Appendix A
A General Formulation of the Pseudospectral Method

The optimal train trajectory planning problem of Chap. 3 can be formulated as a multiple phase optimal control problem and then can be solved by the pseudospectral method. Below we describe a general optimal control problem with multiple phases and the solution procedure of the general optimal control problem using the pseudospectral method. This explanation is based on [1–5].

A.1 The Multiple-Phase Optimal Control Problem

The general optimal control problem with N_p phases is formulated as follows [2].

The objective function is

$$J = \sum_{i=1}^{N_\mathrm{p}} \left(\varphi^{(i)}\left(x^{(i)}(t_\mathrm{f}^{(i)}), p^{(i)}, t_\mathrm{f}^{(i)}\right) + \int_{t_0^{(i)}}^{t_\mathrm{f}^{(i)}} L^{(i)}\left(x^{(i)}(t), u^{(i)}(t), p^{(i)}, t\right)\mathrm{d}t \right), \quad \text{(A.1)}$$

where $[t_0^{(i)}, t_\mathrm{f}^{(i)}]$ is the time interval for the ith phase, $u^{(i)}(\cdot)$ and $x^{(i)}(\cdot)$ are the control trajectories and state trajectories, $p^{(i)}$ are the static parameters, for $i = 1, 2, \ldots, N_\mathrm{p}$.

The objective function (A.1) is subject to the differential constraints

$$\dot{x}^{(i)}(t) = f^{(i)}\left(x^{(i)}(t), u^{(i)}(t), p^{(i)}, t\right), \quad t \in [t_0^{(i)}, t_\mathrm{f}^{(i)}], \quad \text{(A.2)}$$

the path constraints

$$h_\mathrm{L}^{(i)} \le h^{(i)}\left(x^{(i)}(t), u^{(i)}(t), p^{(i)}, t\right) \le h_\mathrm{U}^{(i)}, \quad t \in [t_0^{(i)}, t_\mathrm{f}^{(i)}], \quad \text{(A.3)}$$

Y. Wang et al., *Optimal Trajectory Planning and Train Scheduling for Urban Rail Transit Systems*, Advances in Industrial Control,
DOI 10.1007/978-3-319-30889-0

the event constraints

$$e_{\mathrm{L}}^{(i)} \le e^{(i)}\left(x^{(i)}(t_0^{(i)}), u^{(i)}(t_0^{(i)}), x^{(i)}(t_{\mathrm{f}}^{(i)}), u^{(i)}(t_{\mathrm{f}}^{(i)}), p^{(i)}, t_0^{(i)}, t_{\mathrm{f}}^{(i)}\right) \le e_{\mathrm{U}}^{(i)}, \quad t \in [t_0^{(i)}, t_{\mathrm{f}}^{(i)}], \tag{A.4}$$

the linkage constraints

$$\begin{aligned}\Psi_{\mathrm{L}} \le \Psi\big(&x^{(1)}(t_0^{(1)}), u^{(1)}(t_0^{(1)}), x^{(1)}(t_{\mathrm{f}}^{(1)}), u^{(1)}(t_{\mathrm{f}}^{(1)}), p^{(1)}, t_0^{(1)}, t_{\mathrm{f}}^{(1)},\\ &x^{(2)}(t_0^{(2)}), u^{(2)}(t_0^{(2)}), x^{(2)}(t_{\mathrm{f}}^{(2)}), u^{(2)}(t_{\mathrm{f}}^{(2)}), p^{(2)}, t_0^{(2)}, t_{\mathrm{f}}^{(2)},\\ &\cdots\\ &x^{(N_{\mathrm{p}})}(t_0^{(N_{\mathrm{p}})}), u^{(N_{\mathrm{p}})}(t_0^{(N_{\mathrm{p}})}), x^{(N_{\mathrm{p}})}(t_{\mathrm{f}}^{(N_{\mathrm{p}})}), u^{(N_{\mathrm{p}})}(t_{\mathrm{f}}^{(N_{\mathrm{p}})}), p^{(N_{\mathrm{p}})}, t_0^{(N_{\mathrm{p}})}, t_{\mathrm{f}}^{(N_{\mathrm{p}})}\big) \le \Psi_{\mathrm{U}},\end{aligned} \tag{A.5}$$

the bound constraints

$$\begin{aligned}u_{\mathrm{L}}^{(i)} \le u^{(i)}(t) \le u_{\mathrm{U}}^{(i)}, \quad x_{\mathrm{L}}^{(i)} \le x^{(i)}(t) \le x_{\mathrm{U}}^{(i)}, \quad t \in [t_0^{(i)}, t_{\mathrm{f}}^{(i)}],\\ p_{\mathrm{L}}^{(i)} \le p^{(i)} \le p_{\mathrm{U}}^{(i)}, \quad \underline{t}_0^{(i)} \le t_0^{(i)} \le \bar{t}_0^{(i)}, \quad \underline{t}_{\mathrm{f}}^{(i)} \le t_{\mathrm{f}}^{(i)} \le \bar{t}_{\mathrm{f}}^{(i)},\end{aligned} \tag{A.6}$$

