THE UNIVERSITY
of EDINBURGH

Pre-trained solution methods for unit commitment

Nagisa Sugishita

Doctor of Philosophy
University of Edinburgh
2022

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Nagisa Sugishita*)

Abstract

This thesis aims to improve the solution methods for the unit commitment problem, a short-term planning problem in the energy industry. In particular, we focus on Dantzig-Wolfe decomposition with a column generation procedure. Special emphasis is placed on approaches based on machine learning, which is of interest when one needs to solve the unit commitment problem repeatedly.

Firstly, an initialisation method of the column generation procedure based on a neural network is studied. After offline training, for each unit commitment problem, the method outputs dual values which can be used to warmstart the solution method, leading to a significant saving of computational time. The training is done efficiently by exploiting the decomposable structure of the problem.

Secondly, primal heuristics are discussed. Two novel primal heuristics are proposed: one based on the decomposition and another based on machine learning. Both of them fix a subset of the binary variable to reduce the problem size. The remaining variable is optimised quickly by an optimisation solver, which gives primal feasible solutions with small suboptimality in a short time.

Finally, the column generation procedure is extended to handle incremental generation of columns. Instead of generating columns for all the components (power plants in the unit commitment problem) in each iteration, our method generates a subset of them and update the dual variable using the partially updated restricted master problem. Convergence analysis of the method is given under various conditions as well as numerical experiments to show the performance of the method.

By combining the above enhancements, we obtain a fast solution method to solve the unit commitment problem to small tolerances down to 0.1%.

Lay Abstract

As our society becomes more and more reliant on electricity, the power system grows both in scale and in complexity. Due to its enormous cost, even a small improvement in efficiency leads to a striking amount of reduction when expressed in total numbers. The complex system requires highly sophisticated and reliable planning methodologies. Various optimisation models and their solution methods have been studied to aid its operation.

The unit commitment problem is an optimisation problem used to find a cost-effective operational schedule of power plants. The timing of switching on and off generators and the amount of power dispatch must be optimised simultaneously. The discrete nature of the decision renders the problem notoriously hard and it is often not practical to solve the problem using a general optimisation software. This thesis studies efficient solution methods for the unit commitment problem and examines its improvement using recent development in optimisation.

In particular, in this thesis special emphasis is placed on the applications of machine learning to accelerate the solution methods. Machine learning is a promising branch of the computer science which aims to extract unknown patterns from data. The techniques are especially useful when the unit commitment problem is to be solved repeatedly with the same problem structure but with slightly perturbed parameter values. When the unit commitment problem is used as a daily planning problem, the problem is solved in a daily basis. It is demonstrated that effective use of machine learning can reduce the computational time.

Acknowledgements

First and foremost I would like to thank my PhD supervisors, Prof. Ken McKinnon and Dr. Andreas Grothey. The work compiled in this thesis would not be possible without the productive and supportive environment provided by my excellent supervisors. I deeply appreciate their continuous support. I am especially grateful for your consistent, patient supervision since my master's program, Ken. Through your sagacious and ingenious discussion in the last six years I have learned a significant amount of expertise and acquired an attitude to enjoy intellectually challenging problems. Additionally I would like to thank Dr. Nicolò Mazzi, who often joined our meetings and gave me indispensable feedback. A large part of this work is inspired by the work done by Dr. Tim Schulze during his PhD. Dr. Tim Schulze kindly helped me to understand his work and provided me with insightful input.

I would like to thank the academic staff at Edinburgh. Especially Prof. Jacek Gondzio, Prof. Miguel Anjos, Dr. Julian Hall, Dr. Sergio García Quiles, Dr. Joerg Kalcsics and Dr. Burak Buke have taught me optimisation and operational research from very basic since I was a master's student. The time I have spent in such an inspirational group is a great asset in my life. Prof. Miguel Anjos and Prof. Antonio Frangioni agreed to be examiners of my viva and I am grateful for their generosity. I acknowledge funding obtained through the Principals' Career Development Scholarship and Edinburgh Global Research Scholarship scheme.

Last but not least, I would like to thank my great colleagues and friends: Nestor, Magda, Jonna, Santi, Ivona, Paula, Maria, Rodrigo, Bárbara, Malte, Monse, and many others. Thank you for all the support and good times.

Contents

Chapter 1

Introduction

1.1 Motivation

The unit commitment (UC) problem is a short-term planning problem. Its typical planning horizon is a single or few days. Given the set of available generators and demand over the planning horizon, the optimal operating schedules are to be computed. The timing of switching on and off generators and the amount of power dispatch must be optimised simultaneously. This gives rise to a large-scale combinatorial problem and due to its practical importance, it has been actively studied over the last few decades.

When the UC problem was developed, a typical electricity market was regulated and the entire system was managed collectively by a single operator [26]. The problem was solved by the operator to plan how to efficiently deliver electricity. Nowadays many electricity markets are deregulated and not only energy but also various ancillary services are traded in the markets. However, the UC problem is still of use in such a modern electricity market. The problem has been extended to model such a market and to determine which generators to provide energy and ancillary services [32].

This thesis focuses on the deterministic UC problem. That is, we assume that the demand is known or an accurate and reliable forecast is available. Popular approaches to the UC problem are decomposition-based, such as Lagrangian relaxation and Dantzig-Wolfe decomposition. This thesis examines practical issues of the

methods, taking recent advances in optimisation into account.

The significant advances in the theory and practice of optimisation can be seen in the performance improvement of general-purpose mixed-integer linear programming (MILP) solvers. Hobbs et al. [32] use CPLEX 3.0 and 6.5 to solve the same UC instance and observe that the solution time is reduced by approximately 95%. The solution methods tailored to the UC problem are still faster than general-purpose solvers applied to the UC problem. However, these improved general-purpose solver can be used to solve auxiliary problems within the tailored solution methods and this allows previously intractable problems to be solved.

There has been a growing interest in the use of machine learning within optimisation [5]. It is especially useful if an optimisation problem is solved repeatedly with perturbed parameters. This is often the case in practice when the UC problem is solved by an electricity generating company as a day-ahead planning problem. The UC problem is solved repeatedly with different demand forecasts while the characterisations of the generators, such as generation costs and ramping limits, remain the same.

It is of interest to improve the efficiency of the solution methods for the UC problems using such advances. We review previous studies and propose various enhancements. Each method is extensively tested with UC instances with between 200 and 1000 generators.

We note that in the literature a large number of nature-inspired heuristics are studied as direct, decomposition-free solution methods for the UC problem. For a recent survey covering such methods, see Saravanan et al. [60]. However, many of them have not been tested comprehensively yet and often the quality of solutions such as their suboptimality is not reported. Since the capability of such methods to deliver primal feasible solutions with small suboptimality (such as 0.1%) is not yet clear, we do not consider them further in this thesis.

1.2 Previous research

In the early days, heuristics, such as priority list [28], were dominant as solution methods for the UC problem. Such methods are capable of delivering a feasible

solution quickly with limited computational resource, but the suboptimality of the outputs is typically large. Later, dynamic programming gained in popularity [43] as an exact solution method. However, the computational time of dynamic programming grows unfavourably as the number of generators in the problem gets larger and solving a large-scale UC instance with the dynamic programming is impractical.

To solve the UC problem of practical size a decomposition-based method, such as Lagrangian relaxation, is commonly used. It is based on the observation that the UC problem has a limited number of complicating constraints (e.g. the load balance and reserve constraints) and the remaining constraints are operational constraints that only involve single generators. By relaxing the complicating constraints one can decompose the problem by generators. Evaluating the Lagrangian involves solving as many single-generator UC problems as there are generators. Such a single-generator UC problem is referred to as a subproblem. The value of the Lagrangian is a lower bound on the optimal objective value of the original UC problem. The dual problem is the optimisation problem to find the best lower bound. This is a convex but non-differentiable problem and an optimisation method which can handle non-differentiability is required.

Muckstadt and Koenig [47] combine Lagrangian relaxation and the branch and bound algorithm. Instead of the linear programming relaxation (LPR), they use Lagrangian relaxation in each node of the branch and bound tree to compute a lower bound. The dual problems are solved with a subgradient method. Compared with the LPR, Lagrangian relaxation gives tighter lower bounds. This improvement leads to a reduction of the number of nodes that needs to be explored. However, this is still a large number of nodes and a dual problem are to be solved at each node, so the overall computational time is still prohibitively large.

As Bertsekas et al. [6] point out, for an approach based on Lagrangian relaxation to be practical, branching should not be used at all. It is therefore necessary that Lagrangian relaxation has a small duality gap (the gap between the optimal objective value and the best dual lower bound) and that a primal feasible solution of sufficiently small suboptimality can be found with a suitable primal heuristic without resorting to branching. A number of numerical experiments have suggested that the duality gap for the UC problem tends to be small if the number of generators is large [6,

1]. Thus, the remaining issue is the approach used to find a good primal feasible solution. The behaviour of a primal heuristic depends on the property of the problem at hand and little theory is known in general. However, it is of critical importance to have a strong primal heuristic to make the approach practical.

Lauer et al. [40], Bertsekas et al. [6], and Zhuang and Galiana [69] propose two-stage methods in which the dual problem is solved first and then a primal solution is constructed from the dual solution. Lauer et al. [40] and Bertsekas et al. [6] solve the dual problem by approximating the problem with a twice-differentiable problem which is solved by a Newton-like method. The dual solution of the dual problem corresponds to a primal solution (i.e. generator schedules) but it does not satisfy the integrality condition. They obtain an integral primal solution by rounding the fractional values in the primal solution.

