0\}$$

is called the *critical cone*.

Indeed for $w \in C(\lambda^*)$ from the first KKT condition it follows that

$$\begin{aligned} \nabla f(\mathbf{x}^*) \cdot w &= \sum_{i \in \mathscr{A}(\mathbf{x}^*)} \lambda_i^* \nabla c_i(\mathbf{x}^*) \cdot w \\ &= 0. \end{aligned}$$

If $\mathbf{x}^*$ is a local solution, then the curvature of the Lagrangian along the directions in $C(\lambda^*)$ must be non-negative in the case of qualified constraints. A positive curvature is instead a sufficient condition for a local optimum.

Theorem 7.2.3 (Second-order necessary condition) *Let f and c be twice continuously differentiable; $\mathbf{x}^*$ is a local solution of the constrained problem and that the LICQ condition is satisfied. Let $\lambda^* \in \mathbb{R}^m$ be the Lagrange multiplier for which the pair $(\mathbf{x}^*, \lambda^*)$ satisfies the KKT conditions. Then*

$$w^T \ \nabla_{xx}^2 \mathscr{L}(\mathbf{x}^*, \lambda^*) \ w \geq 0, \quad \forall w \in C(\lambda^*)$$

Theorem 7.2.4 (Second-order sufficient condition) *Let f and c be twice continuously differentiable; $\mathbf{x}^*$ is a feasible point, $\lambda^* \in \mathbb{R}^m$ such that $(\mathbf{x}^*, \lambda^*)$ satisfies the KKT conditions and*

$$w^T \ \nabla_{xx}^2 \mathscr{L}(\mathbf{x}^*, \lambda^*) \ w > 0, \quad \forall w \in C(\lambda^*), w \neq 0.$$

Then $\mathbf{x}^*$ is a strict local minimum of the constrained problem.

7.2.1.2 Algorithms

In the last 50 years, a variety of approaches have been developed to solve NLP problems, first tackling the most simple unconstrained NLP problem and then expanding their application also to the constrained case. A starting point, denoted by $\mathbf{x}_0$, is always provided to the algorithm by the knowledge of the user or left to the optimiser. The optimisation process iterates exploiting information on the objective, constraints, their derivatives and the previous iterates to terminate whenever no further progress can be made or the optimal solution is approximated with acceptable accuracy.

The algorithm for unconstrained NLP is presented first. They are divided into two groups: line search based and trust region.

- **Line search**: the algorithm determines a search direction p_k and searches along this direction from the current iterate $\mathbf{x}_k$ for a new iterate with a lower function value. The step length to move along p_k can be found by approximately solving the minimisation problem

$$\min_{\alpha>0} f(\mathbf{x}_k + \alpha p_k).$$

  At the new point, a new search direction and step length are computed, and the process is repeated until convergence.
- **Trust region**: the algorithm constructs a model function m_k whose behaviour near the current iterate x_k is similar to that of the actual objective function f. The iteration direction of search p is found as the solution of the problem

$$\min_{p\in\Omega} m_k(\mathbf{x}_k + p),$$

  where $\mathbf{x}_k + p$ lies inside the trust region. If the solution does not produce a sufficient decrease in f, it means that the trust region is too large. In this case the trust region is shrunk and the minimisation problem is solved again. Usually the trust region is the ball

$$\|p\|_2 \leq \Delta, \quad \text{where } \Delta \text{ is the trust region radius}$$

  and the model m_k is usually a quadratic function of the form

$$m_k(\mathbf{x}_k + p) = f(\mathbf{x}_k) + p^T \nabla f(\mathbf{x}_k) + \frac{1}{2} p^T H(\mathbf{x}_k) p$$

  where H is the Hessian matrix of the Lagrangian.

The two approaches differ in the way they choose the direction and the distance of the move: line search based fixes the direction p_k and optimises the length of the step. Thrust region instead first chooses the maximum distance of the move, the trust

region radius, and then seeks for the best move to attain the best improvement of the objective function.

As an example for line search methods, there are (refer to [2] for the details about the methods):

- **Steepest descent method**: it chooses as search direction the descent one $p_k^{SD} = -\nabla f(\mathbf{x}_k)$.
- **Newton methods**: the search direction is the solution of the Newton equation $p_k^N = -H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$.
- **Non-linear conjugate gradient methods**: where the search direction is defined as $p_k^{CG} = -\nabla f(\mathbf{x}_k) + \beta_k p_{k-1}$ with $\beta_k \in \mathbb{R}$.
- **Quasi-Newton methods**: they don't require the computation of the second-order derivatives but use an approximation of it (B), $p_k^{QN} = -B(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$; quasi-Newton methods significantly increase convergence speed compared with Newton ones.

Newton and quasi-Newton methods are the ones that attain a superlinear rate of convergence, but they require the computation (or approximation) and the storage of the Hessian matrix. On the other hand, the methods that rely just on the gradient information are slower at convergence.

Most of the methods have a counterpart for the thrust region approach. In the quadratic model, the Hessian matrix is substituted by the one used by each method (identity matrix for the steepest descent, H_k for the Newton method and its approximation B_k for quasi-Newton methods). It is possible to prove that the resulting search direction is defined as in the line search methods and its length constrained by the trust region radius.

The presentation of the algorithms for the unconstrained case was necessary to introduce the techniques for solving constrained NLP problems as parts of them rely on the idea of converging to the solution of the constrained problem by approximating it with a sequence of unconstrained problems.

The algorithms for constrained NLP problems can be grouped in:

- **Penalty, barrier, augmented Lagrangian methods** and **sequential linearly constrained methods**: they solve a sequence of simpler subproblems (unconstrained or with simple linearised constraints) related to the original one. The solutions of the subproblems converge to the solution of the primal one either in a finite number of steps or at the limit.
- **Newton-like methods**: they try to find a point satisfying the necessary conditions of optimality (KKT conditions in general). The sequential quadratic programming (SQP) method is part of this class.

The *penalty methods* combine the objective function and constraints into a penalty function $\alpha(\mathbf{x})$ which is null for feasible points and positive otherwise. The problem to be minimised is the unconstrained problem

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}) + \mu\alpha(\mathbf{x})$$

for a series of increasing values of the penalty parameter μ, such that $\mu\alpha(\mathbf{x}) \to 0$ as $\mu \to \infty$, until the solution of the constrained optimisation problem is identified with sufficient accuracy. From a computational point of view, superlinear convergence rates might be achieved, in principle, by applying Newton's method to solve the minimisation problem (or its variants such as quasi-Newton methods). The algorithmic behaviour is strictly related to the choice of the penalty parameter. If μ is large, more importance is given to the feasibility than the optimality, and the iterates could move to feasible regions far from the optimum, causing slow convergence and premature termination.

The *barrier methods* or *interior-point methods* add terms to the objective function that act as a barrier and prevent the iterates from leaving the feasible region. For example, in the case of inequality constrained problems, a barrier problem can be formulated as

$$\min_{\mathbf{x}\in\Omega} \theta(\mu),$$

where $\mu \geq 0$ and $\theta(\mu) = \inf\{f(\mathbf{x}) + \mu b(\mathbf{x}) : c_i(\mathbf{x}) < 0, \forall i \in \mathscr{I}\}$. The barrier function b should be non-negative and continuous on the feasible region and go to infinity as the boundary is approached from the interior. This would guarantee that the iterates do not leave the domain. The starting point must be chosen in the interior of the feasible region, and the Newton or quasi-Newton methods can solve the successive barrier problem.

In the *augmented Lagrangian methods*, a penalty functions is added to the Lagrangian:

$$\mathscr{L}_A(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) - \lambda^T c(\mathbf{x}) + \frac{1}{2\mu}\|c(\mathbf{x})\|_2^2$$

Fixing λ to some estimate of the optimal Lagrange multipliers and $\mu > 0$ to some positive value, it is possible to find a value of $\mathbf{x}$ that approximately minimises $\mathscr{L}_A(\cdot, \lambda, \mu)$. Then the process is repeated updating λ and μ with the information from the previous $\mathbf{x}$-iterate.

In *sequential linearly constrained methods*, at every iteration, a Lagrangian is minimised subject to a linearisation of the constraints.

The *sequential quadratic programming* has instead a completely different approach. It employs Newton-like methods to solve directly the KKT conditions of the original problem. The problem turns out to be a minimisation problem of a quadratic approximation of the Lagrangian subject to a linear approximation of the constraints. The search direction p_k at the iterate $(\mathbf{x}_k, \lambda_k)$ is the solution of the problem

$$\begin{aligned} &\min_p \ \tfrac{1}{2} p^T \nabla^2_{xx} \mathscr{L}(\mathbf{x}_k, \lambda_k) p + \nabla f(\mathbf{x}_k) \cdot p \\ &\text{s.t.} \ \ \nabla c_i(\mathbf{x}_k) \cdot p + c_i(\mathbf{x}_k) \leq 0, \quad i \in \mathscr{I}. \end{aligned}$$

A trust region constraint can be added to the algorithm to control the length of the step, and a quasi-Newton approximation of the Hessian can be used instead of the second derivatives of the Lagrangian.

7.2.2 Global Optimisation

The effectiveness of the traditional local optimisation techniques on multimodal objective functions strongly depends on the initial guess solution given to a method. If a previous knowledge of the problem is available, the designer can provide a good initial guess to the algorithm to ensure convergence to the global optimum. Otherwise the algorithm will mostly fail in the global search, getting trapped in one of the multiple local minima.

The purpose of global optimisation is to find the best solution of a non-linear optimisation problem,

$$\min_{\mathbf{x}\in D} f(\mathbf{x})$$

in the presence of multiple optima and a non-smooth objective function.

Nevertheless, local optimisation techniques will often play an important role also in global optimisation strategies since some promising global approaches combine both global and local strategies of search. This is the case, for example, for memetic algorithms (MA) [3] that combine gradient-based technique with evolutionary algorithms: the global search generates a set of trial points over the feasible region (solutions of the evolutionary strategy), and the local algorithm performs local descent search from the best available points, in an iterative loop that alternates the two steps till convergence. Obviously the best compromise between global and local strategies and the effectiveness of their use depends on the characteristic of the problem such as the geometry of the feasible region, the number of local minima and the sharpness of the objective function in the neighbourhood of the global solution. However, the collaborative use of the global exploration capabilities of the first algorithm to prune the search space narrowing the area of search and the exploitation of local strategies to converge to the exact location of the global minimum is a very simple but effective approach for the local refinement of the selected global optimal solutions.

The limit of combining the two optimisation approaches may be related to the inefficiency of the local strategies in dealing with multiple-objective problems, especially when proper scalarisations are not considered. First, a brief overview to the available global optimisation algorithms and to the historical background that made them evolving to the actual state of the arts is given (see [4, 5] for a complete survey).

The methods that were first used in global optimisation were deterministic techniques. They were introduced in the late 1950s with the advent of the first

electronic computers into the research community. They are mostly based on the idea of trying to construct a sequence of approximate solutions which converge to the exact one by dividing the problem into smaller subproblems or approximations of the original one. With the evolution of the computational power at the beginning of the 1990s, different probabilistic global optimisation approaches were affirmed as new strategies. Among them it is worth to mention simulated and nested annealing [6, 7] and the large family of evolutionary strategies [8]. They are in general computationally less efficient than deterministic techniques, but due to their structure, they are able to tackle a wider range of problems, and no assumptions on the model regularity and smoothness is required. For these reasons they are considered as one of the most promising techniques for solving also global discrete non-linear optimisation problems.