and the time constraints

$$t_{\mathrm{f}}^{(i)} - t_0^{(i)} \ge 0. \tag{A.7}$$

A.2 The Solution Process of the Optimal Control Problem

Let $i \in \{1, 2, \ldots, N_{\mathrm{p}}\}$ be a particular phase of the optimal control problem (A.1)–(A.7) and let $(\cdot)^{(i)}$ denote information for the ith phase. The ith phase of the optimal control problem can be transformed from the interval $t \in [t_0^{(i)}, t_{\mathrm{f}}^{(i)}]$ to the interval $\tau \in [-1, 1]$ for $i = 1, 2, \ldots, N_{\mathrm{p}}$ by introducing the following transformation [2]:

$$\tau = \frac{2}{t_{\mathrm{f}}^{(i)} - t_0^{(i)}} t - \frac{t_{\mathrm{f}}^{(i)} + t_0^{(i)}}{t_{\mathrm{f}}^{(i)} - t_0^{(i)}}. \tag{A.8}$$

Now we approximate the state and control functions using Legendre pseudospectral approximation. The state $x_k^{(i)}(\tau)$, $\tau \in [-1, 1]$ is approximated by the N_ith order Lagrange polynomial $x_k^{N_i,(i)}(\tau)$ based on interpolation at the Legendre-Gauss-Lobatto (LGL) points [4]:

$$x_k^{(i)}(\tau) \approx x_k^{N_i,(i)}(\tau) = \sum_{n=0}^{N_i} \bar{x}_k^{N_i,(i)}(\tau_n)\phi_n^{(i)}(\tau), \tag{A.9}$$

where $\bar{x}_k^{N_i,(i)}(\tau_n)$ is a discrete approximation of the LGL point τ_n and the Lagrange basis polynomials $\phi_n^{(i)}(\tau)$ for $n = 0, 1, \cdots, N_i$ are defined as

$$\phi_n^{(i)}(\tau) = \prod_{m=0, m\neq n}^{N_i} \frac{\tau - \tau_m^{(i)}}{\tau_n^{(i)} - \tau_m^{(i)}}, \tag{A.10}$$

and $\tau_n^{(i)}$ for $n = 0, 1, \ldots, N_i$ are the LGL points, which are defined as $\tau_0^{(i)} = -1$, $\tau_{N_i}^{(i)} = 1$, and τ_n for $n = 1, 2, \ldots, N_i - 1$ being the subsequent roots of the derivative of the Legendre polynomial

$$L_{N_i}(\tau) = \frac{1}{2^{N_i} N_i!} \frac{\mathrm{d}^{N_i}}{\mathrm{d}\tau^{N_i}} (\tau^2 - 1)^{N_i},$$

in the interval $[-1, 1]$. The control $u^{(i)}(\tau)$ can be approximated in a similar way. The derivative of $x_k^{N_i,(i)}(\tau)$ at the LGL points τ_n can be obtained by differentiating (A.9), which can be expressed as a matrix multiplication as follows:

$$\dot{x}_k^{(i)}(\tau_n) \approx \dot{x}_k^{N_i,(i)}(\tau_n) = \sum_{j=0}^{N_i} \tilde{D}_{nj}^{(i)} \bar{x}_k^{N_i,(i)}(\tau_j), \tag{A.11}$$

where $\tilde{D}^{(i)}$ is the $(N_i + 1) \times (N_i + 1)$ differential approximation matrix [6] given by

$$\tilde{D}_{nj}^{(i)} = \begin{cases} \frac{\phi_{N_i}^{(i)}(\tau_n)}{\phi_{N_i}^{(i)}(\tau_j)} \frac{1}{\tau_n - \tau_j}, & \text{if } n \neq j, \\ -N_i(N_i + 1)/4, & \text{if } n = j = 0, \\ N_i(N_i + 1)/4, & \text{if } n = j = N_i, \\ 0, & \text{otherwise.} \end{cases} \tag{A.12}$$

The differential constraints can be recast into algebraic constraints via the differential approximation matrix. In addition, the path constraints (A.3) can be discretized at the LGL points. Note that the dynamic constraints and path constraints are only considered at the LGL points, which means both the dynamic and path constraints might be violated in between the LGL points [7]. The objective function (A.1) can be approximated using the LGL points as

$$J = \sum_{i=1}^{N_p} \left(\varphi^{(i)}\left(\bar{x}^{N_i,(i)}(-1), \bar{x}^{N_i,(i)}(1), p^{(i)}, t_0^{(i)}, t_f^{(i)} \right) + \frac{t_f^{(i)} - t_0^{(i)}}{2} \sum_{n=0}^{N_i} L^{(i)}\left(\bar{x}^{N_i,(i)}(\tau_n), \bar{u}^{N_i,(i)}(\tau_n), p^{(i)}, \tau_n \right) \omega_n \right), \tag{A.13}$$

where ω_n are weights given by

$$\omega_n = \frac{2}{N(N+1)} \frac{1}{(L_{N_i}(\tau_n))^2}, \quad \text{for } n = 0, 1, \ldots, N_i. \tag{A.14}$$

If we include all the decision variables in vector y, the optimal control problem can then be expressed as a nonlinear programming problem:

$$\min_y J(y), \tag{A.15}$$

subject to

$$\begin{aligned} G_{\mathrm{L}} \leq G(y) \leq G_{\mathrm{U}} \\ y_{\mathrm{L}} \leq y \leq y_{\mathrm{U}}. \end{aligned} \tag{A.16}$$

By defining $G(y)$, G_{L}, G_{U}, y_{L}, and y_{U} properly, we can write all constraints in the form (A.16).