Zhuang and Galiana [69] use a subgradient method to optimise the dual objective function. After the optimal dual values are found, they construct a primal candidate solution using the solutions to the subproblems obtained in the last iteration of the subgradient method. First, they create a candidate primal solution from the subproblem solutions. Such a solution satisfies all the operational constraints of the generators, such as the minimum up/downtime constraints, as well as the integrality constraint. However, it does not necessarily satisfy the relaxed constraints (i.e. the load balance and reserve constraints). If the candidate solution is infeasible, they increase the dual values corresponding to the violated constraints and solve the subproblems with the update dual values again. Larger dual values encourage more generators to be committed on the time periods with constraint violation, leading to a primal candidate solution with a smaller constraint violation. They repeat this adjustment of the dual values until they find a primal feasible solution.

A major drawback of the above approaches is that a primal feasible solution becomes available only after the dual problem is solved to completion. Instead of waiting for the dual problem to be solved, Merlin and Sandrin [46] run primal heuristics in each iteration of a subgradient method applied to the dual problem. In each iteration of the subgradient method, a candidate primal solution is created from the subproblem solutions. When the candidate primal solution is infeasible, they modify the step direction to increase the dual values corresponding to the violated

constraints. This approach provides primal feasible solutions before the termination of the optimisation method for the dual problem. However, the modification of the subgradient method is likely to affect the optimisation of the dual problem and the resulting lower bound is likely to be suboptimal.

Most primal heuristics studied today fix infeasibility by modifying the subproblem solutions directly instead of manipulating the dual values. As in the above approaches such methods are run in each subgradient iteration. Guan et al. [29] and Barnhart et al. [2] apply local search to fix infeasibility of the candidate primal solution obtained from the subproblem solutions. Takriti et al. [63] combine subproblem solutions obtained over multiple iterations to find a primal feasible solution. These methods are studied more closely in Chapter 4.

Most of the papers described above use subgradient methods to solve the dual problem. The cutting-plane method is an alternative approach to solving the dual problem. The cutting-plane method for Lagrangian relaxation is the dual of Dantzig-Wolfe decomposition with a column generation procedure [16]. It suffers from instability and regularisation is often added to tackle the issue. The cutting-plane method is used to solve the UC problem by Redondo and Conejo [58] and Madrigal and Quintana [44]. The bundle method is another solution method that uses a cutting-plane model with a quadratic regularisation [41]. These methods also solve the subproblems in each iteration and the aforementioned primal heuristics can be used to find primal feasible solutions.

1.3 Objective

In this thesis we study Dantzig-Wolfe decomposition with the column generation procedure to solve the UC problem. Its implementation is examined and various enhancements are proposed. The aims and main contributions of this thesis are as follows.

- Initialisation methods to warmstart the column generation procedure are studied. Initialisation has had little attention in many of the previous works. However, our numerical experiments show that the initialisation of the column

generation procedure has a critical impact on the performance. We first analyse a standard initialisation method based on the LPR. This method provides a good starting point to warmstart the column generation procedure. However as we later observe in numerical experiments this initialisation method may take a significant amount time (more than 20% of the total computational time on average on 1000-generator instances). Given the substantial room for improvement, we propose a new initialisation method based on using a neural network. The neural network can be trained efficiently using the decomposable structure of the problem.

- Two novel primal heuristics are proposed: one integrated with decomposition and another based on machine learning. Some analysis on the first primal heuristic is provided to support its scalability. Our numerical experiments show that in our test instances our primal heuristics find a primal feasible solution of small suboptimality (e.g. 0.1%) in a short time.

- The column generation procedure is extended to handle incremental generation of columns. Instead of generating columns for all the components in each iteration, our method generates ones for a subset of them. Convergence analysis of the method is given under various conditions. Furthermore, the numerical experiments are provided to show that the proposed technique successfully accelerates the method.

- We demonstrate that by combining the above approaches, we obtain a faster solution method than any of them.

1.4 Structure

The remainder of this thesis is structured as follows. Chapter 2 describes the problem formulation and its solution methods, Lagrangian relaxation and Dantzig-Wolfe decomposition. In Chapter 3 initialisation methods for the algorithm are reviewed and a method based on a neural network is proposed. Chapter 4 discusses primal heuristics. Novel primal heuristics are proposed and their performances are compared.

Chapter 5 extends the column generation procedure to handle incremental updates. The convergence property of the method is studied and numerical experiments are given to show the behaviour of the new algorithm. In Chapter 6 the above techniques (initialisation, primal heuristics and incremental updates) are combined as a single method and its performance is studied. Finally, Chapter 7 summarises the results and draws conclusions.

Chapter 2

Problem Formulation and Solution Methods

2.1 Unit commitment

2.1.1 Full problem formulation

In this section we describe the formulation of the UC problem. The formulation of the UC is still actively studied and new formulations are proposed. For recent development see [38] In this thesis we closely follow one of the standard formulations in literature, referred to as the 3-binary variable formulation by Ostrowski et al. [55].

- **Load balance**: Generators have to meet all the demand in each time period (generation shedding at 0 cost is allowed).
- **Reserve**: To deal with contingencies, it is required to keep a sufficient amount of back up in each time period, which can be activated quickly.
- **Power output bounds**: Each generator's power output has to be within its limit.
- **Ramp rate bounds**: Generators can only change their outputs within the ramp rates.
- **Minimum up/downtime**: If switched on (off), each generator has to stay on (off) for a given minimum period.

The formulation of the model is as follows.

- Parameters
  - G: number of generators
  - T: number of time periods where decisions are taken
  - C_g^{nl}: no-load cost of generator g
  - C_g^{mr}: marginal cost of generator g
  - C_g^{up}: startup cost of generator g
  - $P_g^{\mathrm{max/\,min}}$: maximum/minimum generation limit of generator g
  - $P_g^{\mathrm{ru/rd}}$: operating ramp up/down limits of generator g
  - $P_g^{\mathrm{su/sd}}$: startup/shutdown ramp limits of generator g
  - $T_g^{\mathrm{u/d}}$: minimum up/downtime of generator g
  - P_t^{d}: power demand at time t
  - P_t^{r}: reserve requirement at time t
- Variables
  - $\alpha_{gt} \in \{0,1\}$: 1 if generator g is on in period t, and 0 otherwise
  - $\gamma_{gt} \in \{0,1\}$: 1 if generator g starts up in period t, and 0 otherwise
  - $\eta_{gt} \in \{0,1\}$: 1 if generator g shuts down in period t, and 0 otherwise
  - $p_{gt} \geq 0$: power output of generator g in period t
- Total cost (the objective to be minimised)

$$\min \sum_{t=1}^{T} \sum_{g=1}^{G} \left(C_g^{\mathrm{nl}} \alpha_{gt} + C_g^{\mathrm{mr}} p_{gt} + C_g^{\mathrm{up}} \gamma_{gt} \right).$$

- Load balance

$$\sum_{g=1}^{G} p_{gt} \geq P_t^{\mathrm{d}} \qquad t = 1, 2, \ldots, T.$$

- Reserve

$$\sum_{g=1}^{G} (P_g^{\max} \alpha_{gt} - p_{gt}) \geq P_t^{\mathrm{r}} \qquad t = 1, 2, \ldots, T.$$

- Power output bounds

$$P_g^{\min}\alpha_{gt} \le p_{gt} \le P_g^{\max}\alpha_{gt} \qquad g = 1,2,\ldots,G, t = 1,2,\ldots,T$$

- Ramp rate bounds

$$p_{gt} - p_{g\,t-1} \le P_g^{\mathrm{ru}}\alpha_{g\,t-1} + P_g^{\mathrm{su}}\gamma_{gt} \qquad g = 1,2,\ldots,G, t = 2,3,\ldots,T.$$

$$p_{g\,t-1} - p_{gt} \le P_g^{\mathrm{rd}}\alpha_{gt} + P_g^{\mathrm{sd}}\eta_{gt} \qquad g = 1,2,\ldots,G, t = 2,3,\ldots,T.$$

- Minimum up/downtime

$$\sum_{u=\max\{t-T_g^{\mathrm{u}}+1,1\}}^{t} \gamma_{gu} \le \alpha_{gt} \qquad g = 1,2,\ldots,G, t = 1,2,\ldots,T$$

$$\sum_{u=\max\{t-T_g^{\mathrm{u}}+1,1\}}^{t} \eta_{gu} \le 1 - \alpha_{gt} \qquad g = 1,2,\ldots,G, t = 1,2,\ldots,T$$

- Logical constraints (to enforce binaries to work as we expect)

$$\alpha_{gt} - \alpha_{g\,t-1} = \gamma_{gt} - \eta_{gt} \qquad g = 1,2,\ldots,G, t = 2,3,\ldots,T$$

$$1 \ge \gamma_{gt} + \eta_{gt} \qquad g = 1,2,\ldots,G, t = 1,2,\ldots,T$$

When the initial states of the generators are given, one can define α_{gt} and p_{gt} for $t = 0$ and $g = 1,2,\ldots,G$ accordingly and add the ramp rate bounds constraint and logical constraints for $t = 1$.

2.1.2 Compact formulation

The UC problem described in the previous section can be written as

$$\begin{aligned} \min_{x_1,x_2,\ldots,x_G} \quad & \sum_{g=1}^{G} {c_g}^T x_g \\ \text{s.t.} \quad & \sum_{g=1}^{G} A_g x_g = a(\omega) \\ & x_g \in X_g(\omega), \qquad g = 1,2,\ldots,G, \end{aligned} \tag{2.1}$$

where ω is a vector of problem data (e.g. demand), $x_1, x_2, \ldots, x_G$ are vectors of decision variables for each $g = 1,2,\ldots,G$, and

$$X_g(\omega) = \left\{ x_g = (u_g, z_g) \in \mathbb{R}^n \times \{0,1\}^m \mid D_g x_g \leq d_g(\omega) \right\}, \quad g = 1,2,\ldots,G.$$

We assume that $X_g(\omega)$ is a nonempty, bounded set and problem (2.1) has a feasible solution for any ω. We let $c^*(\omega)$ to denote the optimal objective value of (2.1) and $x^*(\omega)$ to denote any one of the optimal solutions. To reduce clutter in what follows we drop the dependence on ω except where this might cause confusion. We also write $x = (x_1^T, x_2^T, \ldots, {x_G}^T)^T$ and $X = X_1 \times X_2 \times \cdots \times X_G$.