There are several classifications of global optimisation strategies. One is the already mentioned division between **deterministic** and **stochastic** algorithms. In the first category, the model and the optimisation variables are completely known, and the algorithm performs through predefined steps. The stochastic component of the latter group instead lies either on the random sampling of the trial points, random parameters of the algorithm itself that made the single step not predictable, or on the use of a stochastic model for the objective function. Another division can be made between **exact** methods and **heuristic** methods. Exact methods provide a mathematical proof that the optimal solution can be found, while heuristic methods are not based on convergence theories. In most of the cases, no guarantee of finding the optimal solution can be provided and used to stop the search process: the optimisation process is constituted of iterative steps that improve the candidate solutions based on a measure of the quality of their fitness, a function that combines indexes of optimality and feasibility.

An overview of the relevant methods is given below according to the first classification deterministic or stochastic with an internal differentiation between exact and heuristic methods. The objective of the section is to give a comprehensive overview of the available methodologies. For details about a specific algorithm, please refer to the corresponding bibliography. The extension of the methodologies to the multi-objective case is discussed in the next section.

7.2.2.1 Deterministic Strategies

The first group are deterministic and exact global optimisation strategies [9]. It means that no randomness is involved in the optimisation process and the algorithm will always produce the same solutions for the same starting condition or initial state. The optimisation steps are predictable and a proof of convergence exists.

- **Uniform grid search** [4]: it is a trivial search strategy that makes use of a grid over the search domain to evaluate cost and constraints functions. Local search from a point in each element of the grid can be performed, and the feasible local minimum with lowest objective function is the approximation of the global

optimum. The success of the local search obviously depends on the finesse of the search grid, and global convergence can be trivially guaranteed by the fact that the mesh can be made arbitrarily dense. Such a simple scheme however rapidly becomes inefficient with the enlargement of the bounds on the optimisation variables and the raising of the dimension. The computational load will increase as an exponential function of the dimensionality of the problem.

- **Complete (enumerative) search** [5]: it is based on the simple principle of searching through all potentially optimum points in the search space, through enumeration of the possible candidates and evaluation of the objective. If, for example, the feasible region D is a polyhedra and the objective function is concave, then it is possible to prove that the problem must have a global optimal solution which is a corner of D. Since D has a finite number of extreme points, the problem could be solved by enumerating the extreme points of D in an appropriate way until an optimal solution is found [10]. Enumerative methods have few applications in continuous optimisation. Convergence properties are trivially provable.
- **Homotopy and trajectory methods** [11, 12]: the two strategies have the ambitious objective of visiting all stationary points of the objective function on the feasible domain, tracing the paths on the feasible space that include them. The solutions are then explored through enumeration techniques and evaluation of the objective. The two methods differ in the way of constructing their paths: the homotopy method makes use of homotopy transformations between the solution of a simplified problem and the original one; the trajectory problem solves a set of ordinary differential equations. The methodologies are applicable to smooth problems with continuous variables, and the enumeration techniques employed guarantee convergence to the optimum.
- **Sequential approximation (relaxation) methods** [13]: the idea is to build and solve a series of approximate (or relaxed) optimisation subproblems converging to the exact (or approximate) global optimum. A classification of such methods is based on the target of the approximation (relaxation), either specific model parameters or the entire system and subsystem models in a non-decomposed or decomposed problem, and the method employed to perform the approximated model fitting (response surface methodology (RSM), Taguchi methods, kriging [14]). The methods can be applied to a wide range of optimisation problems with continuous and discrete variables, and they are particularly suitable for expensive or noisy simulation models as a complete analysis is performed only in the experimental data points of the metamodeling techniques. The methods form a subset of the derivative-free optimisation techniques, based on model approximation, as they are completely free from derivative computation or approximation. Method-specific convergence theories are available in the suggested reference.
- **Interval arithmetic methods** [15]: it is possible to develop a complete theory based on interval entities analogous to the real one. The strength of exploiting the global information over large domains given by interval analysis in optimisation methods ensures the convergence to all global optima. The idea is to start with

an initial box and to delete the sub-boxes that cannot contain the global solution by a branch-and-bound procedure. The process terminates, when the bounds on the solutions and on the global minimum are below a predefined tolerance. The main drawback of the interval approach is its computational complexity. It is applicable to MINLP problems and non-smooth functions.

On the other hand, there is no proof of exactness for the following global deterministic strategy.

- **Sequential improvements of local optima** [16]: the basic idea is to generate an improving sequence of local minima. Deflection techniques, tunnelling and filled function methods are examples of this approach. The tunnelling method consists of two phases: seek for a local minimum and apply a tunnelling function to find a point in the domain that has the same value of the objective function. The newly formed point is the starting point for the next iteration. The process terminates when it is not possible to detect any point during the second phase. The last found local optimum is also the global one. There is no rigorously established convergence theory associated with these methods, and they are applicable only to smooth continuous optimisation problems.

7.2.2.2 Stochastic Strategies

Stochastic strategies are methods that contain not deterministic elements, either random generated algorithm parameters or stochastic approximations of model functions. As expected it is difficult to develop a rigorous convergence theory for such a class of algorithms, due to the randomness introduced in the optimisation process. However two of them provide a convergence proof based on probabilistic theories, and they can be classified as exact methods.

- **Random search methods** [17]: the objective of these search methods is to find the global minimum with an adaptive-probabilistic distribution of random points over the feasible region. These algorithms ensure that the global minimum will be found with probability one as the sample size grows to infinity. The difference to the deterministic grid search algorithm lies in its adaptivity. The number of experimental points doesn't need to be decided in advance, but it is generated in the successive steps. These methods are applicable to both discrete and continuous global optimisation problems with very mild assumptions on the model regularity.
- **Random function approach** [18, 19]: also known in literature as Bayesian methods, they are the stochastic counterpart of the sequential approximation approach with an adaptive probabilistic model for the approximation of the objective function. They are suitable for cost functions that have a highly computational load. They can deal with continuous and discrete variables and non-smooth functions. A theoretical convergence to the global optimum is guaranteed only by generating a dense set of search points.

The larger group of stochastic optimisation techniques are heuristic. They are most widely applied in practice, but in general no mathematical proof of convergence exists. However, some results on convergence for evolutionary methods, provided that the method satisfies some very general conditions, have been published for single- [20] and multi-objective [21] problems.

- **Two-phase methods** [22]: they are the stochastic counterpart of the deterministic grid search technique. They combine two phases of search: a global one and a local one. The process starts with a random sampling of the feasible space followed by the application of a local refinement. Multistart [22], clustering methods [23] and multilevel single linkage [24] are the examples. The range of applications for the technique is constrained to the local search used. The greedy global strategy is suitable for both continuous and discrete variables with no assumptions on the model structure.
- **Simulated annealing** [25]: the technique is based on the analogy between minimising a cost function and the cooling process of a material till it reaches its state of low energy equilibrium. The algorithm iteratively brings the actual state (optimisation variables) to a lower level of the internal energy of the system (objective function). The changes between the states are done probabilistically. The new configuration is constructed by imposing a random displacement at each step. If the energy of the new state is lower than the previous one, the change is accepted. If the energy is greater, the new configuration is accepted with a probabilistic value. The probabilistic acceptance of upward moves is aiming to avoid the convergence to the local minima. It is able to tackle global optimisation problems with discrete and continuous variables under mild assumptions on the model regularity.
- **Genetic algorithms** (GAs) [26]: are stochastic search methods that take their inspiration from natural selection and survival of the fittest in the biological world. Each iteration of a GA involves a competitive selection that eliminates poor solutions. The solutions with high fitness are recombined with other solutions by swapping parts of a solution with another. The solutions are also mutated by making a small change to a single element, or a small number of elements, of the solution. Recombination and mutation are used to generate new solutions that are biased towards the regions of the space for which good solutions have already been seen. GAs were born and are well suited, to solve discrete problems, and they have been successfully applied to continuous problems as well. Most of their efficacy is due to a powerful recombination operator, which, for this reason, becomes the main operator. The recombination operation used by GAs requires that the problem can be represented in a manner that makes combinations of the two solutions likely to generate interesting solutions. Selecting an appropriate representation is a challenging aspect to properly apply these methods. Usually a binary coding is used, and many applications have demonstrated the validity of this approach.
- **Estimation of distribution algorithms (EDA)**: with the idea that probabilistic modelling may offer a more efficient/effective way to treat real problems,

instead of using standard genetic operators used in traditional EAs, in EDAs new candidate solutions to the problem are generated using *regression*, i.e. estimating a probabilistic model based on the statistics collected from the set of candidate solutions (regression), and *sampling* the achieved probabilistic model, bringing a new paradigm in evolutionary computation. Because of the different natures of both optimisation and probabilistic modelling in discrete and continuous domains, developed EDAs also have differences depending on the representation type they use for the problem. Many of the early continuous EDAs as well as their recent improvements are based on the assumption that design variables can be characterised by Gaussian distribution. The continuous population-based incremental learning (PBIL_C) [27] extends the original discrete version to continuous domains by updating a vector of independent Gaussian distributions. The continuous univariate marginal distribution algorithm (UMDA_C) [28] uses maximum likelihood estimation to learn the parameters of the Gaussian distribution for each variable from the population of solutions. The continuous mutual information maximisation for input clustering (MIMIC_C) [28] learns the chain structured probabilistic model for continuous variables by adapting the concept of conditional entropy for univariate and bivariate Gaussian distributions.

Other probabilistic models estimate a non-parametric distribution for the variables have also been used in continuous EDAs. The multi-objective Parzen-based estimation of distribution (MOPED) [29] uses a Parzen estimator to build the probabilistic model. Both Gaussian and Cauchy kernels are used alternatively during evolution to exploit their complementary characteristics.

A review of methods and their characteristics can be found at [30].

- **Differential evolution (DE)** [31]: it is an optimisation method particularly suitable for multidimensional multimodal functions, belonging to the class of evolution strategy (ES). The main idea is to generate a variation vector by taking the weighted difference between two other solution vectors randomly chosen within a population of solution vectors and to add that difference to the vector difference between the considered solution and a third solution vector.

  An approach used to create new algorithms is to hybridise existing ones by appropriately mixing some of their building blocks. By following this approach, and based on some new theoretical results on the convergence of DE, the inflationary differential evolution algorithm (IDEA) [32] was proposed, combining DE with the restarting procedure of monotonic basin hopping (MBH) algorithm [33, 34]. Although IDEA showed very good results when applied to problems with a single or multi-funnel landscape, its performance was found to depend on the parameters controlling both the convergence of DE and MBH and the inflationary stopping criterion used to terminate the DE search.

  Despite its simplicity, the standard DE alone shows good performance on a broad range of problems featuring multimodal, separable and non-separable structures, but the performance is strongly influenced by three parameters: the population size, n_{pop}; the crossover probability, CR; and the differential weight (or step parameter), F. In addition, it was reckoned that the chosen strategies for mutation and crossover [35] plays an important role.

The need of self-adapting techniques especially for these two parameters has been widely recognised in the literature. In [36] the authors introduced a fuzzy adaptive differential evolution algorithm using fuzzy logic controllers to adapt the parameters for the mutation and crossover operators. The self-adaptive DE (SADE), described in [37], incorporates a mechanism that self-adapts both the parameters CR and F and the trial vector generation strategy. In [38] an adaptation strategy is proposed for parameter F, while CR is kept constant. In [39] both control parameters are added to each individual of the population and evolve with it. An alternative approach for the on-line adaptation of both CR and F parameters and embedded into the general framework of IDEA is proposed in [40]. The proposed approach uses the Parzen kernel method to build a joint probabilistic representation of the most promising region of the bivariate $CR - F$ space. The resulting probability density function (PDF) is updated during the optimisation process on the basis of obtained results. A further development of AIDEA is multi-population adaptive inflationary differential evolution algorithm (MP-AIDEA) [41] where multiple populations are initialised in the search space and exchange information during the optimisation process.