There exist several commercial and free packages that implement the pseudospectral method: SOCS [8] and DIRCOL [9] are Fortran-based proprietary packages, while PROPT [7] and DIDO [10] are commercial software packages with MATLAB® interface. A MATLAB-based open source tool that uses the Gauss pseudospectral method is GPOPS [11]. PSOPT is an open source optimal control package written in C++, including Legendre and Chebyshev pseudospectral discretizations [1]. These software packages start with a general optimal problem formulated as (A.1)–(A.7), then transform this problem into a nonlinear programing problem with objective function (A.15) and constraints (A.16), and finally solve it possibly using an NLP solver, such as SNOPT [12].

Appendix B
Background—Optimization

In this appendix, background material on optimization (or mathematical programming) is presented. In Sect. B.1 a brief introduction of optimization is presented. Solution approaches for nonlinear programming problems are given in Sect. B.2. In addition, solution approaches for mixed-integer programming problems are introduced shortly in Sect. B.3.

B.1 Introduction of Optimization

Many real-world optimization problems from mechanical engineering, control engineering, economics, etc. can be modeled using the following general form [13]:

$$\begin{aligned} &\min_{x} f(x) \\ &\text{s.t. } h(x) = 0 \text{ and } g(x) \leq 0, \\ &x = \left[x_{\text{in}}^T \; x_{\text{re}}^T\right]^T, \; x_{\text{in}} \in \mathbb{Z}^{n_{\text{in}}}, \; x_{\text{re}} \in \mathbb{R}^{n_{\text{re}}}. \end{aligned} \tag{B.1}$$

where $\mathbb{Z}$ and $\mathbb{R}$ are the sets of integers and real numbers respectively, $f(\cdot)$ is the objective function of the optimization problem, x is the parameter vector that can be used to minimize the objective function, $h(\cdot) = 0$ and $g(\cdot) \leq 0$ are the equality and inequality constraints that restrict the solution to a certain subset of the parameter space. Note that there is no fundamental difference between maximization and minimization; so we only consider minimization here. For an optimization problem, the objective function $f(\cdot)$ and the parameter vector x are always present. However, the elements $g(\cdot)$ and $h(\cdot)$ can be omitted if no constraints are imposed on the values of the parameters. The unconstrained optimization problem and constrained problem can be distinguished based on the presence or absence of the constraints. For a

Y. Wang et al., *Optimal Trajectory Planning and Train Scheduling for Urban Rail Transit Systems*, Advances in Industrial Control,
DOI 10.1007/978-3-319-30889-0

constrained problem, if a point x satisfy the constraints, i.e., $h(x) = 0$ and $g(x) \leq 0$, then the point x is called feasible; otherwise, the point x is called infeasible.

Furthermore, the optimization problems can also be distinguished based on the parameter vector x, objective function $f(\cdot)$, and constraints $h(\cdot)$ and $g(\cdot)$. As given in (B.1), the parameters in x could be integer variables and/or real-valued variables. If the optimization problem is a real-valued optimization problem, then x_{in} in (B.1) does not exist. For a real-valued optimization problem, we have the following three cases:

- Linear programming problem: $f(\cdot)$ is a linear function and the constraints $g(\cdot)$ and $h(\cdot)$ are also linear[1] functions.
- Quadratic programming problem: $f(\cdot)$ is a quadratic function and the constraints $g(\cdot)$ and $h(\cdot)$ are linear functions.
- Nonlinear programming problem: $f(\cdot)$ is a nonlinear function (non quadratic if the constraints are linear) and/or the constraints $g(\cdot)$ and/or $h(\cdot)$ are nonlinear functions.

If x has integer values only, i.e., x_{re} does not exist, then the optimization problem is a pure integer optimization problem. Furthermore, if both x_{re} and x_{in} are present, then the optimization problem is a mixed-integer optimization problem.

If both the objective function and the feasible set are convex, then the minimum obtained is a global minimum; otherwise, there may be several local minima. A local minimum x^* is defined as a point for which exists some $\delta > 0$ so that for all x such that

$$\|x - x^*\| \leq \delta, \tag{B.2}$$

the expression

$$f(x^*) \leq f(x), \tag{B.3}$$

holds, i.e., on a certain region around x^* all the function values are larger than or equal to the function value of x^* [13].

The optimization problems considered in this book are mostly nonlinear programming and mixed integer nonlinear programming problems. We solve these optimization problems via sequential quadratic programming, iterative convex programming, genetic algorithms, pattern search, branch-and-bound, multi-start method, and bi-level optimization method. A brief introduction for these methods used is given next.