In the remaining of this chapter we assume that ω is fixed and review solution methods for the UC problem.

2.2 Linear programming relaxation

The LPR is a relaxation which is obtained by relaxing the integrality in X_g:

$$\begin{aligned} \min_{x_1,x_2,\ldots,x_G} \quad & \sum_{g=1}^{G} {c_g}^T x_g \\ \text{s.t.} \quad & \sum_{g=1}^{G} A_g x_g = a(\omega) \\ & x_g \in \bar{X}_g(\omega), \qquad g = 1,2,\ldots,G, \end{aligned} \tag{2.2}$$

where

$$\bar{X}_g(\omega) = \left\{x_g = (u_g, z_g) \in \mathbb{R}^n \times [0,1]^m \mid D_g x_g \leq d_g(\omega)\right\}, \quad g = 1, 2, \ldots, G.$$

Let $\bar{c}^*$ denote the optimal objective value of (2.2). Since $X_g \subset \bar{X}_g$ for each g, this gives a lower bound on the optimal objective value:

$$\bar{c}^* \leq c^*.$$

2.3 Lagrangian relaxation

Lagrangian relaxation is another technique to compute lower bounds for problem (2.1). We note that the first constraint in (2.1) involves with all the variables and makes the problem difficult to solve. Without this complicating constraint the problem could be split into G subproblems. In this approach the complicating constraint is removed from problem (2.1) and a penalty of its violation is added to the objective. The modified objective function

$$L(x, y) = \sum_{g=1}^{G} {c_g}^T x_g - y^T \left(\sum_{g=1}^{G} A_g x_g - a \right) = \sum_{g=1}^{G} (c_g - {A_g}^T y)^T x_g + a^T y \tag{2.3}$$

is called the Lagrangian. The dual function is given by

$$q(y) = \min_{x \in X} L(x, y) = \sum_{g=1}^{G} \min_{x_g \in X_g} \left(c_g - {A_g}^T y\right)^T x_g + a^T y = \sum_{g=1}^{G} q_g(y) \tag{2.4}$$

where

$$q_g(y) = \min_{x_g \in X_g} \left(c_g - {A_g}^T y\right)^T x_g + \frac{1}{G} a^T y.$$

To evaluate component q_g we need to solve a single-generator scheduling problem on generator g. We refer to this single-generator scheduling problem as a subproblem. In general subproblems can be solved with an MILP solver. If the subproblem has a special structure, using a method exploiting the structure is likely to speed up the evaluation of q_g. For example, a method based on dynamic programming has been

developed to solve a UC problem with a single thermal generator, which will be discussed in Section 2.5. However, for simplicity the numerical experiments in this thesis use an MILP solver to solve the subproblems.

For any value of y the dual function gives a lower bound on the optimal objective value of the original problem:

$$\begin{aligned} q(y) &= \min_{x \in X} L(x, y) \\ &= \min_{x \in X} \left\{ \sum_{g=1}^{G} {c_g}^T x_g - y^T \left(\sum_{g=1}^{G} A_g x_g - a \right) \right\} \\ &\leq \sum_{g=1}^{G} {c_g}^T {x_g}^* - y^T \left(\sum_{g=1}^{G} A_g {x_g}^* - a \right) \\ &= \sum_{g=1}^{G} {c_g}^T {x_g}^* \\ &= c^*. \end{aligned}$$

The dual problem is the maximisation problem of the lower bound

$$\max_{y} \left\{ q(y) = \sum_{g=1}^{G} q_g(y) \right\}. \tag{2.5}$$

Being the pointwise minimum of a family of concave functions with respect to y, $q(y)$ and $q_g(y)$ for every ω and g are concave with respect to y [59, Theorem 5.5, page 35]. Furthermore, it can be shown [52, Theorem 6.2, page 327] that under the nonemptiness and boundedness assumption of X, the maximum of the dual problem, denoted by q^*, is attainable and coincide with the optimal objective value of the following problem:

$$\begin{aligned} \min_{x_1, x_2, \ldots, x_G} \quad & \sum_{g=1}^{G} {c_g}^T x_g \\ \text{s.t.} \quad & \sum_{g=1}^{G} A_g x_g = a \\ & x_g \in \text{conv}(X_g), \qquad g = 1, 2, \ldots, G, \end{aligned} \tag{2.6}$$

where $\text{conv}(X_g)$ is the convex hull of X_g for each g. (2.6) is a relaxation of (2.1) since $X_g \subset \text{conv}(X_g)$ for each g. The difference $c^* - q^* \geq 0$ between the optimal primal

and dual objective values is called the duality gap. We note that (2.6) can be written as an linear programming (LP):

$$\begin{aligned} \min_{p} \quad & \sum_{g=1}^{G} \sum_{i \in I_g} c_g^T x_{gi} p_{gi} \\ \text{s.t.} \quad & \sum_{g=1}^{G} \sum_{i \in I_g} A_g x_{gi} p_{gi} = a, \\ & \sum_{i \in I_g} p_{gi} = 1 \quad g = 1, 2, \ldots, G, \\ & p_{gi} \geq 0 \quad g = 1, 2, \ldots, G, i \in I_g, \end{aligned} \tag{2.7}$$

where $\{x_{gi} \mid i \in I_g\}$ are the extreme points of X_g for each g.

Simple calculation shows $\mathrm{conv}(X_g) \subset \bar{X}_g$ for each g and thus (2.2) is a relaxation of (2.6) and we have

$$\bar{c}^* \leq q^* \leq c^*.$$

Namely, Lagrangian relaxation gives a lower bound which is not worse than the LPR. In practice, we often observe that the inequality between $\bar{c}^*$ and q^* is strict and Lagrangian relaxation gives a strictly better lower bound than the LPR.

When the complicating constraint is inequality $\leq$ ($\geq$), the corresponding dual variable is constrained to be non-negative (non-positive).

2.3.1 Subgradient methods

In the following (Subsection 2.3.1 and 2.3.2) we review methods to solve the dual problem (2.5). Many references below study minimisation of a convex function rather than maximisation of a concave function. To facilitate the comparison we also consider minimisation of a convex function of the following form:

$$\min_{y \in Y} \left\{ f(y) = \sum_{g=1}^{G} f_g(y) \right\},$$

where Y is a closed, convex set and f_g is a convex function which is finite and subdifferentiable over Y for any g. We obtain this form from (2.5) by defining $f_g(y) = -q_g(y)$ for every g.

Subgradient methods are iterative methods to minimise a convex function [7]. When y is unconstrained, given the current point $y_k \in Y$, the next point is computed by

$$y_{k+1} = y_k - t_k s(y_k),$$

where $s(y_k)$ is any subgradient of f at $y = y_k$ and t_k is a positive parameter called the step size. When y is constrained, we apply the projection:

$$y_{k+1} = P_Y(y_k - t_k s(y_k)),$$

where P_Y is the Euclidean projection onto the set Y. If $s_g(y_k) \in \partial f_g(y_k)$ for each g, we have

$$\sum_{g=1}^{G} s_g(y_k) \in \partial f(y_k),$$

which follows from the definition of the subgradient. Thus, the algorithm can be written as Algorithm 1.

Algorithm 1 Subgradient method

select initial point $y_0 \in Y$, step size $\{t_k\}$.
for k in $\{1,2,\ldots\}$ **do**
  Evaluate $f_g(y_k)$ and $s_g(y_k) \in \partial f_g(y_k)$ for all $g \in G$.
  Let

$$y_{k+1} = P_y\left(y_k - t_k \sum_{g=1}^{G} s_g(y_k)\right).$$

end for

In the UC problems, given the optimal solution $\hat{x}_g$ to subproblem g at y, we get

$$A_g \hat{x}_g - \frac{a}{G} \in \partial f_g(y).$$

Thus, the iteration of the subgradient methods become

$$y_{k+1} = P_Y\left(y_k - t_k\left(\sum_{g=1}^{G} A_g \hat{x}_g - a\right)\right).$$

The subgradient methods have been widely used for convex optimisation problems due to its simplicity. There is also a rich body of literature to analyse the convergence property of the methods. However, the subgradient methods often

require a large number of iterations to find a solution of high precision.

2.3.2 Regularised cutting-plane methods

The cutting-plane methods construct a piecewise affine model of the objective function and compute candidate points $\{y_k\}$ using the model [34, 17]. In the kth iteration, given that the function and subgradient values are evaluated at the previous points $y_0, y_1, \ldots, y_k$, the method computes the next point y_{k+1} by minimising a model

$$y_{k+1} = \arg\min_{y \in Y} \left\{ \varphi_k(y) = \sum_{g=1}^{G} \varphi_{k,g}(y) \right\}, \tag{2.8}$$

where $\varphi_{k,g}$ is a piecewise affine model of the gth component

$$\varphi_{k,g}(y) = \max_{l=0,1,\ldots,k} f_g(y_l) + s_g(y_l)^T (y - y_l), \tag{2.9}$$

and $s_g(y_l)$ is any subgradient of f_g at y_l ($l = 0, 1, \ldots, k$). The function and subgradient values of the components are evaluated at the new point y_{k+1} to update the model (2.9) and the above process is repeated.

It is known that the plain cutting-plane methods suffer from instability. In the first iteration, after subgradients of the components are computed, the model is an affine function and likely to be unbounded. To make the methods working we need to add suitable cuts in the model. Even when the model is feasible, the progress of the methods is rather slow due to its instability.

One approach to tackle this issue is regularisation [16]. Instead of computing the next point by minimising the piecewise affine model, we use the following regularised model

$$y_{k+1} = \arg\min_{y \in Y} \sum_{g=1}^{G} \varphi_{k,g}(y) + \frac{1}{2t_k} \|y - \bar{y}_k\|^2, \tag{2.10}$$

where t_k is a parameter to adjust the strength of the regularisation and $\bar{y}_k$ is the regularisation centre.