- **Particle swarm optimisation (PSO)** [42]: it is a population-based stochastic optimisation technique developed by Eberhart and Kennedy in 1995 [43], inspired by the social behaviour of bird flocking or fish schooling. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates in the problem space, which are associated with the best solution it has achieved so far. The particle swarm optimisation concept consists of, at each iteration, changing the velocity of each particle i according to a close-loop control mechanism.

7.2.3 Multi-Objective Optimisation

The problem of optimising concurrently two or more objective functions falls into the category of multi-objective optimisation problems. In contrary to single-objective optimisation, the purpose is not to find a unique global optimal solution but rather a set of solutions representing the compromise (trade-offs) between the different objectives.

Also in multi-objective optimisation, as in single-objective, it is possible to distinguish between local and global solutions: they will be referred as global frontier and local frontier.

The generic multi-objective optimisation problem is defined as

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0 \end{aligned}$$

where $\mathbf{x}$ is the optimisation variables vector, $f : \Omega \to \mathbb{R}^{n_{\text{obj}}}$, with $n_{\text{obj}} > 1$, the objective function and $c : \Omega \to \mathbb{R}^m$ the constraints function. As before the set of feasible points is denoted by D.

To extend the methodologies presented for global optimisation to the multi-objective case, it is necessary to introduce some definitions.

Definition 7.2.6 A point $\mathbf{x}_1 \in D$ *Pareto dominates* $\mathbf{x}_2 \in D$ if

$$f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2), \qquad i = 1, \ldots, n_{\text{obj}}$$

and there is at least one component $j \in \{1, \ldots, n_{\text{obj}}\}$ such that

$$f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2).$$

This is indicated by

$$\mathbf{x}_1 \preceq \mathbf{x}_2.$$

Definition 7.2.7 A point $\mathbf{x}^* \in D$ is *Pareto optimal* if it isn't dominated by any $\mathbf{x} \in D$,

$$\mathbf{x} \preceq \mathbf{x}^*.$$

In other words, a solution is said to be Pareto optimal, or equivalently *nondominated*, if there is no other point in the feasible space for which a decrease in one objective will not cause a simultaneous increase of at least one of the other objectives.

Definition 7.2.8 For a multiple objective optimisation problem, the *Pareto optimal set* is defined as

$$\mathscr{P}^* = \{\mathbf{x} \in D \mid \nexists \mathbf{x}' \in D \ \ \mathbf{x}' \preceq \mathbf{x}\}.$$

Definition 7.2.9 The union of the objective values of all Pareto optimal points is called *Pareto front* or equivalently

$$\mathscr{PF}^* = \{f(\mathbf{x}) \in \mathbb{R}^{n_{\text{obj}}} \mid \mathbf{x} \in \mathscr{P}^*\}.$$

The Pareto front is the set of all solutions in the feasible space that are not dominated by any other possible solution. The minima in the sense of Pareto will lie on the boundary of the feasible region or in the tangent points of the objective functions. Generally it is not possible to derive analytically the equation of the front. Approximation techniques have been developed during the years to approach the Pareto frontier by successive iterations or to solve in parallel a sequence of single-objective optimisation problems.

A comprehensive survey of multi-objective optimisation techniques is given in [44–46], the last two focusing mainly on global evolutionary multi-objective

strategies. Evolutionary programming is the area of multi-objective optimisation research that in the last years registered the fastest growth. This is due to the intrinsic structure of the evolutionary algorithms, population based, well suited for an extension to multi-objective problems.

The multi-objective approaches are divided in methods that use the concept of Pareto dominance for the selection mechanism of the next iterates and methods that develop a special handling of the objective functions for reformulating the problem as single objective. The latter techniques are applicable to all the presented global optimisation strategies, while the former are typically for evolutionary algorithms.

The aggregation of the multiple objectives into a common single objective can be achieved by the different techniques presented below, outlining their main advantages and disadvantages.

- **Weighted sum approach**: the objectives are aggregated into a single function using weighting coefficients. The optimisation problem becomes

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & \sum_{i=1}^{n_{\text{obj}}} w_i f_i(\mathbf{x}) \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0, \end{aligned}$$

  where $w_i \geq 0$ and it is usually assumed that

$$\sum_{i=1}^{n_{\text{obj}}} w_i = 1.$$

  By varying the values of the coefficients, different solutions on the Pareto front are traced. To cover the entire front, a sequence of single-objective optimisation problems needs to be solved, making the procedure very inefficient from a computational point of view. Moreover, this technique has the drawback of not generating proper Pareto optimal solutions in the presence of non-convex search spaces [47]. Additionally, there is no a priori knowledge about how a change in the weights will affect the position on the Pareto front of the new solution.
- **Goal programming** [48]: the designer has to assign targets to the objectives, and the optimisation problem is transformed in the problem of minimising the sum of the norms of the deviations from the targets

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & \sum_{i=1}^{n_{\text{obj}}} \| f_i(\mathbf{x}) - T_i \|_2 \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0. \end{aligned}$$

  Prerequisite in the application of such a technique is a deep knowledge about the optimisation problem to be able to assign meaningful target values to the objectives. The search space is explored by varying the T_i targets, and convergence to the Pareto front is achieved with a prior knowledge of the problem, to assign the targets close to the objectives values of the Pareto optimal points.

- **Goal attainment**: it is a combination of the previous two techniques. Objectives goals are assigned as before, together with relative under or over attainment weight coefficients. The problem becomes

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & \alpha \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0 \\ & f_i(\mathbf{x}) \leq T_i + \alpha w_i, \quad i = 1, \ldots, n_{\text{obj}}, \end{aligned}$$

  where $\alpha \in \mathbb{R}$ and the weights $w_i \geq 0$ are normalised so that

$$\sum_{i=1}^{n_{\text{obj}}} w_i = 1.$$

  It is possible to prove that the Pareto front can be covered varying the weight coefficients and the methodology is able to deal also with non-convex problems [49].
- **The ε constraint method**: the objectives are minimised one at a time, constraining the others below a certain level

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & f_j(\mathbf{x}) \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0 \\ & f_i(\mathbf{x}) \leq \varepsilon, \quad i = 1, \ldots, n_{\text{obj}}, \ i \neq j. \end{aligned}$$

  The main weaknesses of the approach are the same as listed above, computational efficiency, and a necessary a priori knowledge of the problem for covering the global Pareto front.
- **Lexicographic order**: the objectives are sorted by user intervention. The optimisation problem is divided in n_{obj} subproblems solved sequentially with a pre-established order and with additional constraints for not violating the satisfaction of the minimum values of the former subproblems. Assuming that $\{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_{n_{\text{obj}}}(\mathbf{x})\}$ are the ordered objectives and f_i^* the minimum value achieved for the i-th objective. Then the i-th subproblem is defined as

$$\begin{aligned} \min_{\mathbf{x}\in\Omega} \quad & f_i(\mathbf{x}) \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0 \\ & f_j(\mathbf{x}) = f_j^*, \quad j = 1, \ldots, i-1. \end{aligned}$$

  To cover the Pareto front, different optimisation runs with different sequences of objectives must be performed, heavily increasing the overall computational time.
- **Game theory**: a 'player' is assigned to each objective function. The player has the goal to minimise its objective. Assuming that the players are playing a non-cooperative game (i.e. the players make decisions independently), the

intersection of the best strategy of each player is a *Nash equilibrium*, in the sense that no player can deviate unilaterally from this point for further improvement of the proper objective.

- **Weighted min-max approach**: the deviations from the attained minima, in the n_{obj} single-objective subproblems are estimated for the i-th objective as

$$\bar{z}_i(\mathbf{x}) = \frac{\|f_i(\mathbf{x}) - f_i^*\|_2}{\|f_i^*\|_2}, \quad \bar{\bar{z}}_i(\mathbf{x}) = \frac{\|f_i(\mathbf{x}) - f_i^*\|_2}{\|f_i(\mathbf{x})\|_2}$$

assuming that the objective values do not vanish.

Defining $z_i(\mathbf{x}) = \max\{\bar{z}_i(\mathbf{x}), \bar{\bar{z}}_i(\mathbf{x})\}$, the desirable solution of the multi-objective problem is the one that gives the smallest values of all increments of all the objective functions

$$\min_{\mathbf{x}\in D} \max_{i\in\mathscr{I}} \{z_i(\mathbf{x})\},$$

where $\mathscr{I}$ is the set of the objective indexes. The entire front can be covered by weighting the deviation function.

Note that some of the scalarisation approaches, such as the weighted sum, the goal attainment and the ϵ constraint, can be obtained as particular cases of the Pascoletti−Serafini scalarisation scheme [50, 51].

The exploitation of the concept of Pareto dominance in the population-based strategies led in the current years to the development of efficient multi-objective global optimisation techniques. The particular structure of the algorithms, based on a family of solutions that evolves at each step, made the introduction of the concept of Pareto dominance in its ranking process possible [52]. The basic idea is to find a set of solutions that are Pareto nondominated by the rest of the solutions of the feasible set, assign to them the highest rank and remove them from the group. The process then repeats recursively for lower values of the rank. This procedure can be applied for sorting the solutions of a current iteration and selecting a subgroup from it to apply the criteria of evolution of the species, resulting in a next generation of solutions that is different from the previous one and has an average better fitness.

Genetic algorithms are the larger class of evolutionary algorithms. They are divided in two groups:

- **First generation**: they are characterised by the introduction of the concept of Pareto dominance in the process of selection of the population and for the niching operator to maintain the diversity and avoid premature convergence to local fronts. Representative algorithms of this class are multi-objective genetic algorithm (MOGA) [53], nondominated sorting genetic algorithm (NSGA) [54] and niched Pareto genetic algorithm (NPGA) [55].
- **Second generation**: they exploit the concept of *elitism*. This means that they use an external archive to store the nondominated solutions found in the previous generation in a way that the best solutions found in every iteration cannot be lost

during successive iterations and a better global minima frontier can be achieved. The algorithms than differ in the way they interact with the external population. Representative algorithms of this class are strength Pareto evolutionary algorithm (SPEA) [56, 57], NSGA2 [58], Pareto archived evolution strategy (PAES) [59], Pareto envelope-based selection algorithm (PESA) [60, 61] and micro-genetic algorithm (Micro-GA) [62, 63].

Another group of population-based algorithms not classifiable as genetic algorithms already mentioned in the previous section gets inspired by natural phenomena such as the cooling state of a metal or the behaviour of an ant colony in the search of food. A corresponding reformulation of the already presented algorithms is available for multi-objective optimisation problems. Namely, they are multi-objective simulating annealing (MOSA) [64], multi-objective particle swarm optimisation (MOPSO) [65] and multi-objective ant colony optimisation (MOACO) [66].

7.2.4 Optimal Control

The general statement of an *optimal control problem* (OCP) requires the definition of [67]:

- The mathematical model of the dynamic system to control
  Usually it is described by a system of ordinary differential equations (ODEs) in the form $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))$. The independent variable has been indicated by t, usually appointed as time, but there is no restriction on its choice. The variables x_i in the vector of $\mathbf{x}$ are usually called *state variables*, while u_j in the vector $\mathbf{u}$ are the *control variables*.
- The performance index J to be minimised (or equivalently maximised)
  The performance index in the general form is written as:

$$J = \phi[t_f, \mathbf{x}(t_f)] + \int_{t_0}^{t_f} L[t, \mathbf{x}(t), \mathbf{u}(t)]dt \tag{7.5}$$

The *optimal control problem* is in the Bolza form if both the end-cost and the integral terms are present. If the end-cost term ϕ is zero, it is known as a Lagrange problem. On the contrary, if the integral term L is zero, the problem is referred as a Mayer one. Mathematically these formulations are equivalent and convertible into each other. For example, a Lagrange problem can be restated as a Mayer one by simply adding one state variable of the form $\dot{x}_{n+1} = L[t, \mathbf{x}(t), \mathbf{u}(t)]$, leading to $J = x_{n+1}(t_f)$. However, [68] states that, even if they are mathematically equivalent, they are not numerically corresponding. The Lagrange form shall be preferred as the Mayer form leads to an increased number of state variables, which are then discretised in numerical methods, leading to a higher size of the NLP subproblem and a more time-consuming algorithm.