[1] A function f is affine if it is of the form $f(x) = a_1x_1 + a_2x_2 + \cdots + a_nx_n + b = a^Tx + b$ with $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. If $b = 0$ then we say that the function is linear. Note that many people also use the phrases "linear" even if $b \neq 0$. Here we also use "linear" in this appendix.

B.2 Solution Approaches for Nonlinear Programming Problems

B.2.1 Sequential Quadratic Programming

Sequential quadratic programming (SQP) is an iterative method for real-valued nonlinear programming problems that has been widely used to solve different nonlinear programming problems in practice [14]. The SQP method belongs the category of gradient-based nonlinear programming methods, which rely on gradient or Hessian information. If this information is not available, it can be approximated numerically. So a requirement for the SQP algorithm is that the objective function and the constraints of the nonlinear programming problem should be continuously differentiable [14]. In the SQP method, the nonlinear programming problem is recast as a sequence of quadratic programming problems, which can be solved easily and efficiently by an interior point method, an active set method, etc.

For the sake of simplicity of the exposition, we consider a nonlinear optimization problem in (B.1) but only with equality constraints [13, 15]. A Lagrange function is introduced as follows:

$$L(x, \lambda, \sigma) = f(x) + \lambda^T h(x), \tag{B.4}$$

where the parameter vector x is in $\mathbb{R}^n$, $\lambda \in \mathbb{R}^p$ are the Lagrange multipliers for the equality constraints. The Kuhn-Tucker necessary conditions for optimality can be stated as

$$F(x, \lambda) = \begin{bmatrix} \nabla f(x) + \nabla h(x)\lambda \\ h(x) \end{bmatrix} = 0, \tag{B.5}$$

where $\nabla h(x)$ is the Jacobian matrix of the equality constraints $h(x)$ and is an $n \times p$ matrix. So $F(x, \lambda)$ is a set of $n+p$ nonlinear equations with $n+p$ unknown variables, i.e., $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^p$. These nonlinear equations can be solved using Newton's method. The Jacobian matrix of $F(x, \lambda)$ can be written as

$$\nabla_{x,\lambda} F(x, \lambda) = \begin{bmatrix} \nabla^2_{x,x} L(x, \lambda) & \nabla h(x) \\ \nabla^T h(x) & 0 \end{bmatrix}. \tag{B.6}$$

According to the Newton's method, the solution of (B.5) can be found iteratively as

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \end{bmatrix}, \tag{B.7}$$

where Δx_k and $\Delta \lambda_k$ are obtained by solving

$$\begin{bmatrix} \nabla^2_{x,x} L(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla^T h(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) - \nabla h(x_k)\lambda_k \\ -h(x_k) \end{bmatrix}. \tag{B.8}$$

By substituting $\Delta\lambda_k = \lambda_{k+1} - \lambda_k$ given in (B.7) into (B.8), we have

$$\begin{bmatrix} \nabla^2_{xx} L(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla^T h(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -h(x_k) \end{bmatrix}. \tag{B.9}$$

The iterative process indicated by (B.7) and (B.9) can be continued until the convergence is achieved.

We consider the following quadratic programming problem:

$$\begin{aligned} \min_{\Delta x_k} &= \nabla^T f(x_k)\Delta x_k + \frac{1}{2}\Delta x_k^T \nabla^2_{x,x} L(x_k, \lambda_k)\Delta x_k \\ &\text{s.t. } h(x_k) + \nabla^T h(x_k)\Delta x = 0. \end{aligned} \tag{B.10}$$

The Lagrange function of the quadratic programming problem can be defined as

$$\tilde{L} = \nabla^T f(x_k)\Delta x_k + \frac{1}{2}\Delta x_k^T \nabla^2_{x,x} L(x_k, \lambda_k)\Delta x_k + \tilde{\lambda}_k^T (h(x_k) + \nabla^T h(x_k)), \tag{B.11}$$

where $\tilde{\lambda}_k$ is the Lagrange multiplier. The Kuhn-Tucker necessary conditions can be written as

$$\nabla f(x_k) + \nabla^2_{x,x} L(x_k, \lambda_k)\Delta x_k + \nabla h(x_k)\tilde{\lambda}_k = 0, \tag{B.12}$$

$$h(x_k) + \nabla^T h(x_k)\Delta x_k = 0. \tag{B.13}$$

It is noted that (B.12) and (B.13) can be identified to be the same as (B.9). So the original nonlinear problem of (B.1) but only with equality constraints can be solved in an iterative way by solving the quadratic programming problem defined by (B.10).

The SQP procedure for the equality-constrained nonlinear programing can be extended to a general nonlinear programming problem with both equality and inequality constraints [13, 15]. Note that the Hessian of the Lagrangian $\nabla^2_{x,x} L(x_k, \lambda_k)$ is made up of the second derivatives of the objective function and constraints, which may be time-consuming. So it is useful to consider replacing the Hessian $\nabla^2_{x,x} L(x_k, \lambda_k)$ by a quasi-newton approximation, e.g., the Broyden-Fletcher-Goldfarb -Schanno approximation [15]. SQP methods have been implemented in many commercial and free software packages, such as SNOPT [12, 16], NPSOL [17], and MATLAB [18].