The regularisation penalises large deviation of y_{k+1} from $\bar{y}_k$. When t_k is too large, the method behaves similar to the plain (non-regularised) cutting-plane method and would show instability, slowing down the progress. On the other hand, if t_k is too

small, the effect of the regularisation becomes dominant and the method cannot update the iterate sufficiently, again slowing down the progress. The appropriate value of t_k depends on the problem at hand. As we will discuss in Chapter 5, t_k has a close relationship to the step size in the subgradient methods so we refer to t_k as a step size in the context of the cutting-plane method as well.

In practice, the regularisation centre $\bar{y}_k$ and step size t_k are updated adaptively. Schulze et al. [61] update regularisation centre $\bar{y}$ to the current dual value y_k and increase t_k when the lower bound (2.4) has improved, and keep $\bar{y}$ and decrease t_k otherwise. Furthermore, it is beneficial to add cuts only if they are unique. We use these enhancements in our implementation. The algorithm is described in Algorithm 2. We note that to run the algorithm we need to specify the initial point y_0. As we see in Chapter 3, the choice of the initial point has a significant impact on the performance of the algorithm. One of the standard methods (which solves LPR to obtain the initial point) spends more than 20% of the total computational time doing so. We discuss this point in more depth in Chapter 3.

Algorithm 2 Regularised cutting-plane method

select initial point $y_0 \in Y$, step size $\{t_k\}$.
for k in $\{1,2,\ldots\}$ **do**
Evaluate $f_g(y_k)$ and $s_g(y_k) \in \partial f_g(y_k)$ for all $g = 1,2,\ldots,G$.
Update the regularisation centre $\bar{y}_k$ and t_k.
Let

$$y_{k+1} = \arg\min_{y \in Y} \widetilde{\varphi}_k(y), \tag{2.11}$$

$$\widetilde{\varphi}_k(y) = \sum_{g=1}^{G} \varphi_{k,g}(y) + \frac{1}{2t_k}\|y - \bar{y}_k\|^2,$$

$$\varphi_{k,g}(y) = \max_{l=0,1,\ldots,k} f_g(y_l) + s_g(y_l)^T (y - y_l).$$

end for

Note that the above algorithm is well-defined: $\widetilde{\varphi}_k$ is a proper closed $(1/2t_k)$-strongly concave function for each k and a minimise of $\widetilde{\varphi}_k$ exists and is unique [3, Theorem 5.25].

If the method is applied to the UC problem, the model (2.9) becomes

$$\varphi_{k,g}(y) = \max_{l=0,1,\ldots,k} \left(-c_g + {A_g}^T y\right)^T \hat{x}_{lg} - \frac{1}{G} a^T y, \tag{2.12}$$

where $\hat{x}_{lg}$ is the solution to subproblem g in the lth iteration.

Figure 2.1 is a schematic example of an application of the cutting-plane method to maximise a concave 1-dimensional function. The objective function consists of a single component ($G = 1$). The method evaluates the function and supergradient values of the objective function at the initial point y_0 as shown in the figure. The resulting unregularised model φ_0 has a single cut which is tangent to the objective function at $y = y_0$, which is shown by a dotted line. The regularised model $\widetilde{\varphi}_0$ is drawn by a dashed line, whose maximiser gives y_1. We note that the unregularised model φ_0 is unbounded and the maximiser does not exist. In the next iteration the function and supergradient values are evaluated at $y = y_1$ and another cut is added to the model. The regularised model $\widetilde{\varphi}_1$ yields y_2, which is closer to the optimal solution. In the remainder of the thesis, unless it may cause confusion, we drop the word "regularisation" when we refer to the regularised cutting-plane method.

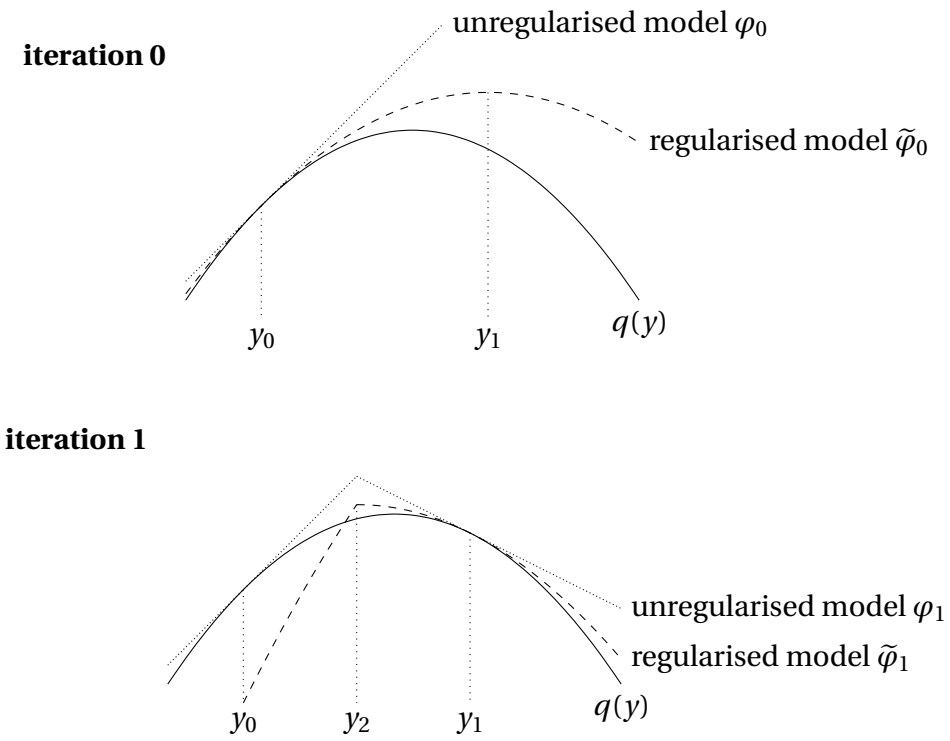

Figure 2.1: Skematic example of the behaviour of the cutting-plane method

2.3.3 Numerical experiments

In this section the subgradient method and the cutting-plane method are applied to a small 1-dimensional ($G = 1$) optimisation problem and their behaviour is analysed numerically. Figure 2.2 shows the objective function to be maximised. It is a piecewise-affine concave function which has the optimal solution $y^* = 2/3$ and the optimal objective value $q^* = 4/3$.

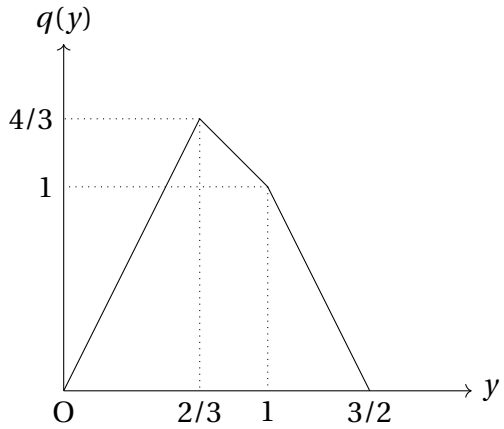

Figure 2.2: Objective function to be maximised

The initial point is set to $y = 0$ and the two methods are applied with step size

$$t_k = \frac{t_0}{k}, \qquad k = 1, 2, \ldots,$$

where t_0 is the initial step size.

The performance of the methods are shown in Figure 2.3. The subgradient method requires many iterations to achieve a solution with small suboptimality. When the step size is large ($t_0 = 0.5$), the iterate oscillates significantly and the method cannot find a solution with suboptimality smaller than 0.1% within 40 iterations. If the step size is small ($t_0 = 0.1$), the oscillation becomes smaller and the method finds a solution with suboptimality smaller than 0.1% in 25 iterations. However, even after the method finds a near-optimal solution, the following iterates keep to oscillate.

When the initial step size is 0.5 or 0.2, the cutting-plane method finds the optimal solution in 5 or fewer iterations. Moreover, unlike the subgradient method, after the optimal solution is found, the iterate stays to be optimal in the remaining iterations.

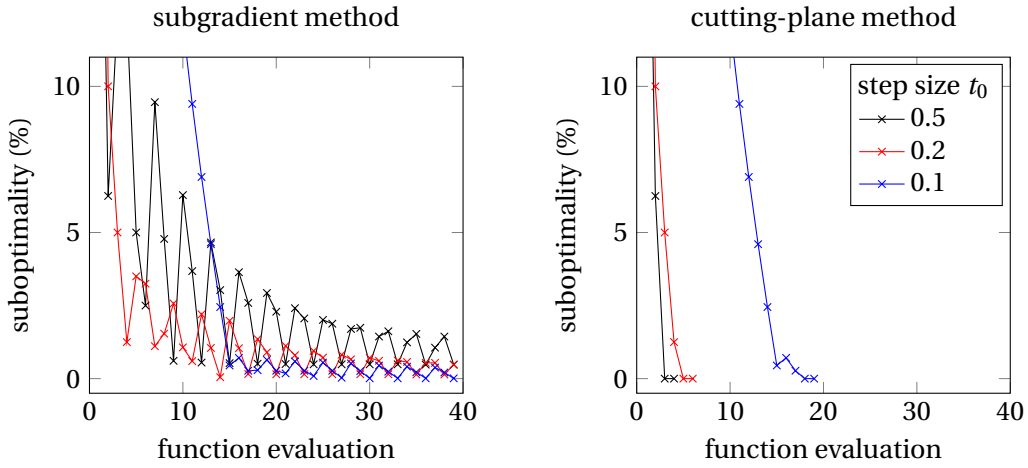

Figure 2.3: Suboptimality of the iterates

When the step size is too small (0.1%), the progress of the method in early iterations is identical to that of the subgradient method (with the step size 0.1%). However, it finds a solution of suboptimality 0.1% in 19 iterations and the iterate stays at optimal in the following iterations.