- Specification of constraints

  They are divided into two different classes, i.e. *fixed-event* or *path* constraints. The first type is described as an algebraic function of the state and control $g_L^f \leq g^f[(\bar{t}_j), \mathbf{y}(\bar{t}_j), \mathbf{u}(\bar{t}_j)] \leq g_U^f$ at a fixed time $\bar{t}_j$. The initial and final boundary conditions fall into this form for $g_L^f = g_U^f$. A *path constraint* is formulated as an algebraic function of the state and control variables $g_L^p \leq g^p[(t), \mathbf{x}(t), \mathbf{u}(t)] \leq g_U^p$ over a trajectory's phase. Bounds on the control magnitude fall into this category as $\mathbf{u}_L \leq \mathbf{u}(t) \leq \mathbf{u}_U$. This general notation [68] deals with both equality and inequality constraints, depending on the lower and upper boundary values.

Once the aforementioned statements have been formulated, the *optimal control problem* aims to find the control profile $\mathbf{u}^*(t)$, in the space of all admissible controls U, which minimises the performance criterion J while respecting the differential model $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))$ and the specified physical constraints. Briefly stated:

$$\begin{aligned} &\min \;\; J = \phi[t_f, \mathbf{x}(t_f)] + \int_{t_0}^{t_f} L[t, \mathbf{x}(t), \mathbf{u}(t)]dt, \quad u \in U \\ &\text{subject to} : \dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \\ &\qquad\qquad \mathbf{g}_L^p \leq \mathbf{g}^p[(t), \mathbf{x}(t), \mathbf{u}(t)] \leq \mathbf{g}_U^p \\ &\qquad\qquad \mathbf{g}_L^f \leq \mathbf{g}^f[(\bar{t}_j), \mathbf{x}(\bar{t}_j), \mathbf{u}(\bar{t}_j)] \leq \mathbf{g}_U^f \end{aligned} \tag{7.6}$$

where the Bolza formulation is used to obtain the necessary conditions in the most general case.

7.2.4.1 Indirect Methods

Indirect methods are based on Pontryagin's maximum principle, adapting the sign convention for minimisation problem. This principle's derivation employs *calculus of variations* techniques, of which comprehensive references are [69] and [70]. The goal is to convert the *optimal control problem* as defined in the chapter's introduction into a *two-point boundary value problem* through the statement of the necessary conditions that a profile shall satisfy to be an optimal solution.

The process starts with the definition of an *augmented performance index* $\bar{J}$, in a fashion similar to equality-constrained static optimisation problems, where Lagrange's multipliers λ_j multiplying the dynamical constraints are summed to the objective function to form the *augmented performance index*:

$$\bar{J} = \Phi + \int_{t_0}^{t_f} \left[L[t, \mathbf{x}(t), \mathbf{u}(t)] + \boldsymbol{\lambda}^T(t) \left\{ \mathbf{f}[t, \mathbf{x}(t), \mathbf{u}(t)] - \dot{\mathbf{x}} \right\} \right] dt \tag{7.7}$$

According to the *calculus of variation*, the necessary conditions for a stationary extremum is that the first-order variation $\delta \bar{J}$ shall nullify at any instant of time for any constraint-allowed variation $\delta \mathbf{u}(t)$. The problem *Hamiltonian* is defined as:

$$H = L[t, \mathbf{x}(t), \mathbf{u}(t)] + \boldsymbol{\lambda}^T(t)\mathbf{f}[t, \mathbf{x}(t), \mathbf{u}(t)] \tag{7.8}$$

When path constraints are present, the Hamiltonian shall be augmented with the constraints' violation weighted by associated dual variables. After mathematical manipulation (see [67] for a detailed derivation), the necessary conditions for a control profile $\mathbf{u}^*(t)$ to be a stationary function of the performance index are represented by the following *Euler-Lagrange equations*:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \\ \dot{\boldsymbol{\lambda}} &= -\left[\frac{\partial H}{\partial \mathbf{x}}\right]^T \\ \mathbf{0} &= \left[\frac{\partial H}{\partial \mathbf{u}}\right]^T \end{aligned} \tag{7.9}$$

where the relations in Eq. (7.9)-2 are labelled as *adjoint equations* and Eqs. (7.9)-3 as *control equations*. These differential equations, which a control profile has to necessarily satisfy to be a stationary solution, are coupled with a set of *transversality conditions*:

$$\begin{aligned} t_0 \ \text{given} \quad &\vee \quad H(t_0) = 0 \\ t_f \ \text{given} \quad &\vee \quad H(t_f) = -\left.\frac{\partial \Phi}{\partial t}\right|_{t_f} \\ \mathbf{x}(t_0) \ \text{given} \quad &\vee \quad \boldsymbol{\lambda}(t_0) = 0 \\ \mathbf{x}(t_f) \ \text{given} \quad &\vee \quad \boldsymbol{\lambda}(t_f) = \left.\frac{\partial \Phi}{\partial \mathbf{x}}\right|_{t_f} \end{aligned} \tag{7.10}$$

Hence, if any of the boundary conditions is a free parameter, either on time or state variables, the above conditions complete the minimum required number of known conditions at the initial or final time. Up to this point, the process defined the necessary conditions for a solution to be a stationary one. The *Legendre-Clebsch* condition about local convexity of the *Hamiltonian* shall be satisfied to ensure that the solution is an actual local minimum:

$$\left.\frac{\partial^2 H}{\partial \mathbf{u}^2}\right|_{\mathbf{u}^*} \geq 0 \tag{7.11}$$

The *TPBVP* defined by Eq. (7.9), coupled with the conditions (7.10) and (7.11), has no analytical closed-form solution for complex problems Hence, numerical methods shall be employed. However, further information can be obtained by exploitation of the problem's first integrals. If the functions L and $\mathbf{f}$ defined in the System (7.6) do not depend explicitly on the independent variable t, then the *Hamiltonian* is a first integral of the TPBVP along an optimal trajectory [71]. In general, if a first integral is found, the redundant information that it generates can be exploited to eliminate one *adjoint equation*, formally transforming the original TPBVP into another one of lower dimension, by following the procedure shown by Visser [67].

7.2.4.2 Direct Methods

A direct method does not require the derivation of the necessary conditions needed by indirect methods. On the contrary, it aims to find a sequence of profiles which progressively reduce the non-augmented performance index J and the constraint's violation. Direct methods require a parametrisation of the control functional form over trajectory's arcs. This is generally achieved by two conceptually different methods [71]:

- A grid at different times where the control parameters are to be found and the values within an interval are computed through interpolation.
- A set of orthogonal basis of mathematical functions dependent on time. Usually Fourier series, Legendre polynomials or the Chebyshev ones.

The goal is then to determine the values of the specified free parameters, either control values at fixed times in the grid form or the coefficients of the series in the second case, able to minimise the objective index and to respect the constraints. In this step, the number of free parameters is reduced from infinite degrees of freedom to a finite number of parameters, depending on the chosen parametrisation. This passage could seem a limitation of the direct methods when compared to the indirect ones. However, as already stated in the previous section, a numerical procedure is necessary also for indirect methods when dealing with complex cases such as low-thrust trajectory optimisation. These numerical methods require a so-called transcription to convert the infinite-dimension optimal problem into a solvable finite-dimension one. Hence, what seemed a limitation of the direct methods is a required passage of any technique nonetheless.

A direct method's solution is generally not an optimal solution itself, i.e. not a local minimum of the performance index, but just an approximation as a consequence of the discretisation or interpolation steps. Hence, the necessary conditions (7.9) and (7.11) can be used as an indicator of how close the found solution is to the real local optimum [72].

7.2.4.3 Comparison of Direct and Indirect Methods

Loosely comparing an optimal control problem to a static constrained optimisation, the direct method's goal is to pinpoint a local minimum of the performance function, while an indirect method aims to find a root of the necessary conditions. The latter shall be preferred when a closed-form solution is aimed for. Indeed, indirect methods allow to extract the control in an analytical way [73, 74]. However, this is possible only when several approximations are employed or simplified cases are considered. When a numerical approach is necessary, a direct method often results to be the simpler choice due to several considerations [68]:

- The quantities $\left[\frac{\partial H}{\partial \mathbf{x}}\right]^T$ and $\left[\frac{\partial H}{\partial \mathbf{u}}\right]^T$ needed by indirect methods must be analytically computed and changed when different models are employed Furthermore, when a problem is divided into phases, these quantities change along the trajectory. This requires an extensive preliminary analytical stage for any different problem in the matter. On the contrary, a direct method is a flexible approach, more suitable for *black box* implementations, and able to handle a problem divided into different phases.
- Path inequalities, which are quite ordinary in low-thrust applications, represent a relevant issue for indirect methods. Indeed, a first guess of the *active-inactive* sequence is needed for practical methods as it changes the form of the Hamiltonian, by adding the *Lagrange multipliers*, the number of constrained arcs and the junction conditions. However, a priori knowledge of the right series is quite hard to achieve.
- Another issue with first guesses emerges from the initial estimate of the *adjoint variables* $\boldsymbol{\lambda}$. As remarked by Bryson and Ho [75], the extremal solutions can be very sensitive to small changes in the unspecified boundary conditions. As usually the initial state variables are specified, the *transversality conditions* (7.10) show that the initial values of the adjoint variables for the optimal trajectory are not known. Further, these variables are not representing physical quantities. Hence, setting the right initial conditions, or even reasonable ones, is very complex, and a bad initialisation often results in numerically ill-conditioned solutions. On the contrary, direct methods disregard those variables and require only initial guesses on the physical state and control variables.

7.2.4.4 Practical Techniques for Optimal Control

As stated in numerous occasions, in general the continuous optimal control problem does not have a closed-form solution, and practical numerical optimisation methods come into play. Any numerical technique cannot handle an infinite-dimension problem, but it needs a discrete problem with a finite set of variables and constraints to work with. This transition can be performed with conceptually different methods which will be investigated in the present section. It is important to emphasise that

the following techniques are applicable to both indirect and direct approaches. A complete review of the common methods, with a focus on low-thrust trajectory optimisation, has been compiled by Betts [76], whereas in this section two major classes will be addressed.

Single Shooting

Typically, the single shooting method does not actually find application in the field of complex non-linear optimal control. However, it is useful to introduce the notation and several concepts shared by its extension, the *multiple shooting* method. The discretisation grid is composed by only two points, the initial and final times. Initially, the n free parameters in $\mathbf{y}^T = [\bar{x}_1, \ldots, \bar{u}_{n_c}]$, composed by the initial conditions and the control parameters, are guessed. Hence, the trajectory is propagated forward (or equivalently backward) from the starting to the end time, leading to the final state:

$$\mathbf{x}_f^p = \mathbf{x}_0 + \int_{t_0}^{t_f} \mathbf{f}(t, \mathbf{x}, \mathbf{u}) dt$$

In general the propagated state $\mathbf{x}_f^p$ will not coincide with the required final one $\mathbf{x}_F$. Hence, the difference between these two quantities becomes a constraint to nullify. In literature, this constraint is generally labelled as *defect*:

$$\mathbf{c}(\mathbf{y}) = \mathbf{x}_f^p - \mathbf{x}_f \tag{7.12}$$

The numerical values of the violation of the boundary conditions can be exploited to iteratively adjust the control parameters with NLP algorithms in order to finally solve the constrained minimisation.
The advantage of this basic method is that the NLP subproblem has only a small number of variables to optimise, i.e. the initial state guess and control parameters. However, for long time-scales and non-linear dynamics, even small changes in the parameters can result in very large defects change, leading to hypersensitivity with respect to the free parameters.