B.2.2 Pattern Search

Pattern search is a direct-search method [19], which does not require the gradient or Hessian of the objective function and of the constraints to solve the optimization problem. Therefore, pattern search can be applied to functions that are not

continuous nor differentiable. Pattern search methods [15] choose a certain set of search directions at each iteration point and evaluate the objective function at a given step length along each of these directions. If a point with a significantly lower function value is found, then this point is adopted as the new iteration point, where the set of directions may be changed and the step length may grow or shrink as explained next.

We define x_k to be the solution at iterative step k, D_k to be the set of possible search directions, and ℓ_k to be the step size. Then the candidates for the current iteration are $x_k + \ell_k d_k$ for all $d_k \in D_k$. In addition, a function $\rho(\cdot)$ of the step length is introduced to evaluate the decrease of the objective function. The iterative procedure of pattern search is given as follows [15]:

- Choose initial point x_0, initial step length ℓ_0, and initial direction set D_0. Note that the initial step length ℓ_0 should be larger than the convergence tolerance step length ℓ_{tol}.
- Evaluate the candidates $x_k + \ell_k d_k$ for all $d_k \in D_k$ for each iteration k.
  - If $f(x_k + \ell_k d_k) \leq f(x_k) - \rho(\ell_k)$ for some $d_k \in D_k$, then set $x_{k+1} = x_k + \ell_k d_k$ for such d_k. In addition, also set $\ell_{k+1} = \phi_k \ell_k$ with $\phi_k \geq 1$ to increase the step length.
  - If the inequality $f(x_k + \ell_k d_k) \leq f(x_k) - \rho(\ell_k)$ cannot be satisfied for any of the candidates, then we set $x_{k+1} = x_k$ and reduce the step length by $\ell_{k+1} = \theta_k \ell_k$ with $0 < \theta_k < 1$.
- Terminate the iterative procedure if the step length ℓ_k is less than the given convergence tolerance ℓ_{tol}.

Note that the pattern search iterative procedure does not require us to choose the point $x_k + \ell_k d_k$ with the smallest objective value, so the function of evaluations could be reduced by not evaluating the function values for all the candidates, but performing the evaluations one at a time and accepting the first candidate point that satisfies the sufficient decrease condition.

Several pattern search algorithms have been implemented in the Global Optimization Toolbox of the MATLAB [20], such as the generalized pattern search algorithm, the generating set search algorithm, and the mesh adaptive search algorithms.

B.2.3 Genetic Algorithm

Genetic algorithms are based on the concepts of biological evolution, which involves the principles of natural genetics and nature selection [21, 22]. The basic elements of nature genetics are reproduction, crossover, and mutation, which are used in genetic search procedure. So genetic algorithms do not depend on the gradient and/or Hessian information, but only based on the function evaluations. Many practical optimization problems, such as non-convex nonlinear programming problems, mixed integer

nonlinear programming problems, can be solved by genetic algorithms and in some cases the global optimum solution can be obtained with a high probability [13].

In genetic algorithms, a set of possible candidate solutions, i.e. a population, is used for starting the procedure, where each candidate is represented by a string of binary variables corresponding to the chromosomes in nature genetics. For continuous variables, the desired resolution can be achieved by varying the string length. Since a whole population of possible solutions is considered, genetic algorithms can escape from local minima. Each candidate in the initial population is evaluated to obtain its fitness value. Then three genetic operators, i.e., selection, crossover, and mutation, are operated to produce a new population of solutions. The new population is then evaluated to get the fitness values and to obtain the convergence of the iterative process. If the convergence criterion or the maximum number of generations is reached, then the iterative procedure terminates; otherwise, the population is iteratively operated by the three operators and the resulting new population is evaluated.

By adopting the survival-of-the-fittest principle, genetic algorithms try to maximize the fitness function $F(\cdot)$. The fitness function can be constructed by the objective function $f(\cdot)$, the equality constraints $h(x) = 0$, and the inequality constraints $g(x) \leq 0$ of the minimization problem given in (B.1). A possible construction of the fitness function $F(\cdot)$ could be [13]

$$F(x) = \frac{1}{1 + \phi(x)}, \tag{B.14}$$

where

$$\phi(x) = f(x) + \alpha h(x) h^T(x) + \beta \hat{g}(x) \hat{g}^T(x), \tag{B.15}$$

with

$$\hat{g}_i(x) = \begin{cases} g_i(x) & \text{if } g_i(x) > 0, \\ 0 & \text{if } g_i(x) \leq 0, \end{cases} \quad \text{for } i = 1, 2, \ldots, m \tag{B.16}$$

and where α and β are the penalty parameters associated with the constraints $h(x) = 0$ and $g(x) = 0$, and m is the total number of inequality constraints.