Since the two methods have different complexities per iteration, which depends on the problem to be solved, we cannot draw a definitive conclusion from this result. However, when evaluation of the objective function is expensive, it is advantageous for a solution method to require fewer function evaluations. This is typically the case when one needs to solve the dual of a combinatorial optimisation problem since evaluating the dual function amounts to solving the subproblems. The cutting-plane method would be a preferable choice in such a case.

2.4 Dantzig-Wolfe decomposition

Dantzig-Wolfe decomposition was originally proposed as a solution method for a LP problem by Dantzig and Wolfe [18]. For problems with integer variables, the Dantzig-Wolfe reformulation is a relaxation of the original problem and it provides a lower bound for the original problem (2.1). For further background, see Vanderbeck and Savelsbergh [66].

Replacing X_g with $\text{conv}(X_g)$ for every g, we obtain a relaxation of (2.1), referred to as the master problem (MP). Let $\{x_{gi} \mid i \in I_g\}$ be the extreme points of X_g. Given

the boundedness assumption on X_g, it follows that

$$\text{conv}(X_g) = \left\{ \sum_{i \in I_g} x_{gi} p_{gi} \;\middle|\; \sum_{i \in I_g} p_{gi} = 1, \quad p_{gi} \geq 0 \quad (i \in I_g) \right\}, \quad g = 1, 2, \ldots, G.$$

This implies that the MP can be written as an LP with decision variables

$$\{ p_{gi} \mid g = 1, 2, \ldots, G, \; i \in I_g \}.$$

However, finding all the extreme points is time consuming and leads to a formulation that is too large to solve explicitly, so a column generation procedure is used. The restricted master problem (RMP) is defined by replacing I_g in the MP with a subset $\hat{I}_g \subset I_g$ for every g. Thus, the RMP is given by

$$\begin{aligned} \min_{p} \; & \sum_{g=1}^{G} \sum_{i \in \hat{I}_g} {c_g}^T x_{gi} p_{gi} \\ \text{s.t.} \; & \sum_{g=1}^{G} \sum_{i \in \hat{I}_g} A_g x_{gi} p_{gi} = a, \\ & \sum_{i \in \hat{I}_g} p_{gi} = 1, \quad g = 1, 2, \ldots, G, \\ & p_{gi} \geq 0, \quad g = 1, 2, \ldots, G, i \in \hat{I}_g. \end{aligned} \tag{2.13}$$

Suppose that the RMP is feasible, which can be ensured by adding some artificial columns. Then the RMP has an optimal solution. Let y and σ_g for $g = 1, 2, \ldots, G$ be the optimal dual solution to the RMP corresponding to the first and second constraints respectively. Then the pricing subproblems are given by

$$r_g(y) = \min_{x_g} \{ ({c_g}^T - y^T A_g) x_g \mid x_g \in X_g \}, \quad g = 1, 2, \ldots, G. \tag{2.14}$$

If $r_g(y) \geq \sigma_g$ for all g, the optimal solution to the RMP is also optimal to the MP. Otherwise, the solutions to the pricing subproblems are added to the set $\hat{I}_g$ and the above process is repeated. It follows from LP duality that given dual values y

$$q(y) = a^T y + \sum_{g=1}^{G} r_g(y) \tag{2.15}$$

is a lower bound to the MP, which also bounds the optimal objective value in (2.1). We note that this lower bound is the same as the one used in Lagrangian relaxation (2.4).

The dual of (2.13) is

$$\begin{aligned}
&\max_{y,r} a^T y + \sum_{g=1}^{G} r_g \\
&\text{s.t. } r_g \leq (c_g - {A_g}^T y)^T x_{gi}, \quad g = 1,2,\ldots,G, i \in \hat{I}_g \\
&\quad y, r : \text{free},
\end{aligned}$$

where y and r are the dual variables of the first and second constraints in (2.13). The above problem is equivalent to (2.8) and the pricing subproblems are the same as the subproblems to evaluate the dual function up to a constant. That is, the cutting-plane method for the dual function and the column generation procedure for the RMP is the dual of each other. Therefore the same discussion on the cutting-plane method, especially its instability, is applicable to the column generation procedure for the RMP. By dualising the regularised model (2.10), we obtain a regularised version of the RMP

$$\begin{aligned}
&\min_{p,y} \sum_{g=1}^{G} \sum_{i \in \hat{I}_g} {c_g}^T x_{gi} p_{gi} + \frac{1}{2t_k}\left(\|y\|^2 - \|y_k\|^2\right) \qquad (2.16)\\
&\text{s.t. } \sum_{g=1}^{G} \sum_{i \in \hat{I}_g} A_g x_{gi} p_{gi} - \frac{1}{t}(y - y_k) = a, \\
&\quad \sum_{i \in \hat{I}_g} p_{gi} = 1 \quad g = 1,2,\ldots,G, \\
&\quad p_{gi} \geq 0 \quad g = 1,2,\ldots,G, i \in \hat{I}_g, \\
&\quad y : \text{free},
\end{aligned}$$

which mitigates the instability of the dual variables.

2.5 Dynamic programming

As discussed earlier, dynamic programming is not a practical approach to solve the UC problem as a whole. However, it is one of the most efficient method to solve the UC problem with a single thermal generator given the problem only consists of standard constraints such as those described in this thesis. A single-generator problem arises as a subproblem in Lagrangian relaxation or Dantzig-Wolfe decomposition. In the following we discuss dynamic programming approaches for the single-generator UC problem.

When the UC problem does not have the ramp rate bounds constraint, the optimal solution can be computed with standard dynamic programming. Without the ramp rate bounds constraint, when the generator is committed in the time period t, the optimal amount of power output $p_{g,t}$ in that time period can be computed independent of the decisions in other time periods. The resulted cost is added to the fixed cost of the corresponding commitment decision α_{gt}. Then the optimal commitment decision α_{gt} is computed by solving a shortest path problem. The optimal amount of power output can be computed quickly based on the optimal commitment decision.

The above approach fails when the ramp rate bounds constraint is considered since the amount of power output $p_{g,t}$ cannot be optimised independently. Fan et al. [23] propose a dynamic programming approach to solve the single-generator UC problem with a piecewise-linear cost function. In this approach a section of consecutive running or idle hours is considered as a state. The problem to compute the cost associated with consecutive running time periods is the economic dispatch problem, which can be solved efficiently by a constructive dynamic programming method as follows. The economic dispatch problem is first reformulated as a dynamic programming problem whose state represents the amount of power output in each time period. When the cost function is piecewise-linear, the cost-to-go functions in this dynamic programming problem are also piecewise-linear. This allows to represent the cost-to-go functions exactly and to obtain the optimal power dispatch. After solving the economic dispatch problems, finding the optimal commitment decision amounts to the standard shortest path problem.

Frangioni and Gentile [25] study a dynamic programming method which is applicable to problem with a general convex cost function. Their method is particularly

efficient when the cost function is quadratic. Their dynamic programming method also defines a state as a consecutive time period of running or idle hours. The economic dispatch problem is then solved with a dynamic programming approach to compute the cost associated with consecutive running hours. When the cost function is quadratic, the cost-to-go function becomes piecewise-quadratic and the economic dispatch problem can be solved using a dynamic programming approach. An implementation of this algorithm by the authors is available on the internet[1].

2.6 Primal heuristics

Both of Lagrangian relaxation and Dantzig-Wolfe decomposition are methods to compute lower bounds for problem (2.1). However, to find feasible solutions it is necessary to use a primal heuristic. Typically, a primal heuristic based on decomposition is run in each iteration of the solution methods (e.g. the subgradient method and the column generation procedure). Primal heuristics are discussed in Chapter 4.

2.7 Examples

In the remaining of this chapter we consider small examples.

Example 2.A We first consider a UC instance with a single time period and two generators ($G = 2$). The set of the generators are given in Table 2.1. To simplify the exposition we do not consider the startup cost ($C^{\text{up}} = 0$), the reserve constraint ($P^{\text{r}} = 0$), the minimum generation limit constraint ($P^{\min} = 0$), the ramp rate bounds constraint ($P^{\text{ru}}, P^{\text{rd}}, P^{\text{su}}, P^{\text{sd}} \gg 0$) and the minimum up/downtime constraints ($T^{\text{u}} = T^{\text{d}} = 0$).

For each point $x \in X$ the total cost

$$c^{\text{sum}} = c^T x = \sum_{g=1}^{G} {c_g}^T x_g$$

[1] https://gitlab.com/smspp/ucblock

Table 2.1: Generators in Example 2.A

generator g	1	2
no-load cost C_g^{nl}	1	2
maximum generation limit $P_g^{\max}$	1	3

and the total power output

$$p^{\mathrm{sum}} = Ax = \sum_{g=1}^{G} A_g x_g$$

are computed and plotted in the cost-power space in Figure 2.4. Solid lines and the origin correspond to X. The corresponding commitment schedules α are also shown in the figure. For example, $\alpha = (1,0)^T$ indicates that generator 1 is committed while generator 2 is not. For any value of demand P^{d}, by reading off the cheapest cost, we obtain the primal optimal objective value, which is shown in the figure as a function of the demand. For example, if the demand is 2, the optimal commitment schedule is $\alpha = (0,1)^T$ and the optimal primal objective value is 2.