Multiple Shooting

In order to overcome the drawback of parameter sensitivity, it is possible to segment the overall time interval into a set of $m-1$ smaller steps discretising the interval at m grid points $t_0 < t_1 < t_2 < \cdots < t_f$. Then, each of the segments can be treated as an independent single shooting method, with continuity constraints added. Therefore, first guesses of the n_s state variables for each intermediate segment are now needed. The first guess trajectory is usually found by fast and low-fidelity methods. The

state variables at intermediate grid points are now control variables to be optimised. Hence, the number of control parameters in $\mathbf{y}$ increases with respect to the single shooting method, precisely $n_y = (m-1)(n_s + n_c \cdot n_p)$, where n_s is the number of state variables, n_c the control components and n_p the control parameters per each component. The *defect* equations can be expressed in the general form as:

$$\mathbf{c}(\mathbf{y}) = \begin{pmatrix} \mathbf{x}_2^p - \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_f^p - \mathbf{x}_f \end{pmatrix} \tag{7.13}$$

where again the goal is to nullify $\mathbf{c}(\mathbf{y})$. The dimension that the NLP subproblem shall solve, in order to link the different phases and minimise the objective function, dramatically increases with an increasing number of steps. However, the effects of changing a particular parameter are more intuitive for smaller steps, leading to an improvement of the convergence properties. In addition, the main drawback of the single shooting is solved, and, when the number of steps is high enough, the variables related to the first stages of the trajectory do not heavily influence the last phases. The segment decoupling mathematically translates into very sparse *Jacobian* and *Hessian* matrices, later involved by the NLP algorithm. For example, the Jacobian gets sparser and sparser as more phases are employed, because the percentage of non-zero elements is proportional to $1/(m-1)$. This sparsity can be exploited to construct a computationally efficient non-linear programming subroutine, making the multiple shooting method both robust and competitive [77].

Collocation

The basic goal of *collocation* methods is to avoid repeated propagations over each segment. This is achieved by partitioning again the whole trajectory into $m-1$ segments, leading to m grid points. Hence, the trajectory is only represented by the set of state variables $\mathbf{x}(t_k)$ and their derivatives $\mathbf{f}(t_k, \mathbf{x}(t_k), \mathbf{u}(t_k))$ at mesh points as well as the control profile nodes $\mathbf{u}(t_k)$. As these values are treated as NLP variables, gathered in the vector $\mathbf{y}$, the optimal control problem has been completely transcribed into a finite-dimensional NLP. For this reason, also collocation methods need a first guess solution, which can be sought with the aforementioned approaches. The state, state-derivative and control values within each interval are computed by interpolation through piecewise functions, usually Hermite (third order), Chebyshev or Lagrange polynomials (see [68] for detailed schemes) or Fourier series [78], whose coefficients depend on the adjacent grid points' state and derivatives. This a priori shape replaces the numerical integration process of shooting techniques with a much faster analytical propagation.

The differential equations $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t))$ are substituted by a discretised form, which for a simple Euler scheme takes the following form:

$$\dot{\mathbf{x}} = \mathbf{f}(t_k, \mathbf{x}_k, \mathbf{u}_k) \approx \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{h} \tag{7.14}$$

where h is the interval size $t_{k+1} - t_k$. This Euler form is then transformed into a set of NLP constraints to be nullified:

$$c_k(\mathbf{y}) = \|\mathbf{x}_{k+1} - \mathbf{x}_k - h\mathbf{f}(t_k, \mathbf{y}_k, \mathbf{u}_k)\| \tag{7.15}$$

These constraints, which ensure the equation of motion to be approximately satisfied, are then coupled with the fixed-event ones and the path constraints, to construct a continuous trajectory and to respect the requested bounds at the grid points.

In collocation methods the choice of the interval size is vital because it influences the accuracy of the interpolated function in representing the true trajectory. An efficient procedure could be to compute initial estimates with a sparse grid and then refine it progressively. This implementation makes this technique very robust to imprecise initial guesses. Also in this method, the sparsity of the matrix shall be exploited as much as possible to make the algorithm efficient.

The greater drawback of collocation methods is that for problems dominated by highly non-linear dynamics, a very dense grid is needed to compute an accurate solution which, when integrated forward for validation, leads to small errors in the final state. This problem arises from the finite-difference approximation of the dynamics, as in Eq. (7.14) for an Euler scheme, and from the parametrisation of the shape. However, a dense grid translates into an expensive matrix inversion during the NLP subproblem, leading to the degradation of the computational performance.

Pseudospectral methods are a special class of direct collocation where the optimal control problem is transcribed by parameterising the state and control using global polynomials and collocating the differential-algebraic equations using the nodes obtained from a Gaussian quadrature [79, 80]. The terms pseudospectral and orthogonal collocation are used interchangeably in the literature.

7.3 Combinatorial and Network Optimisation

Until now, all optimisation problems and variables have been continuous, that is, each design vector has consisted of a set of variables composed of possible values within a specified range. In this section ‘combinatorial optimisation’ problems shall be discussed, where some or all of the design variables are restricted to a discrete set, most commonly binary integers, but also non-negative integers. This section will also discuss problems with a finite number of possibilities, namely, problems formulated to find a maximum or minimum of one or more functions, with many variables, which can be limited by a series of equality constraints, inequality constraints and bounds. All of these problems are ‘linear problems’ or can be reformulated as such with the inclusion of some constriction on one or more of the

design variables to ensure that only discrete values can be considered, described as 'linear integer programs'. If the design vectors are pure (or all) integer, the problem is classed as a 'pure integer program', whereas if at least one (but not all) design vector integer, the problem is classed as a 'mixed-integer program', discussed in Sect. 7.3.2.1. Alternatively, these problems can be categorised into general non-negative integer problems or 'binary', where the discrete, integer design vectors hold a value of either 0 or 1. Most mixed-integer problems in practice are binary, and the use of integer variables, although beneficial in certain circumstances, is much less common.

Often in 'combinatorial optimisation', the phrases 'combinatorial', 'discrete' and 'integer' are used interchangeably with little explanation of the differences between them. Each of the three terms can be used to describe a problem or optimisation method formulated for use with integers, as opposed to continuous variables, as inputs and outputs of the problem or as components of the optimisation process. Often, the term **discrete**, when used to describe problems and processes, is used to simply describe the discrete nature of one or more aspects of said process, i.e. a 'discrete problem', as opposed to a continuous problem. It is not accurate to say that a discrete problem is always an integer problem, e.g. if a problem has discrete design variables $\mathbf{x} = 0, 0.3, 0.6, 0.9, 1.2$, the problem is considered 'discrete', but not integer. Similarly, the term **combinatorial** describes the problem formulation but can also be used to describe the origin or solution of a problem, and is categorised by the exponential explosion of variables or constraints, often modelled by 'integer programming'. Finally, the phrase **integeri**, with respect to optimisation, is usually intended to describe the use of integer values in formulation or solution and thus also modelling. The similarities between these terms allows for a certain degree of interchangeability, though it is important to know where each term should and should not be used.

Integer programming was first recognised in the 1940s to 1950s, where the simplex algorithm (derived in 1948 and published in 1951) described a finite method suitable for application on any linear objective function subject to a finite set of linear constraints [81]. It was not until 1955 that Harold Kuhn derived a combinatorial algorithm for a single, specific integer problem using a dual-primal linear algorithm. Since then, a number of papers have expanded upon the available techniques and solving algorithms, introducing these tools to a wider and wider audience, such that many modern-day application rely on integer programming techniques.

Many real-world problems require the evaluation of integer problems; thus the interest in and knowledge applicable to optimisation of these problems are highly valued and ever-increasing. Many industries require the use of integer programming to solve practical problems. Communications, activity management, resource management, time scheduling and machine sequencing are vital to the cost minimising, resource management and time management of large commercial and industrial firms, whereas other problem applications are less grounded in real-life scenarios, such as high-energy physics and X-ray crystallography. These problems are generally more difficult to solve than problems that are linear and/or

continuous. Although the true advantage of combinatorial optimisation methods lies in the ability to process indivisible, discrete, real-life parameters, this optimisation category can also be utilised to convert continuous inputs to integer-only inputs with the intention of providing yes-no output values (which can be formulated as 0-1 integer problems), a particularly useful trait in machinery diagnosis.

Network optimisation is a special form of linear programming, where the structure of the program allows even faster solution approaches such as network simplex algorithm, and they are highly valued in their ability to optimise some of the most common, fundamental problems with minimal cost and a free flow of data to and from each network node. Analytically, network flow problems can solve some classes of combinatorial optimisation problems, such as shortest path, assignment and transportation. Network flow problems, although often complex, are utilised in the design and analysis of large connected problems, proving to be vital to the operation of many transportation, communication, manufacturing and social networks. Network optimisation methods make use of powerful techniques such as data caching, streamlining of data protocols and even data elimination. These techniques, when correctly applied, can assist in developing faster data transfers, accurate transport solutions and improved response times for software applications.

7.3.1 Pure Integer Optimisation

As introduced earlier, integer-only programming is a form of combinatorial optimisation developed for cases where all design vectors are integer, i.e. $\mathbf{x} \in \{0, 1, 2 \ldots\}$. Mathematically speaking, the original problem formulation, shown in the Introduction section of this chapter, can be altered to describe the case of an integer-only problem. Integer-only or 'pure integer' optimisation can make use of combinatorial optimisation algorithms since the search space, and hence the number of potential solutions is finite. Moreover, in a constrained integer-only problem, the number of potential solutions is limited by the number of possible combinations of every integer. For smaller problems, an exhaustive search may be used to evaluate each point in the design space, but this cannot be extended to larger problems due to the curse of dimensionality. This can be seen visually by Fig. 7.1, a simple integer-only problem with two integer inputs and three linear inequality constraints, as formulated in Eqs. (7.16) to (7.19).

$$\mathbf{x} = [x_1\ x_2] \in \{0, 1, 2 \ldots\} \tag{7.16}$$

$$A = [3\ -1] \tag{7.17}$$

$$b = [0\ -10] \tag{7.18}$$

$$lb = [0\,0]; \quad ub = [10\,10]; \tag{7.19}$$

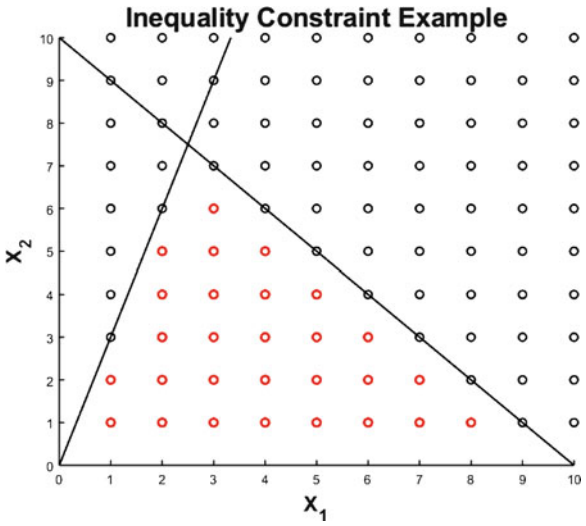

Fig. 7.1 Integer problem with linear constraints

This can be seen graphically in Fig. 7.1. We note that the number of points available for evaluation by the function are extremely limited and clearly seen as finite and by extension, the number of different values provided by the function is finite. This is due to the small number of dimensions. As the number of dimensions increases, the number of possible points increases exponentially. For simple problems such as this, an exhaustive search can be used. Complications arise upon the introduction of additional dimensions of the design vector where the size of the search space increases drastically depending upon the bounds, as described by the 'curse of dimensionality'.