Furthermore, the three genetic operators are explained in detail as follows [21, 22]:

- Selection. The selection operator is used to pick the good strings from the current population and to put their multiple copies in the mating pool based on a probabilistic procedure. In the selection process, the strings with a higher fitness value will be selected more frequently and get more copies in the mating pool because of the larger probability and vice versa. Note that no new strings are produced in the selection phase, but only the existing candidates are copied in the mating pool.
- Crossover. After the selection process, the crossover operator is applied to generate new strings, where some parts of the strings are exchanged between two randomly picked individual strings in the mating pool. The two strings participating in the crossover operator are called parents strings and the strings created by crossover

process are child strings. The child strings generated via a random crossover that could be better or worse than their parents. If they are better than their parents, the average fitness of the new population will be improved faster. However, if they are worse than their parents, they would be disregarded in the next selection process.

- Mutation. The mutation operator is applied to the new population with a small mutation probability, where one or more binary variables of a string changes from 1 to 0 and vice versa. Through the mutation process, a string in the neighborhood of the current string is generated to accomplish a local search around the current solution and to maintain diversity of the population.

The computational procedure of genetic algorithm includes the following steps:

1. **Initialization**: Choose a proper string length and create an initial population. Set appropriate values for the crossover probability, mutation probability, etc.
2. **Iteration**: Evaluate the fitness values for the candidate solutions in the population and carry out the selection process, crossover process, and mutation process to create a new population.
3. **Stopping criterion**: The optimization process stops if the maximum number of generations is reached or the convergence criterion is satisfied.

B.2.4 Iterative Convex Programming

Inspired by the iterative quadratic programming approach proposed in [23], the iterative convex programming (ICP) approach is proposed to solve nonlinear non-convex optimization problems with real-valued variables approximately [24]. In the ICP approach, the variables x of the nonlinear non-convex problem given in (B.1) are split into two vectors x_1 and x_2, so the optimization problem (B.1) can be written as:

$$\begin{aligned} &\min_{x_1, x_2} f(x_1, x_2) \\ \text{s.t. } &h(x_1, x_2) = 0, \\ &g(x_1, x_2) \leq 0, \end{aligned} \tag{B.17}$$

where $x = \left[x_1^T \; x_2^T\right]^T$. The optimization problem in (B.17) can be split in to the following two subproblems:

$$\begin{aligned} &\min_{x_1} f(x_1, \hat{x}_2) \\ \text{s.t. } &h(x_1, \hat{x}_2) = 0, \\ &g(x_1, \hat{x}_2) \leq 0, \end{aligned} \tag{B.18}$$

and

$$\begin{aligned} &\min_{x_2} f(\hat{x}_1, x_2) \\ &\text{s.t. } h(\hat{x}_1, x_2) = 0, \\ &\quad g(\hat{x}_1, x_2) \leq 0, \end{aligned} \tag{B.19}$$

where $\hat{x}_1$ and $\hat{x}_2$ are not variables any more, but are fixed values in the optimization models. If both (B.18) and (B.19) are convex programming problems, then the ICP approach can be applied to solve this nonlinear non-convex problem. A convex programming problem can be solved for the global optimum by use of efficient algorithms such as interior point methods [15]. However, the simplifications for the ICP approach mean that the global optimum of the convex problem is not necessarily the global minimum of the original nonlinear non-convex problem. Therefore, an iterative process is adopted to reduce the errors introduced by the simplifications. The iterative procedure can be described as the following steps:

1. Obtain the initial estimates $\hat{x_2}(0)$ of x_2 for the nonlinear non-convex problem.
2. Based on the estimates $\hat{x}_2(k)$, perform the convex programming for (B.18) to find the estimates $\hat{x}_1(k)$.
3. Based on the new estimate $\hat{x}_1(k)$ and perform the convex programming for (B.19) to find estimates $\hat{x}_2(k+1)$.
4. Repeat step 2 and 3 until the maximum number of iterations is reached or the convergence of optimization variables is achieved.

Remark B.1 When the programming problem is nonlinear and non-convex, there could exist multiple local minima. So the nonlinear programming approaches introduced before should be combined with the multi-start methods to obtain a solution close to a globally optimal solution. Multi-start methods generally increase the computation time required to solve an optimization problem significantly; however, multi-start methods can typically be executed in a highly parallel manner. In particular, when a straightforward multi-start method is chosen that relies on randomly generated initial solutions, then each optimization problem involved in the multi-start method can be solved on an independent processor.

B.3 Solution Approaches for Mixed-Integer Optimization Problem

Mixed-integer programming problems have both real-valued and integer-valued parameters as given in (B.1). Sections B.3.1 and B.3.2 introduce some solution approaches for mixed integer linear programming problems and mixed integer nonlinear programming problems respectively.

B.3.1 Mixed Integer Linear Programming

There are several advanced algorithms available for solving mixed integer linear programming (MILP) problems, such as the cutting plane algorithm [25] and the branch-and-bound method [26]. In particular, the branch-and-bound method has been applied widely for solving MILP problems because of its effectiveness. Hence, a detailed description of the branch-and-bound method is introduced next.