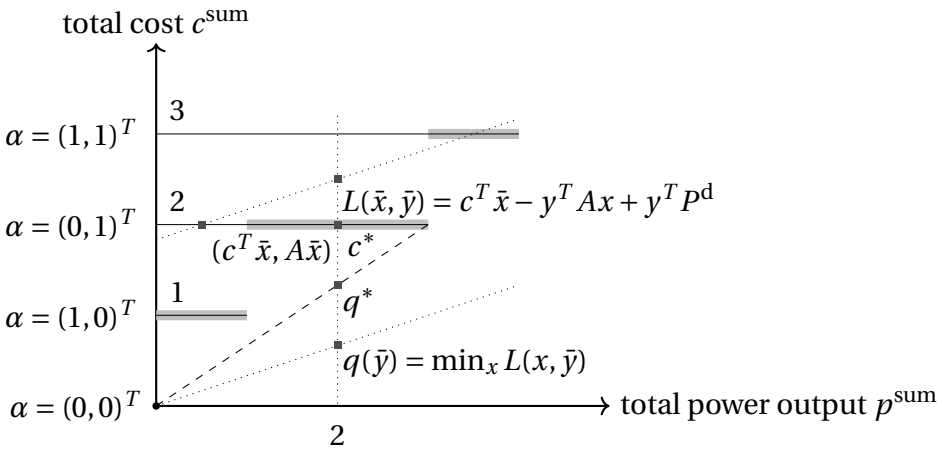

Figure 2.4: X (—, •), optimal primal (▬) objective value and the value of the dual function with $y = \hat{y}$ and the optimal dual objective value when $P^{\mathrm{d}} = 2$ (α: commitment decision)

Now assume that the demand is 2 and fix y, say to $\bar{y} \geq 0$, and pick a point $\bar{x}$ in X. The y-coordinate of the intersection of the line of slope $\bar{y}$ passing the point $(A\bar{x}, c^T\bar{x})$

and the vertical line corresponding to demand P^{d} gives the value of the Lagrangian

$$L(\bar{x}, \bar{y}) = c^T \bar{x} - \bar{y}^T \left(A\bar{x} - P^{\mathrm{d}} \right).$$

Fix the slope and vary the point $x \in X$ to minimise the height of the intersection. The resulting minimum y-coordinate of the intersection gives the lower bound

$$q(\bar{y}) = \min_{x \in X} L(x, \bar{y}).$$

By varying the values of y and repeating the above procedure we can find the optimal dual objective value

$$q^* = \max_{y \geq 0} q(y).$$

The optimal dual objective value q^* can be also found by using the fact that it is equal to the optimal objective value of problem (2.6), which is obtained by replacing X_g with $\mathrm{conv}(X_g)$ for each g in problem (2.1). $\mathrm{conv}(X)$ corresponds to the shaded area including the boundary in Figure 2.5. For any value of demand P^{d}, by finding the point with the smallest cost among those which meet the demand, we can find the optimal dual objective value. In this way we obtain the optimal dual objective value as a function of the demand, as shown in Figure 2.5.

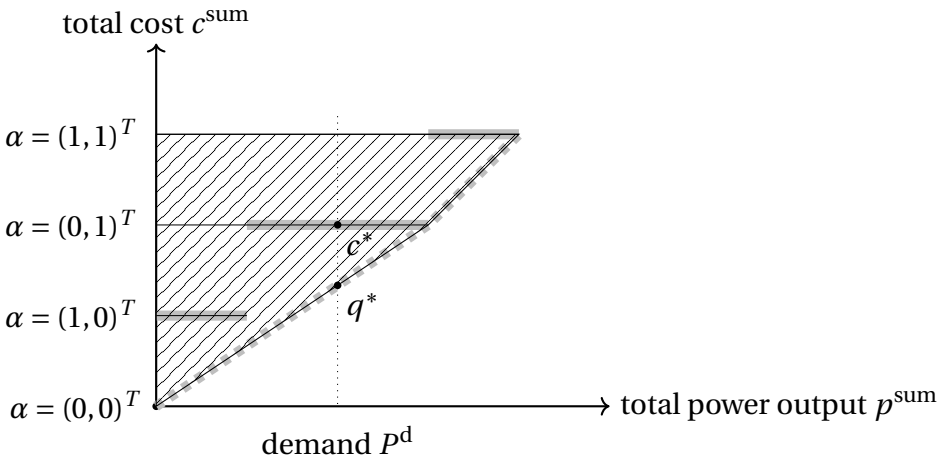

Figure 2.5: X (—, •), $\mathrm{conv}(X)$ (▨), optimal primal (▬) and dual (▪▪▪) objective values as functions of demand P^{d} (α: commitment decision)

In this simple example it is possible to compute the dual function explicitly. Since

the problem does not have the reserve constraint, it can be written as

$$\begin{aligned} c^*(P^{\mathrm{d}}) = \min_{\alpha,p} \sum_{g=1}^{G} \sum_{t=1}^{T} C_g^{\mathrm{nl}} \alpha_{gt} \\ \text{s.t.} \sum_{g=1}^{G} p_{gt} \geq P_t^{\mathrm{d}}, \qquad t = 1,2,\ldots,T, \\ (\alpha_g, p_g) \in X_g \subset \{0,1\}^T \times \mathbb{R}^T, \end{aligned}$$

where P^{d} is demand, α_g is the indicator of the on/off status of generator g and p_g is the power output of generator g. $G = 2$ and $T = 1$ in this example. Without the ramp rate bounds constraints and the minimum up/downtime constraints, there are no constraints which bind multiple time periods. Thus for each $g = 1,2,\ldots,G$ the set of the feasible schedules X_g is given by

$$X_g = \prod_{t=1}^{T} X_{gt},$$

where

$$X_{gt} = \left\{ (\alpha, g) \in \{0,1\} \times \mathbb{R} \mid P_g^{\min} \alpha \leq p \leq P_g^{\max} \alpha \right\}.$$

Denoting the dual variables associated to the load balance constraint by y, the dual function is given by

$$\begin{aligned} q(P^{\mathrm{d}}, y) &= \sum_{g=1}^{G} \left(\min_{(\alpha_g, p_g) \in X_g} \sum_{t=1}^{T} \left(C_g^{\mathrm{nl}} \alpha_{gt} - y_t p_{gt} \right) + \frac{1}{G} y^T P^{\mathrm{d}} \right) \\ &= \sum_{g=1}^{G} \left(\sum_{t=1}^{T} \min_{(\alpha_{gt}, p_{gt}) \in X_{gt}} \left(C_g^{\mathrm{nl}} \alpha_{gt} - y_t p_{gt} \right) + \frac{1}{G} y^T P^{\mathrm{d}} \right) \end{aligned}$$

for $y \geq 0$ and otherwise $q(P^{\mathrm{d}}, y) = -\infty$. One can solve the minimisation problem on the right-hand side by comparing the value of the dual variable y_t and the average generation cost at the maximum output $C_g^{\mathrm{nl}} / P_g^{\max}$. The optimal solution is to commit the generator and output power at its maximum limit if the corresponding dual value is larger, and to shut down the generator otherwise. For example, the dual function

with $P^{\mathrm{d}} = 2$ is given by

$$q(y) = \begin{cases} -\infty & y < 0 \\ 2y & 0 \le y \le \frac{2}{3}, \\ 2-y & \frac{2}{3} \le y \le 1, \\ 3-2y & 1 \le y. \end{cases}$$

We note that this is the function used in the numerical experiments in Subsection 2.3.3.

Example 2.B Now we compare the strengths of relaxations (2.2) and (2.6). To this end we consider a UC instance with a single generator and a single time period. The generator data is given in Table 2.2. In addition to the constraints considered in Example 2.A, we consider the ramp rate bounds constraint. We assume that the generator is off ($\alpha_{10} = p_{10} = 0$) initially. Under this assumption, since the ramp up limit is 2, the maximum feasible power output at time $t = 1$ is 2 ($\alpha_{11} = \gamma_{11} = 1$, $\eta_{11} = 0$ and $p_{11} = 2$).

Table 2.2: Generators in Example 2.B

generator g	1
no-load cost C_g^{nl}	1
maximum generation limit $P_g^{\max}$	3
ramp limits $P_g^{\mathrm{ru/rd/su/sd}}$	2
minimum up/downtime $T_g^{\mathrm{u/d}}$	0
initial status α_{g0}	0

X and the optimal primal and dual objective values as functions of the demand are drawn in Figure 2.6 in the same fashion as in Example 2.A. If we compute the optimal objective value of the LPR for various values of the demand, we obtain the dotted line in the figure. If the demand is larger than 0 but smaller than 2, we obtain

$$\bar{c}^* < q^* < c^*.$$

Namely, Lagrangian relaxation (2.6) gives a strictly better lower bound than the LPR

(2.2).

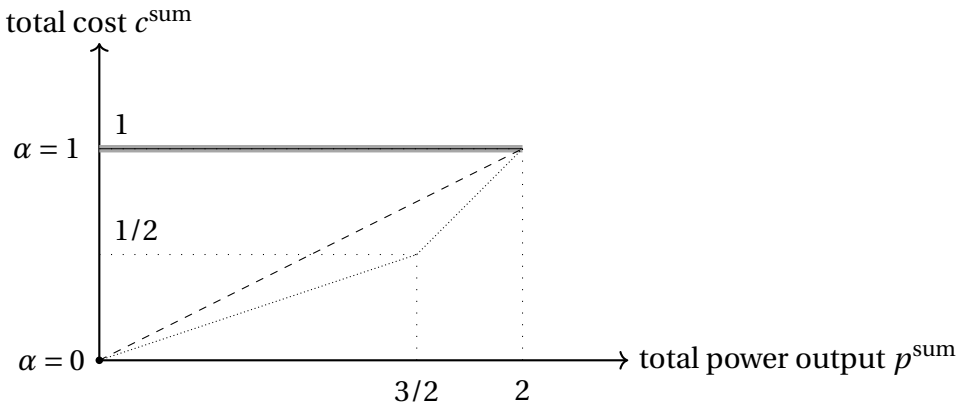

Figure 2.6: X (—, •), optimal primal (▬), dual (- - -) and LPR (·····) objective values (α: commitment decision)

When the demand is 3/2, the optimal solution to (2.6) is given by $\alpha_{11} = 3/4, p_{11} = 3/2, \gamma_{11} = 3/4, \eta_{11} = 0$ while the optimal solution to the LPR is given by $\alpha_{11} = 1/2, p_{11} = 3/2, \gamma_{11} = 3/4, \eta_{11} = 1/4$. We note that none of the points in X, and hence in $\text{conv}(X)$, has non-zero value of η_{11}. Thus we have

$$X \subsetneq \text{conv}(X) \subsetneq \bar{X}.$$

We note that the minimum up/downtime constraints with $T_1^{\text{u}} = T_1^{\text{d}} = 1$ does not alter X nor $\text{conv}(X)$. Thus, the constraints does not affect the optimal primal and dual objective values at all. However, the constraints tighten $\bar{X}$ significantly. As a result, with the minimum up/downtime constraints the optimal objective value of the LPR coincides with the optimal dual objective value for any value of demand P^{d}. It also modifies the values of the primal and dual optimal solutions to the LPR.