7.3.1.1 Special Case: 0-1 Integer Programming

As introduced earlier, 0-1 integer programming or 'binary programming', is a special category of integer-only problem where the design variables are either one or zero (binary). This problem formulation is most commonly used for decision-making, when the inputs to a function is one of only two possible values: yes/no, open/closed, true/false, etc. [82].

Its mathematical formulation is

$$\sum_{j=1}^{n} c_j^T \mathbf{x}_j \tag{7.20}$$

$$\text{subject to: } A_{i,j}\mathbf{x}_j \leq b_i \tag{7.21}$$

$$\mathbf{x}_i = \{0, 1\} \tag{7.22}$$

Many real-life decision-making problems involve significant yes/no decisions, most often at strategic and tactical levels. The knapsack problem is a classic example of this type of problem. Moreover, other non-binary problems can be converted into this form if beneficial, where values can be split into ones and zeros to represent high/low temperatures, fast/slow speeds and other extremes. This can be added in practice by introducing equality constraints such that $0 = \mathbf{x}$ and $1 = \mathbf{x}$. Take a condition monitoring system which uses a temperature input to determine if a particular part has overheated. Binary programming may be applied where temperatures above a certain value are considered a failure (1) and temperatures below this are considered acceptable (0).

7.3.2 Mixed-Integer Programming

Mixed-integer programming, although commonly utilised in many modern-day problems, was developed following the formulation of the simplex method, developed by Dantzig in 1951. This was followed by the work from Ford and Fulkerson, whose earliest contribution to network flow began with 'maximal flow through a network', which is often credited as the original algorithm designed to solve maximum flow problems, and thus is considered one of the most influential papers in the development of further algorithms used for solving and analysing network flow models [83]. The simplex method also gave way to the first pure integer optimisation algorithm developed by Gomory in 1958. The increase in complexity resulted in an increase in computational cost, best modelled by a polynomial-time algorithm. These models allowed for the categorisation of problems into categories depending on hardness, where integer programming was considered NP-hard in general.

More complex problems may require the use of mixed-integer variables. That is to say that one or more variables of function $f(\mathbf{x}_{m+n})$ are a set of continuous variable(s), $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m$, and other variable(s) that are discrete values $\mathbf{x}_{m+1}, \mathbf{x}_{m+2}, \ldots \mathbf{x}_{n+m}$.

Mixed-integer problems cannot be solved by a continuous variable-based solver, as the step size, Δx, may be unsuitable for discrete variables. Take the steepest descent algorithm as an example: if the step size, λ, is not compatible with the design variables (in this case, not an integer), this will result in the failure of the algorithm and thus, no solution. Conversely, discrete solvers may disregard step sizes for continuous variables that are smaller than one, thus increasing the inaccuracy in the solver. This acts as a form of proof of the NFL theorem but also highlights the need for the development and correct application of more integer programming methods and categories, such as linear and non-linear.

7.3.2.1 MIP vs MINLP

Mixed-integer programming can be further categorised into mixed-integer programming (MIP) and mixed-integer non-linear programming (MINLP). Problems can be categorised by the nature of the objectives and constraints with respect to the design vector. Mixed-integer programming follows the standard linear programming formulation, where the objectives and constraints are linear with respect to the design variables which, in this case, consist of at least one design parameter composed of integers $(0, 1, 2 \ldots n)$ and at least one design parameter composed of continuous values, described in Eq. (7.23).

$$\text{Minimise} \quad c^T(x) \tag{7.23}$$

where

$$A\mathbf{x} \geq b$$

$$\mathbf{x} \geq 0$$

$$x_j \in Z \quad \forall j \in I$$

Commercial solvers such as IBM Ilog Cplex, FICO Xpress and Gurobi as well as open-source solvers such as COIN-OR all employ the branch-and-bound algorithm at their core, where the solution is iteratively parted into smaller subproblems, most commonly referred to as the left and right child problems (with respect to the original problem), discussed further in Sect. 7.3.2.2. Mixed-integer problems can also be solved by iteratively solving the so-called separation problem, where the feasible region of the problem is cut off by adding valid 'cuts' (i.e. additional constraints) and hence by elimination of sections of the design space. This is commonly known as the 'cutting plane algorithm'. Where the branch-and-bound algorithm employs LP relaxation to simplify the subproblems, the cutting plane algorithm tightens the LP relaxation to find a better approximation of the convex hull. All MIP solvers employ the so-called branch-and-cut method, which combines these two major solution methods for a more effective solution process.

A mixed-integer non-linear programming problem consists of at least one design parameter composed of discrete integers $(0, 1, 2 \ldots n)$ and at least one design parameter composed of continuous values, similar to MIP. However, in this case, either the objective or at least one of the constraints is non-linear with respect to design vectors. These problems follow the form:

$$\text{Minimise} \quad f(\mathbf{x}) \tag{7.24}$$

where

$$A_i(\mathbf{x}) = 0 \quad \forall i \in E \tag{7.25}$$

$$b_i(\mathbf{x}) \leq 0 \quad \forall i \in I \tag{7.26}$$

$$\mathbf{x} \in \{x_1, x_2 \ldots x_{n+m}\} \tag{7.27}$$

where $x_1, x_2 \ldots x_n$ are integer variables and $x_{n+1}, x_{n+2} \ldots x_{n+m}$ are continuous variables. These problems are particularly complex, combining the combinatorial difficulty of integer optimisation with non-linear functions, and so few algorithms have been designed specifically for use with these problems. Most solution algorithms for MINLPs fall into one of the two categories: single-tree and multi-tree methods [84]. MINLPs can be solved using a generalised Benders' decomposition (multi-tree), where a general MIP problem is employed with non-linear programming subproblems, and a modified branch-and-bound method (multi-tree), where modifications are made to improve the performance of the algorithm. Complications arise during the optimisation of a non-convex MINLP since even after relaxation of the integer design variables to continuous design variables, the function can remain non-convex, resulting in many local minima.

Many MINLP methods break the problem into their MIP and NLP components and solve the overall problem iteratively.

7.3.2.2 Methods

The algorithm applied to solve a combinatorial problem is dependent largely on the possible formulation of the problem and the requirements of the user. As per the 'no free lunch' theorem, a single algorithm cannot find the best possible solution to all possible problems. To find the most suitable solution method, the basic problem formulation must be considered: linear/non-linear, small/large, single-objective/multi-objective and binary/integer/mixed-integer.

Similarly to integer-only and continuous-only programming, combinatorial problems can be categorised as linear and non-linear. Also similarly to integer-only and continuous-only problems, linear problems are typically less complex than non-linear problems. As a result, fairly large, moderately complex problems can be solved using 'exact' methods, where every part of the problem and its subproblems are solved either explicitly or implicitly. For larger, more complex problems, 'heuristic' methods may be used. Heuristics use intuitive techniques to find a 'rough' solution to any given problem to a certain degree of accuracy.

Exact Methods

With relatively simple problems with low computational costs, exact methods can be used to solve combinatorial problems. These methods, unlike heuristic methods described in Sect. 7.3.2.2, guarantee an optimal solution and thus are the ideal choice. These solution methods can be placed into one of the two categories: implicit enumeration or explicit enumeration. Explicit solution methods are often

the simplest way of solving a problem using all possible solutions (an extensive search), but due to the 'curse of dimensionality', the complexity of the problem rises exponentially with the increasing number of dimensions within the design vector. As such, explicit enumeration is a valuable tool for small integer problems where dimensionality is limited. For larger problems, implicit enumeration is used.

Within explicit enumeration, the optimisation algorithm builds all the possible solutions to find the optimal solution. This is typically more costly than implicit enumeration, where all possible solutions are considered in some manner without explicit evaluation. Implicit enumeration methods consist of a wide variety of possible optimisation algorithms, where the most common are 'divide-and-conquer' methods, where a problem is divided into sets of m groups of problems iteratively until a subproblem is simple enough to be solved, and the branch-and-bound method, as discussed below.

Branch and Bound

In 1960, Alison Doig and Ailsa Land published a paper entitled 'An Automatic Method for Solving Discrete Programming Problems', introducing the concept of branch-and-bound algorithms. Although first intended to solve combinatorial optimisation problems, many improvements have been made to generalise the algorithm to solve continuous problems and improve the efficiency.

When solving MIP problems, the branch-and-bound method does not consider integer design variables as discrete values but rather converts these discrete values to continuous values by relaxation of the integer restrictions. This simplifies manipulation of the problem and thus, decreases the difficulty to solve.

The 'branch-and-bound' method consists generally of three main techniques: branching, bounding and searching.

- Branching
  - This step splits the continuous search space into several smaller subspaces, eliminating infeasible parts of the continuous space through application of necessary conditions for integer solutions.
- Bounding
  - The method of bounding depends on whether the objective function is to be maximised or minimised. If this function is to be maximised, an upper bounding strategy is used, and if minimised, a lower bounding strategy is applied.
- Searching
  - The process of searching each subspace for an optimal solution, preferably the most promising region first.

To begin the branching process, the search space S is split into a number of smaller and mutually disjoint subsets $S_1, S_2, \ldots S_r$. Following this partition, each subspace is analysed to find a local, feasible minimum, where each subset is also a set of feasible solutions of a 'candidate problem', which is found by imposing

additional constraints on the original function. The search space thought to contain the 'best solution' is then analysed. If the optimal solution is found, the subspace, and thus also the candidate problem, is fathomed. If not, the problem only contains a lower bound for the minimum objective value in it, and this subspace is divided into yet smaller subspaces (or candidate problems), and the process is repeated. This process can be adapted for specific problems. Consider problems with 0-1 variables. To branch these problems, extra constraints can be added to constrict $\mathbf{x}_1 = 0$ or $\mathbf{x}_1 = 1$, creating two candidate subproblems. In this case, $\mathbf{x}_1$ is known as the branching variable

The 'bound' step of the branch-and-bound algorithm is dependent on the objective of the objective function. Assuming that the objective function z is to be minimised, lower bounding strategies are required. Any lower bounding strategy should be simple, efficient and run with a low computational cost. In any case, a lower bounding method should bound closest to the minimum value of z, which can be handled using one of the many strategies [85].

- **Relaxation of constraints:** All difficult or computationally costly constraints are relaxed, and z is minimised for only the remaining constraints. Using this method, the minimum value of z is equal to the lower bound for z_{min} in the original problem.
- **Modification of the objective function:** In this case, the modified objective function is created such that $f \leq z$ for all feasible solutions. Furthermore, f should be easy to minimise subject to the original constraints. Subject to these properties, f is a lower bound to z_{min} of the original problem.
- **Lagrangian Relaxation:** A Lagrangian multiplier is created where u in $L(u, x)$ is associated with the relaxed constraints. In this case, the optimum z is a lower bound of z_{min} of the original problem.
- **Branch-and-cut:** Otherwise known as 'cutting planes', this iterative method solves the LP relaxation at each solution, and depending on if the solution is optimal or not, it is either accepted (if optimal) or a linear constraint is found that excludes the LP solution and no others. This constraint is referred to as a 'cut'.