For the branch-and-bound method, a linear programming relaxation problem of an MILP problem is first solved without considering the integer constraints. If the values of the integer variables happen to be integers, then the MILP problem has been solved successfully: the optimal solution of the linear programming relaxation problem is the optimal solution of the original MILP problem. Otherwise, the optimal solution for the MILP problem can be obtained as follows. Let x_i be an integer variable having a real value in the solution of the linear programming relaxation problem. Then we can find an integer $\hat{x}_i$ that satisfies

$$\hat{x}_i \leq x_i \leq \hat{x}_i + 1. \tag{B.20}$$

With (B.20), the linear programming relaxation problem can be branched on the variable x_i and two subproblems are created as follows [13]:

1. The linear programming relaxation problem with constraint $x_i \leq \hat{x}_i$.
2. The linear programming relaxation problem with constraint $x_i \geq \hat{x}_i + 1$.

Note that the branching process can keep all the feasible integer solutions for MILP problems. Each of the two subproblems can then be solved as a real-valued linear programming problem and the solution of these two subproblems can be branched further on other integer variables. This branching process continues until all $x_{\text{in}} \in \mathbb{Z}^{n_{\text{in}}}$. When such a feasible solution is found, the corresponding objective function value is set as an upper bound for the objective function. Then the branching nodes of which the objective function values are larger than the upper bound can be cut or eliminated because further branching on these nodes will result in bigger objective function values. In addition, the upper bound of the objective function is updated when feasible solution with a smaller objective function value is found. Eventually, all branches except the optimal solution are eliminated from consideration and the branch-and-bound procedure is terminated.

The branch-and-bound procedure can be efficient for eliminating non-optimal candidate solutions because several branches could be cut (i.e., the subproblems on these branch should not be solved anymore) when a upper bound of the objective function is obtained. A branch can be cut if one of the following conditions is satisfied [13]:

- The subproblem does not have a feasible solution.
- The optimal solution of the subproblem is feasible for the original MILP problem.
- The objective function value of the subproblem is larger than the current upper bound.

The MILP problem can be solved by several existing commercial and free solvers [27–29], such as CPLEX, GUROBI, GLPK, LINDO, LINGO, SCIP.

B.3.2 Mixed Integer Nonlinear Programming

The extension of the branch-and-bound algorithm described in Sect. B.3.1 for solving mixed integer nonlinear programming (MINLP) problems is straightforward, where nonlinear programming subproblems have to be solved instead of linear programming subproblems [30]. If the relaxed real-valued nonlinear optimization problem is convex, the MINLP algorithm will converge to the global optimum [30].

The MINLP problems as well as nonlinear programming problems can be solved via a MILP approach, where the nonlinear terms can be approximated by piecewise affine functions. Then the original nonlinear programming problem is transformed into an MILP problem by applying transformation properties [31]. It is noted that the number of binary variables introduced by the nonlinear terms is related to the number of piecewise affine subfunctions. So attention should be paid to keeping the number of binary variables small when reformulating the MINLP and nonlinear programming problems into MILP problems; because the MILP problem has been proven to be NP-hard and increasing the number of variables will increase the computational complexity. However, fewer binary variables would lead to fewer affine pieces in the approximation process, which may cause larger discrepancy with the original nonlinear functions. Therefore, it is important to find a good trade-off between accuracy of approximation and computational complexity. Moreover, the optimal solution of the MILP problems may differ from that of the MINLP problems because of the error caused by the approximations.

In addition, a bi-level optimization method consisting of two levels of optimization could be used for solving the MINLP problems, where the high level optimizes the integer variables and the low level solves real-valued nonlinear programming problems [32]. The high level optimization problem can be formulated as

$$\begin{aligned} &\min_{x_{\text{in}}} f(x_{\text{in}}) \\ &\text{s.t. } \bar{h}(x_{\text{in}}) = 0 \text{ and } \bar{g}(x_{\text{in}}) \leq 0, \\ &x_{\text{in}} \in \mathbb{Z}^{n_{\text{in}}}, \end{aligned} \tag{B.21}$$

where $\bar{h}$ and $\bar{g}$ are the constraints which are only related with the integer variables, and the objection function value $f(x_{\text{in}})$ are calculated by the low level optimization. The optimization problem in the low level can be written as

$$\begin{aligned} &f(x_{\text{in}}) = \min_{x_{\text{re}}} f(x_{\text{in}}, x_{\text{re}}) \\ &\text{s.t. } \tilde{h}(x_{\text{in}}, x_{\text{re}}) = 0 \text{ and } \tilde{g}(x_{\text{in}}, x_{\text{re}}) \leq 0, \\ &x_{\text{re}} \in \mathbb{R}^{n_{\text{re}}}, \end{aligned} \tag{B.22}$$

where $\tilde{h}$ and $\tilde{g}$ are the constraints which are related with the real-valued variables and the integer variables. A branch-and-bound method, exhaustive method, etc. can be applied to the high-level integer programming, while a (multi-start) SQP, pattern search, etc. can be employed by the low-level nonlinear programming.