Example 2.C In the previous example, we saw that the LPR provides a looser lower bound than Lagrangian relaxation. However, the example did not work with the minimum up/downtime constraints with $T_1^{\text{u}} = T_1^{\text{d}} = 1$. In this example we explore a case that works even with the minimum up/downtime constraints with $T_1^{\text{u}} = T_1^{\text{d}} = 1$. Consider the 1-generator 2-period UC instance whose generator data is given in Table 2.3.

Table 2.3: Generators in Example 2.C

generator g	1
no-load cost C_g^{nl}	1
maximum generation limit $P_g^{\max}$	5
ramp limits $P_g^{\text{ru/rd/su/sd}}$	1
minimum up/downtime $T_g^{\text{u/d}}$	1
initial status α_{g0}	0

Let p_1^{sum} and p_2^{sum} denote the (total) power output in period 1 and 2 respectively and c^{sum} the sum of the costs in all the time periods. Figure 2.7 shows X in the 3-dimensional space of c^{sum}, p_1^{sum} and p_2^{sum}. This is analogous to Figure 2.4 and Figure 2.6.

The convex hull of X is plotted in the same manner as in Figure 2.8. For each value of demand $P^{\text{d}} = (P_1^{\text{d}}, P_2^{\text{d}})$, find the point of the minimal cost in the convex hull such that the power outputs meet the demand. In this way the optimal dual objective value as a function of the demand is obtained. Its contour plot is shown on the left side of Figure 2.9.

Additionally, for different values of the power outputs, we can formulate the LPR and obtain their optimal objective values, which are drawn on the right side of Figure 2.9. We observe that the LPR gives a looser lower bound than Dantzig-Wolfe decomposition when the power outputs are positive on both time periods. For example, when the power output is $(1,2)$, the LPR gives $7/5$ as a lower bound while Lagrangian relaxation gives 2.

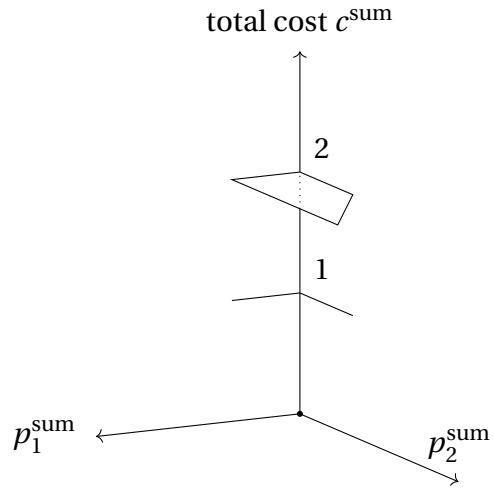

Figure 2.7: X (the origin, the line segments and the surrounded area)

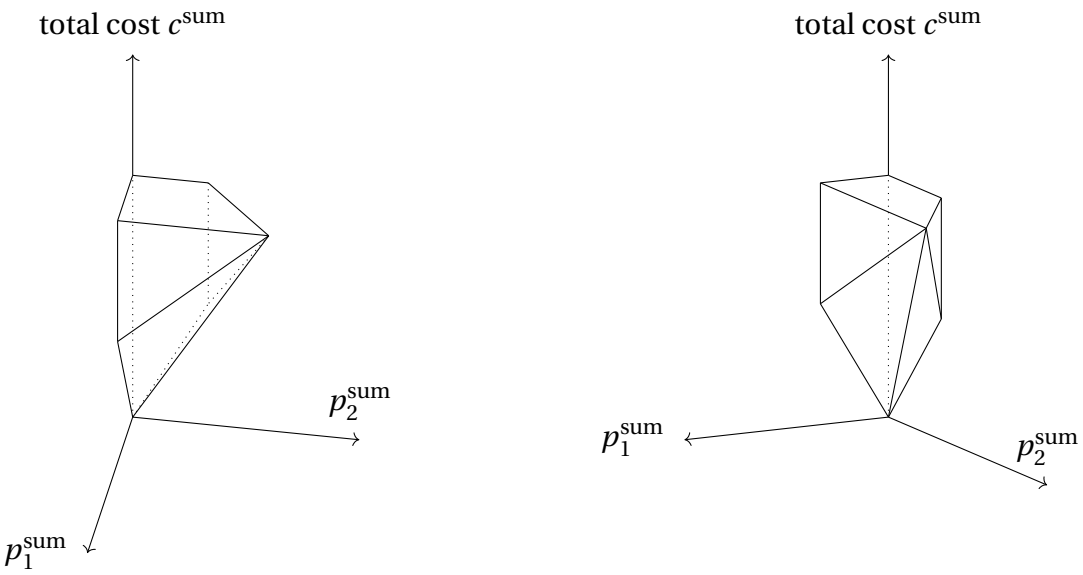

Figure 2.8: Convex hull of X (shown from two different angles)

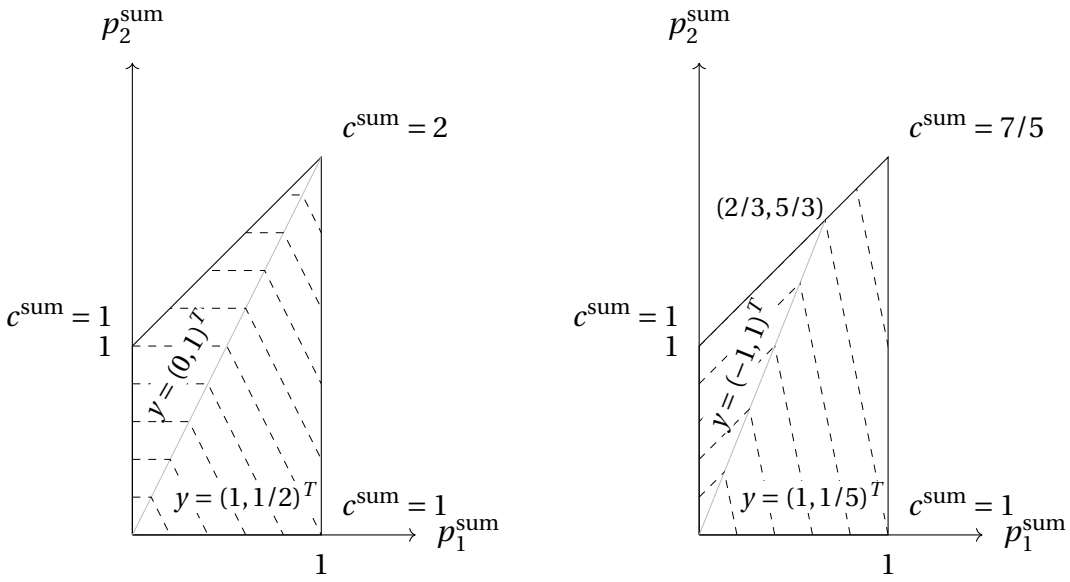

Figure 2.9: Contour plots of the optimal dual objective value (left) and the optimal LPR objective value (right)

Chapter 3

Initialisation Methods

In the column generation procedure the dual variables y play a key role. Consider the case where near-optimal dual values are used as the initial point. It can be shown that the lower bound $q(y)$ is continuous in the dual variables y. Thus the algorithm would provide a near-optimal lower bound in the first iteration. Assuming that the lower bound is sufficiently tight, the algorithm would terminate as soon as a primal feasible solution with sufficiently small suboptimality is found by a primal heuristic. It is expected that feeding near-optimal dual values as the initial dual values helps the algorithm to terminate in a shorter time.

One approach to generate such dual values is to solve an approximation of the original problem and obtain its optimal dual solution. Borghetti et al. [15] and Schulze et al. [61] relax the integrality constraints and obtain a continuous relaxation while Takriti et al. [63] further drop other constraints such as minimum up/down time constraints and minimum power output constraints. However, the continuous relaxation has a similar number of the variables and constraints to the original problem, and solving it even without the integer variables takes a significant computational time.

Another approach is to use a pre-trained model to generate the initial dual values. Recall that the dual problem is given by $\max q(y)$ where

$$q(y) = \min_{x \in X} L(x, y).$$

Problems of this form are referred to as structure prediction [54]. One method to

solve this problem is based on surrogate model or Bayesian optimisation [33, 21]. First a surrogate model is fitted to predict the value of $q(y)$ for any y. After the training, given new values of x, the surrogate model is maximised instead of $q(y)$, which is expensive to evaluate.

In our setting, since $q(y)$ has a decomposable structure (2.15), it is expected to be advantageous to use a specialised method. Nair et al. [48] study a method which can exploit such a structure. In their work neural networks were trained to compute dual values to yield strong lower bounds on integer two-stage stochastic programming. In our work we combine the neural network with the column generation procedure to solve UC and obtain guaranteed lower and upper bounds on the optimal objective value. After the training, when solving a new instance, the neural network is used with problem data as input to generate dual values that yield tight initial lower bounds. The generated dual values are then used to warmstart the column generation procedure and the column generation procedure allows us to further tighten the lower bound and obtain feasible solutions. With this approach, we can exploit the strength of the neural network while maintaining the desirable property of the solution method, such as exactness.