The branch-and-bound algorithm is searched using a 'search tree method'. In this case, the original solution is analysed and branched, splitting the problem into two candidate problems. These two problems are bound, analysed and branched. Candidate problems which do not contain an optimal solution are not branched and become terminal nodes. Candidate problems which contain an optimal solution are further branched, and the process repeats. Terminal nodes may be required in further iterations of the algorithm. This search method continues to branch until an optimal solution is found.

Since its introduction in 1960, the branch-and-bound method has been slightly altered for improved results on specific problems. One such example of this is the 'Beale and Small' method [86]. This method uses a different bounding strategy, includes the termination of particularly non-optimal subspaces and includes a heuristic 'worst alternative' branching method.

For any strategy, the lower bound must be fairly close to the minimum objective value and generate candidate problems where the lower bounds are as high as possible. The computational time of the strategy, including calculation of the lower bounds for every candidate problem, must be low enough that the algorithm can be iterated many times.

Heuristic Methods

Methods to solve combinatorial optimisation problems discussed so far have been exact, i.e. finding the global solution is guaranteed (if there is a feasible solution). When heuristics are involved, this is not the case. Heuristics provide alternative methods for finding solutions to challenging problems (in particular in real-world settings) that do not guarantee an optimal solution (some of them only in statistical way). We note that this is different than approximation algorithms, which provide a performance guarantee such as maximum deviation from the optimal solution. Conversely to deterministic sampling, heuristic sampling requires a distribution of sample points over the search space with a higher density of points in areas of particular interest. Common heuristics include (a) relaxation-based heuristics and (b) rounding-based heuristics.

These methods are valid for only convex or small-scale non-convex MINLPs. There is no method yet that can reliably solve large-scale MINLPs, and, when compared, the algorithms that exist to solve convex MINLPs do not show a clear 'best algorithm', as can be expected. Where MINLP algorithms lack in computational speed and other desirable characteristics, a mixed-integer problem (MIP) is often used as a replacement for large-scale, real-world problems. Even without non-linear constraints, these problems can still be extremely hard, actual NP-hard [87].

The nearest neighbour heuristic and the Christofides algorithm [88] are well-known start heuristics for the TSP. The k-OPT-algorithm is an improvement heuristic which was originally designed for the TSP, but variants of this are used for several other combinatorial optimisation problems. It also formed the basis for the Lin-Kernighan heuristic [89] which is one of the most common algorithms used to find good solutions for TSPs. Balas and Martin [90] presented the pivot-and-complement that was developed for binary programs (BPs) and is based on the observation that, in the nomenclature of the simplex algorithm, an LP-feasible solution of which all basic variables are slack variables is also integer feasible. It performs pivot operations which drive the integer variables out of the basis and the slacks into the basis. The same authors [91] developed another method called pivot-and-shift that can be applied to general MIPs. The method was further improved with more pivot types and new rules for selecting them, as well as an extension of the shifting procedure, and a neighbourhood search related to local branching [92].

Another method is the so-called heuristic ceiling point algorithm, which was restricted to integer problems (IPs) without equality constraints. Scatter search with star paths is a diversification heuristic [93] that creates a couple of points which are

then linked by paths along which feasible solutions are searched. The main goal of Scatter search is to diversify the set of solutions and not improving the incumbent.

The Octahedral Neighbourhood Enumeration (OCTANE) search is a heuristic for BPs based on a ray shooting algorithm starting at the LP-optimum and hitting the facets of the octahedron dual to the unit hypercube [94].

In more recent years, some large neighbourhood search heuristics have been presented, such as the local branching [92] and the relaxation Induced Neighborhood Search (Rins) [95].

7.3.3 Network Optimisation

Network optimisation is a special type of linear programming, where variables are represented as flows in a network. Many real practical problems can be formulated as a 'network optimisation' problem, most commonly very large problems including the study of traffic, train and population flow, distribution analysis and communication problems. Consequently, many optimisation non-specialists understand the importance of these optimisation algorithms, which led to the widespread use of network optimisation in the testing and devising of new theories. This problem-solving method can be used to solve a series of combinatorial problems, for example [96, 97]:

- Space-time networks [98]
  - Traffic flow simulating, airline scheduling [85]
- Physical networks
  - Designing of streets and pipelines [99] to best manage flow
- Route networks
  - Vehicle route flows, map route optimisation (e.g. bus routes) [100]
- Constructing matches
  - Bipartite matching, survey design

Please note that problems such as TSP, VRP and scheduling may be represented using a network, but that does not mean they are network optimisation problems. Network optimisation is still LP and does not contain any integer variables; then the problems can be solved very effectively.

Standard Network Flow Formulation and Notation

A typical 'network' is a series of nodes (or vertices) connected by arcs (or edges), where each node is associated with a new design value and each arc is associated

with some category of moving value. To minimise unnecessary problem evaluations, it is assumed that no points are part of any 'self-loop', and so an arc from one point cannot lead back to the same point. A problem with many arcs and/or edges can be categorised into one of three forms: 'directed network', where only arcs are present; 'undirected network', where only edges are present; or 'mixed network' if there is a combination of arcs and edges [85]. In literature, arcs and edges are often considered as the same entity, and, in the following, they will be generally referred to as lines.

This problem formulation can be optimised for different specific problem requirements. This option to include specification of algorithm type allows for maximisation of accuracy and efficiency, as per the NFL theorem. Problems best suited for network optimisation include:

- Shortest path problem
  - Shortest path problems are some of the most commonly encountered network optimisation problems both in transportation and in communication.
- Maximum flow problem (as discussed)
  - Find a feasible flow path from a single source to a single sink, such that the flow is maximised.
- Minimum weight spanning tree
  - This problem requires each node to connect to every other node. If the links between nodes are expensive, it may be desirable to have each node connect to only two other nodes.

This can be formulated mathematically by considering a graph or directed network $G = (N, A)$, where N is a series of nodes (otherwise known as 'points' or 'vertices') such that $N = \{1, 2, 3 \ldots m\}$ and A is a series of lines $A = \{a_1, a_2, a_3 \ldots a_n\}$ [96, 101], with a cost $c_{i,j}$ and a capacity associated with every line or arc $(i, j) \in A$. These problems can be shown pictorially by placing all nodes and connecting lines on a plane, as can be seen in Fig. 7.2.

This problem can also be described mathematically using a graph. Unless otherwise specified, it can be assumed that the edges are distinct such that if $a = (i, j)$ then $i \neq j$, this would generate a 'simple' graph. Many network problems can be formulated in this way, where N could be a set of locations, A a set of

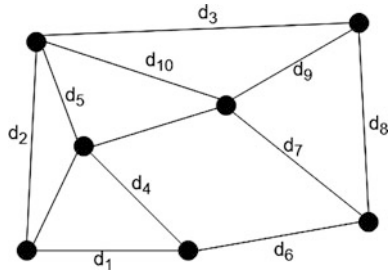

Fig. 7.2 Visual representation of a network

potential routes between these cities and c a set of distances. Typically, network problems are bound by both flow constraints and flow bounds. The upper flow bound or 'capacity' of an arc is denoted commonly by k_{ij}, describing the maximum possible quantity of material that can be moved across each node.

$$l_{ij} \leq f_{ij} \leq k_{ij} \tag{7.28}$$

For any given problem, there is a 'balance' constraint for each node, where basically the net flow from this node (i.e. outflow-inflow) will be equal to the 'supply' of this node. The supply b_i of node i is either positive (e.g. if this node is a location providing entries into the network), negative (e.g. if this node is a client with a demand) or zero (if the node plays a location for transhipment). Then, the balance constraint will be in the following form [102]:

$$\sum_{j \in N} f_{ij} - \sum_{j \in N} f_{ji} = b_i \quad \text{Flow Balance Equations} \tag{7.29}$$

Provided the network follows these bounds and constraints, and the supplies of the nodes are balanced, the network will be valid.

7.4 Summary

This chapter gives a brief introduction to optimisation problem formulations and solution methods. After a general overview of different problems, the chapter is mainly divided in two main sections: continuous problems and methods and discrete problems and methods.

The first section on continuous problems is further divided into four parts, related to local methods, optimal control, global methods and multi-objective optimisation. On the other hand, the section on discrete problems is composed by three parts on pure integer optimisation, mixed-integer optimisation and network optimisation.

This chapter is meant to give an accessible introduction to formulation and solving methods. The reader is kindly invited to use the list of references and read the following chapters, to know more about the methods and to see what are the most recent advances in the field.

References

1. D.H. Wolpert, W.G. Macready, *No Free Lunch Theorems for Optimization* (IEEE, Piscataway, 1997)
2. J. Nocedal, S.J. Wright, *Numerical Optimisation* (Springer, Berlin, 1999)

3. J. Knowles, D. Corne, *Memetic Algorithms for Multiobjective Optimization: Issues, Methods and Prospects* (IEEE Press, Piscataway, 2000), pp. 325–332
4. J.D. Pintr, *Global Optimization in Action* (Springer, Berlin, 1996)
5. R. Horst, P.M. Pardalos, H.E. Romeijn, *Handbook on Global Optimization: Nonconvex Optimization and Its Applications* (Springer, Berlin, 1995)
6. S. Rajasekaran, On simulated annealing and nested annealing. J. Glob. Optim. **16**, 4356 (2000)
7. M. Locatelli, Simulated annealing algorithms for continuous global optimization. J. Optim. Theory Appl. **104**, 121–133 (2000)
8. T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (Oxford University Press, Oxford, 1996)
9. C.A. Floudas, *Deterministic Global Optimization* (Springer, Berlin, 2000)
10. P.M. Pardalos, Enumerative techniques for solving some nonconvex global optimization problems. OR Spektr. **10**, 29–35 (1988)
11. W. Forster, *Homotopy Methods*. Handbook of Global Optimization: Nonconvex Optimization and Its Applications (Kluwer, Dordrecht, 1995), pp. 669–750
12. I. Diener, *Trajectory Methods in Global Optimization*. Handbook of Global Optimization: Nonconvex Optimization and Its Applications (Kluwer, Dordrecht, 1995), pp. 649–668
13. R. Horst, H. Tuy, *Global Optimization: Deterministic Approaches*, 3rd edn. (Springer, Berlin, 1996)
14. T.W. Simpson, J. Peplinski, P.N. Koch, J.K. Allen, Metamodels for computer-based engineering design: survey and recommendations. Eng. Comput. **17**(2), 129–150 (1995)
15. E.R. Hansen, G.W. Walster, *Global Optimization Using Interval Analysis* (CRC Press, Boca Raton, 2003)
16. A.V. Levy, S. Gomez, The tunneling method applied to global optimization, in *Numerical Optimization* (SIAM, Philadelphia, 1985), pp. 213–244
17. F.J. Solis, R.J.-B. Wets, Minimization by random search techniques. Math. Oper. Res. **6**, 19–30 (1981)
18. H.J. Kushner, A versatile stochastic model of a function of unknown and time varying form. J. Math. Anal. Appl. **9**, 379–388 (1962)
19. J. Mockus, On bayesian methods of optimization, in *Toward Global Optimization*, ed. by L.C.W. Dixon, G.P. Szegö (North Holland, Amsterdam, 1975)
20. G. Rudolph, Convergence of evolutionary algorithms in general search spaces, in *IEEE International Conference on Evolutionary Computation* (1996), pp. 50–54
21. G. Rudolph, Evolutionary search under partially ordered fitness sets, in *International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI 2001)* (2001), pp. 818–822
22. A.H.G. Rinnooy Kan, G.T. Timmer, Stochastic methods for global optimization. Am. J. Math. Manag. Sci. **4**, 7–40 (1984)
23. A.H.G. Rinnooy Kan, G.T. Timmer, Stochastic global optimization methods, part I: clustering methods. Math. Program. **39**, 27–56 (1987)
24. A.H.G. Rinnooy Kan, G.T. Timmer, Stochastic global optimization methods, part II: multi level methods. Math. Program. **39**, 57–78 (1987)
25. J.M. Laarhoven Peter, H.L. Aarts Emile, *Simulated Annealing* (Springer, Berlin, 1987)
26. M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, 1998)
27. M. Sebag, A. Ducoulombier, Extending population-based incremental learning to continuous search spaces, in *5th International Conference on Parallel Problem Solving from Nature (PPSN V)*. Lecture Notes in Computer Science, vol. 1498 (Springer, Berlin, 1998), pp. 418–427
28. P. Larrañaga, R. Etxeberria, J. Lozano, J. Peña, Optimization in continuous domains by learning and simulation of Gaussian networks, in *Conference on Genetic and Evolutionary Computation (GECCO00) Workshop Program*, pp. 201–204. (Morgan Kaufmann, San Mateo, 2000)