The MINLP problems can be solved by several existing commercial and free solvers [29], such as MINLP BB, SCIP, NOMAD, and BONMIN.

References

1. Becerra V (2010) Solving complex optimal control problems at no cost with psopt. In: Proceedings of IEEE multi-conference on systems and control, Yokohama, Japan, pp 1391–1396
2. Ross I, Fahroo F (2004) Pseudospectral knotting methods for solving optimal control problems. J Guidance, Control, Dyn 27:397–405
3. Gong Q, Ross I, Kang W, Fahroo F (2008) Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control. Comput Optim Appl 41:307–335
4. Canuto C, Hussaini M, Quarteroni A, Zang T (1988) Spectral methods in fluid dynamics. Springer, New York
5. Elnagar G, Kazemi M, Razzaghi M (1995) The pseudospectral Legendre method for discretizing optimal control problems. IEEE Trans Autom Control 40:1793–1796
6. Gong Q, Kang W, Bedrossian N, Fahroo F, Sekhavat P, Bollino K (2007) Pseudospectral optimal control for military and industrial applications. In: Proceedings of the 46th IEEE conference on decision and control, LA, USA, New Orleans, pp 4128–4142
7. Rutquist P, Edvall M (2008) PROPT: MATLAB optimal control software. Tomlab Optimization Inc., Pullman
8. Betts J (2002) A direct approach to solving optimal control problems. Comput Sci Eng 1:73–75
9. Stryk O (2000) User's guide for DIRCOL (version 2.1): a direct collocation method for the numerical solution of optimal control problems. Technical Report. Technical University of Munich. Munich, Germany
10. Ross I (2004) User's manual for DIDO: a MATLAB application package for solving optimal control problems. Tomlab Optimization Inc., Pullman
11. Rao A, Benson D, Darby C, Patterson M, Francolin C, Sanders I (2010) Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the Gauss pseudospectral method. ACM Trans Math Softw 37:22:1–22:39
12. Gill P, Murray W, Saunders M (2002) SNOPT: an SQP algorithm for large-scale constrained optimization. Soc Ind Appl Math J Optim 12:979–1006
13. Rao S (2009) Engineering optimization: theory and practice. Wiley, Hoboken
14. Boggs P, Tolle J (1995) Sequential quadratic programming. Acta Numerica 4:1–51
15. Nocedal J, Wright S (2006) Numerical optimization. Springer Science + Business Media, New York
16. Gill P, Murray W, Saunders M (2007) User's guide for SNOPT version 7: software for large scale nonlinear programming. Technical report. University of California, San Diego
17. Gill P, Murray W, Saunders M (1998) User's guide for NPSOL 5.0: a Fortran package for nonlinear programming. Technical report. University of California. San Diego
18. The Mathworks Inc. (1999) Optimization toolbox for use with Matlab: user's guide. The Mathworks Inc., Natick, MA, USA
19. Lewis R, Torczon V, Trosset M (2000) Direct search methods: then and now. J Comput Appl Math 124:191–207
20. The Mathworks Inc. (2004) Genetic algorithm and direct search toolbox for use with MATLAB: user's guide. The Mathworks Inc., Natick, MA, USA

21. Goldberg D (1989) Genetic algorithm in search, optimization and machine learning. Addison-Wesley, Reading
22. Vose M (1999) The simple genetic algorithm: foundations and theory. The MIT Press, Cambridge
23. Koehler L, Kraus W, Camponogara E (2011) Iterative quadratic optimiztion for the bus holding control problem. IEEE Trans Intell Transp Syst 12:1568–1575
24. Wang Y, De Schutter B, van den Boom T, Ning B, Tang T (2014) Efficient real-time train scheduling for urban rail transit systems using iterative convex programming. Technical report. Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands. Submitted to a journal
25. Owen JH, Mehrotra S (2001) A disjunctive cutting plane procedure for general mixed-integer linear programs. Math Program 89:437–448
26. Dakin R (1965) A tree-search algorithm for mixed integer programming problems. Comput J 8:250–255
27. Linderoth J, Ralphs T (2005) Noncommercial software for mixed-integer linear programming. In: Karlof J (ed) Integer programming: theory and practice. Operations research series. CRC Press, Boca Raton, pp 253–303
28. Atamturk A, Savelsbergh M (2005) Integer-programming software systems. Ann Oper Res 140:67–124
29. Bussieck M, Vigerske S (2010) MINLP solver software. Wiley Encyclopedia of operations research and management science
30. Quesada I, Grossmann I (1992) An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Comput Chem Eng 16:937–947
31. Williams H (1999) Model building in mathematical programming, 4th edn. Wiley, Chichester
32. Wang Y, De Schutter B, van den Boom T, Ning B, Tang T (2014) Efficient bi-level approach for urban rail transit operation with stop-skipping. IEEE Trans Intell Transp Syst 15:2658–2670

Index

Y. Wang et al., *Optimal Trajectory Planning and Train Scheduling for Urban Rail Transit Systems*, Advances in Industrial Control,
DOI 10.1007/978-3-319-30889-0