29. M. Costa, E. Minisci, MOPED: a multi-objective Parzen-based estimation of distribution algorithm for continuous problems, in *Evolutionary MultiCriterion Optimisation 2003*. Lecture Notes in Computer Science, vol. 2632 (Springer, Berlin, 2003), p. 71
30. P. Larrañaga, H. Karshenas, C. Bielza, R. Santana, A review on probabilistic graphical models in evolutionary computation. J. Heuristics **18**(5), 795–819 (2012)
31. K.V. Price, R.M. Storn, J.A. Lampinen, Differential evolution, in *A Practical Approach to Global Optimization, Natural Computing Series* (Springer, Berlin, 2005)
32. M. Vasile, E. Minisci, M. Locatelli, An inflationary differential evolution algorithm for space trajectory optimization. IEEE Trans. Evol. Comput. **15**(2), 267–281 (2011)
33. D.J. Wales, J.P.K. Doye, Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. J. Phys. Chem. A **101**, 5111–5116 (1997)
34. B.Addis, M. Locatelli, F.Schoen, Local optima smoothing for global optimization. Optim. Methods Softw. **20**, 417–437 (2005)
35. S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011)
36. J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm. Soft Comput. A Fusion Found. Method. Appl. **9**(6), 448–462 (2005)
37. A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. **13**(2), 398–417 (2009)
38. M.M. Ali, A. Trn, Population set based global optimization algorithms: some modifications and numerical studies. Comput. Oper. Res. **31**(10), 1703–1725 (2004)
39. J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**(6), 646–657 (2006)
40. E. Minisci, M. Vasile, Adaptive inflationary differential evolution, in *Congress on Evolutionary Computation (CEC2014)*, July 6–11, Beijin (2014)
41. M. Di Carlo, M. Vasile, E. Minisci, Multi-population adaptive inflationary differential evolution algorithm with adaptive local restart, in *Congress on Evolutionary Computation (CEC2015)* (2015)
42. M. Clerc, *Particle Swarm Optimization* (ISTE, London/Newport Beach, 2006)
43. J. Kennedy, R. Eberhart, Particle swarm optimization, in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4 (1995), pp. 1942–1948
44. K. Miettinen, *Nonlinear Multiobjective Optimization* (Springer, Berlin, 1999)
45. C.A. Coello Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowl. Inf. Syst. **1**, 269–308 (1998)
46. C.M. Fonseca, P.J. Fleming, An overview of evolutionary algorithms in multiobjective optimization. Evol. Comput. **3**(1), 1–16 (2007)
47. J. Brian, J. Ritzel, E. Wayland, S. Ranjithan, Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. Water Resour. Res. **30**(5), 1589–1603 (1994)
48. Y. Ijiri, *Management Goals and Accounting for Controls* (North-Holland Publishing Company, Amsterdam, 1965)
49. Y. L. Chen, C.C. Liu, Multiobjective VAR planning using the goal attainment method. IEE Proc. Gener. Transm. Distrib. **141**(3), 227–232 (1994)
50. L.A. Ricciardi, C.A. Maddock, M. Vasile, Direct solution of multi-objective optimal control problems applied to spaceplane mission design. J. Guid. Control. Dyn. **42**(1), 30–46 (2019)
51. M. Vasile, Multi-objective optimal control: a direct approach, in *Satellite Dynamics and Space Missions*, ed. by G. Baú, A. Celletti, C. Gales, G. Federico Gronchi (Springer, Berlin, 2019)
52. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Boston, 1989)

53. C.M. Fonseca, P.J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization, in *Genetic Algorithms Proceedings of the Fifth International Conference*, San Mateo (1993), pp. 416–423
54. N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol. Comput. **2**(3), 221–248 (1994)
55. J. Horn, N. Nafpliotis, Multiobjective optimization using the Niched Pareto Genetic algorithm, IlliGAL Report n.93005, University of Illinois (1993)
56. E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)
57. E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm, TIK-Report 103 (2001)
58. K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist Multi-Objective Genetic Algorithm: NSGA-II, KanGAL Report No. 200001 (2000)
59. J.D. Knowles, D.W. Corne, Approximating the nondominated front using the Pareto Archived Evolution Strategy. Evol. Comput. **8**(2), 149–172 (2000)
60. J.D. Knowles, D.W. Corne, M.J. Oates, *The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization*. Lecture Notes in Computer Science (2000), pp. 839–848
61. D.W. Corne, N.R. Jerram, J.D. Knowles, M.J. Oates, PESA-II: region based selection in evolutionary multiobjective optimization, in *Proceedings of the Genetic and Evolutionary Computation Conference* (2001)
62. C.A. Coello Coello, G. Toscano Pulido, *A Micro-Genetic Algorithm for Multiobjective Optimization*. Lecture Notes in Computer Science (2001), pp. 126–140
63. C.A. Coello Coello, G. Toscano Pulido, *The Micro Genetic Algorithm 2: Towards On-Line. Adaptation in Evolutionary Multiobjective Optimization*. Lecture Notes in Computer Science (2003), pp. 75
64. P. Czyzak, Pareto Simulated Annealing, a meta-heuristic technique for multiple objective combinatorial problems. J. Multi-Criteria Decis. Anal. **7**(1), 34–47 (1998)
65. C.A. Coello Coello, G. Toscano Pulido, M.S. Lechuga, Handling multiple objectives with Particle Swarm Optimization. IEEE Trans. Evol. Comput. **8**, 256–279 (2004)
66. K. Socha, M. Dorigo, Ant colony optimization for continuous domains. Eur. J. Oper. Res. **185**(3), 1155–1173 (2008)
67. H.G. Visser, Aircraft Performance Optimization. Delft University of Technology (2014)
68. J. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, 1st edn. (Society for Industrial & Applied Mathematics, Philadelphia, 2001)
69. H. Goldstein, *Classical Mechanics*, 3rd edn. (Pearson, London, 2001)
70. G. Bliss, *Lectures on the Calculus of Variations*, 1st edn. (University of Chicago Press, Chicago, 1946)
71. S. Kemble, *Interplanetary Mission Analysis and Design*, 1st edn. (Springer, Berlin, 2006)
72. B.A. Conway, *Spacecraft Trajectory Optimization* (Cambridge University Press, New York, 2010)
73. R.H. Bishp, D.M. Azimov, Analytical space trajectories for extremal motion with low-thrust exhaust-modulated propulsion. J. Spacecr. Rocket. **38**(6) (2001)
74. J.A. Kechichian, Optimal low-thrust transfer using variable bounded thrust. Acta Astronaut. **36**(7) (1995)
75. A. Bryson, Y. Ho, *Applied Optimal Control* (John Wiley & Sons, Hoboken, 1975)
76. J. Betts, Survey of numerical methods for trajectory optimization. J. Guid. Control. Dyn. **21**(2), 193–207 (1998)
77. C. Greco, Variational multiple shooting: theory and applications. Delft University of Technology (2017)
78. F. Zuiani, M. Vasile, Direct transcription of Low-Thrust trajectories with finite trajectory elements, in *61st International Astronautical Congress*, Prague (2010)
79. C. Canuto, M.Y. Hussaini, A.M. Quarteroni, T.A. Zang, *Spectral Methods in Fluid Dynamics*. Springer Series in Computational Physics (Springer, Berlin, 1988)

80. B. Fornberg, *A Practical Guide to Pseudospectral Methods* (Cambridge University Press, Cambridge, 1998)
81. M. Jnger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, L.A. Wolsey, *50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art* (Springer, Berlin, 2009)
82. A. Kaufmann, A. Henry-Labordeère, *Integer and Mixed Programming: Theory and Applications*, vol. 137 (Elsevier, Amsterdam, 1977)
83. M. Josefsson, M. Mützell, Max Flow Algorithms Ford-Fulkerson, Edmond-Karp, Goldberg-Tarjan Comparison in regards to practical running time on different types of randomized flow networks. KTH Computer Science and Communication, Stockholm (2015)
84. P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, A. Mahajan, Mixed-integer nonlinear optimization. Acta Numer. **22**, 1–131 (2013)
85. K. Murty, *Linear and Combinatorial Programming* (John Wiley & Sons, Inc., New York, 1976)
86. J. Abadie, *Integer and Nonlinear Programming* (North-Holland Pub. Co., Amsterdam, 1970)
87. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency* (Springer, Berlin, 2003)
88. N. Christofides, Worst-case analysis of a New Heuristic for the travelling salesman problem. GSIA report 388, Carnegie-Mellon University (1976)
89. S. Lin, B.W. Kernighan, An effective heuristic algorithm for the travelling-salesman problem. Oper. Res. **21**, 498–516 (1973)
90. E. Balas, C.H. Martin, Pivot-and-complement: a heuristic for 0-1 programming. Manag. Sci. **26**(1), 86–96 (1980)
91. E. Balas, C.H. Martin, *Pivot-and-Shift: A Heuristic for Mixed Integer Programming* (GSIA, Carnegie Mellon University, Pittsburgh, 1986)
92. M. Fischetti, A. Lodi, Local branching. Math. Program. **98**(1), 23–47 (2003)
93. F. Glover, A. Lkketangen, D.L. Woodruff, Scatter search to generate diverse MIP solutions, in *OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research* (2000)
94. E. Balas, S. Ceria, M. Dawande, F. Margot, G. Pataki, Octane: a New Heuristic for pure 0-1 programs. Oper. Res. **49**(2), 207–225 (2001)
95. E. Danna, E. Rothberg, C.L. Pape, Exploring relaxation induced neighborhoods to improve MIP solutions. Math. Program. **102**, 71–90 (2005)
96. G.L. Nemhauser, A.G.H. Rinnooy Kan, M.J. Todd, *Optimization*, vol. 1 (North Holland, Amsterdam, 1989)
97. D. Bertsekas, *Network Optimization: Continuous and Discrete Models* (Athena Scientific, Belmont, 1998)
98. N. Shah, S. Kumar, F. Bastani, I.L. Yen, A space-time network optimization model for traffic coordination and its evaluation, in *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing* (SUTC 2008), Taichung (2008), pp. 177–184
99. South Staffs Water, Network Optimisation and Energy Management Business Strategy (2013)
100. C. Sun, L. Cheng, T. Xu, Range of user-equilibrium route flow with applications. Procedia. Soc. Behav. Sci. **138**, 86–96 (2014)
101. G.L. Nemhauser, L.A. Wolsey, *Integer and Combinatorial Optimization* (Wiley, Hoboken, 1999)
102. K. G. Murty, *Network Programming* (Prentice Hall, Upper Saddle River, 1992)