Even if we find a high quality initial point, regularisation on the dual variables is crucial [14]. Without appropriate regularisation, even if near-optimal dual values are used as the initial point, the algorithm is likely to be quite unstable, yielding dual values with poor lower bounds in the following iterations. Fig 3.1 shows an example of too weak regularisation. The objective function is $-|y|$ and the norm of a supergradient is always 1 unless y is equal to 0. In the figure, y_0 is set to close to the optimal solution y^*. However, since the regularisation is too weak, the next point gets far from the solution. Interestingly, as y_0 approaches to y^*, y_1 gets worse in a sense that y_1 becomes farther from y^* and that it yields a looser lower bound.

3.1 Initialisation method based on the LPR

As described at the beginning of this chapter one option to find good dual values is to solve some approximation of (2.1), such as the LPR. In this section we study the property of the initialisation based on the LPR. To this end, we assume that we have

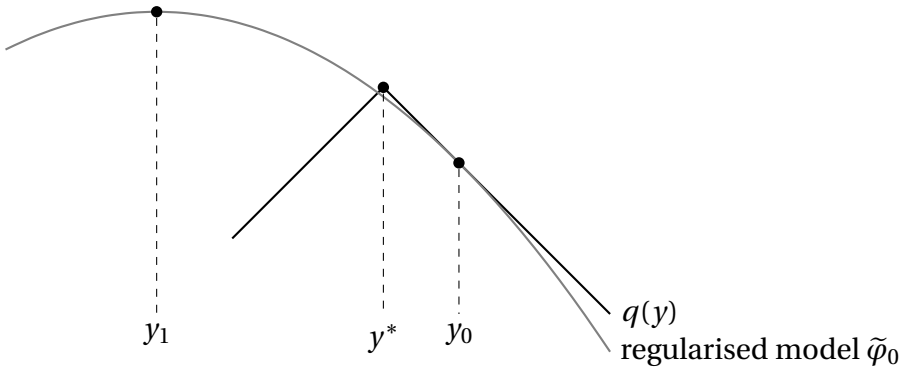

Figure 3.1: Example of too weak regularisation

sufficiently large generation capacity to meet the demand and reserve constraints. That is, there exists x in the relative interior of $\bar{X}$ such that

$$\sum_{g=1}^{G} A_g x_g = a.$$

Since the LPR is a relaxation of problem (2.1), the optimal objective value of the LPR, denoted by $\bar{c}^*$, gives a lower bound on the optimal objective value c^* of (2.1). Furthermore, being a relaxation of the MP, the LPR gives a lower bound on the optimal lower bound q^* given by Dantzig-Wolfe decomposition (and Lagrangian relaxation). That is, we have

$$\bar{c}^* \leq q^* \leq c^*.$$

Let $\bar{y}$ be the optimal dual values to the LPR. Using the definition,

$$q(\bar{y}) = \min_{x \in X} L(x, \bar{y}) \geq \min_{x \in \bar{X}} L(x, \bar{y}).$$

The standard discussion on duality [8, Proposition 5.3.3] shows that the optimal solution $\bar{x}$ to the LPR is also a solution to the minimisation problem on the right-most hand side

$$\min_{x \in \bar{X}} L(x, \bar{y}) = L(\bar{x}, \bar{y}) = \sum_{g=1}^{G} {c_g}^T \bar{x}_g - \bar{y}^T \left(\sum_{g=1}^{G} A_g \bar{x}_g - a \right) = \sum_{g=1}^{G} {c_g}^T \bar{x}_g - \bar{y}^T 0 = \bar{c}^*.$$

Therefore we established the following relation:

$$\bar{c}^* \leq q(\bar{y}) \leq q^* \leq c^*.$$

Namely evaluating the dual function at the dual values obtained from the LPR we get a lower bound which is not worse than the lower bound given by the LPR.

3.1.1 Examples

In this section we study the performance of the initialisation method based on the LPR on small examples.

Example 3.A The first example is a 2-generator 1-period UC instance, whose generator data is given in Table 3.1. For simplicity we do not consider the startup cost ($C^{\text{up}} = 0$), the reserve constraint ($P^{\text{r}} = 0$), the minimum generation limit constraint ($P^{\min} = 0$) and the minimum up/downtime constraints ($T^{\text{u}} = T^{\text{d}} = 0$). We note that this is an extension of Example 2.B in Chapter 2.

Table 3.1: Generators in Example 3.A

generator g	1	2
no-load cost C_g^{nl}	2/3	1
maximum generation limit $P_g^{\max}$	1	3
ramp limits $P_g^{\text{ru/rd/su/sd}}$	1	2
minimum up/downtime $T_g^{\text{u/d}}$	0	0
initial status α_{g0}	0	0

Figure 3.2 shows X in the space of the total cost c^{sum} and the total power output p^{sum}. Each of the solid lines and the origin corresponds to a single feasible generator commitment decision, which is shown aside. The figure also shows the optimal primal, dual and LPR objective values. The optimal dual and LPR objective values are piecewise affine convex functions in the demand.

Suppose that the demand is fixed. We can solve the LPR and obtain its optimal dual solution $\bar{y}$ to initialise the column generation procedure. The optimal LPR objective value $\bar{c}^*$ is a lower bound on the optimal dual objective value q^*. Even if

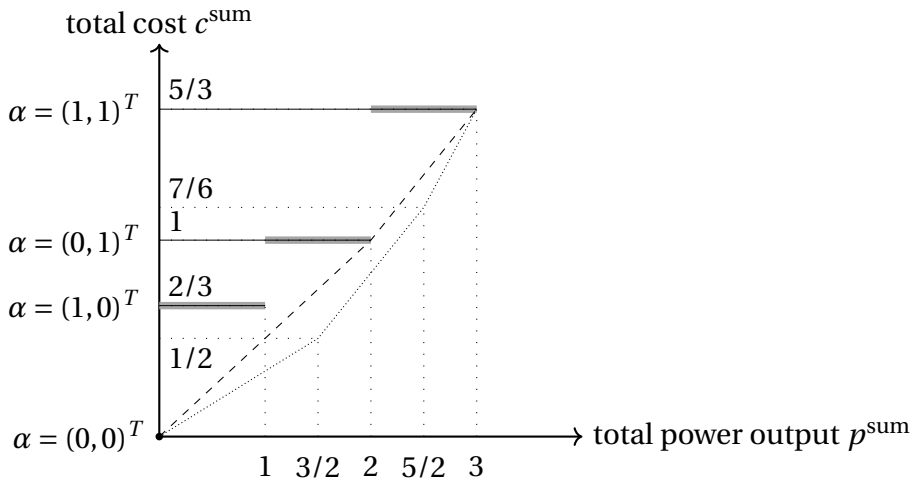

Figure 3.2: X (—, •), optimal primal (▬), dual (- - -) and LPR (·····) objective values (α: commitment decision)

the LPR provides a loose lower bound, the Lagrangian dual function evaluated at the optimal dual solution $\bar{y}$ to the LPR may give a tighter lower bound.

In Figure 3.2 the optimal dual solution to the LPR is given by the slope of the function of the optimal LPR objective value. More precisely, the optimal dual solution is any one of the subgradients of the function. The optimal dual and LPR objective values are parallel between 2 and 5/2 (boundaries are not included in this section). This observation shows us that if the demand is between 2 and 5/2 the optimal dual solution $\bar{y}$ to the LPR is also optimal to the dual problem

$$\bar{c} < q(\bar{y}) = q^*.$$

In other words, the LPR initialisation gives the dual values optimal to the column generation procedure and the procedure yields the optimal lower bound in the first iteration. We note that although the optimal lower bound is found in the first iteration, the column generation procedure must run a few iterations and generate enough columns to prove the optimality. If the demand is between 0 and 3/2 or between 5/2 and 3, evaluating the Lagrangian dual function at the optimal dual solution to the LPR gives the same lower bound as the LPR

$$\bar{c} = q(\bar{y}) < q^*.$$

If the demand is between 3/2 and 2, the Lagrangian dual function evaluated at the optimal dual solution to the LPR gives a lower bound that is better than the optimal LPR objective value but not optimal to the Lagrangian dual problem

$$\bar{c} < q(\bar{y}) < q^*.$$

This example carries the same limitation as Example 2.B in Chapter 2. That is, the minimum up/downtime constraints with $T^{\mathrm{u}} = T^{\mathrm{d}} = 1$ would tighten the LPR and the optimal LPR objective value $\bar{c}^*$ would become the same as the optimal dual objective value q^*. In the next example, we will study another UC instance which works with the minimum up/downtime constraints.

Example 3.B In this section we consider an example which extends Example 2.C in Chapter 2. The example consists of two generators and two time periods. The set of generators are given in Table 3.2. We do not consider the startup cost ($C^{\mathrm{up}} = 0$), the reserve constraint ($P^{\mathrm{r}} = 0$) and the minimum generation limit constraint ($P^{\min} = 0$).

Table 3.2: Generators in Example 3.B

generator g	1	2
no-load cost C_g^{nl}	1	1
maximum generation limit $P_g^{\max}$	5	1
ramp limits $P_g^{\mathrm{ru/rd/su/sd}}$	1	1
minimum up/downtime $T_g^{\mathrm{u/d}}$	1	1
initial status α_{g0}	0	0

For each $x \in X$ the total cost and power output in each time period are computed as plotted in Figure 3.3. Similarly $\mathrm{conv}(X)$ is plotted in the same space in Figure 3.4. Following the same procedure as Example 2.C in Chapter 2, we obtain the optimal dual and LPR objective values as functions of the demand. They are plotted in Figure 3.5 and 3.6, respectively.

We observe that if the demand is small (i.e. $P_1^{\mathrm{d}} \le 1$ and $P_2^{\mathrm{d}} \le P_1^{\mathrm{d}} + 1$), the optimal dual and LPR objective values coincide with those in Example 2.C. This is because generator 1, which is the same as the one in Example 2.C, is on average cheaper

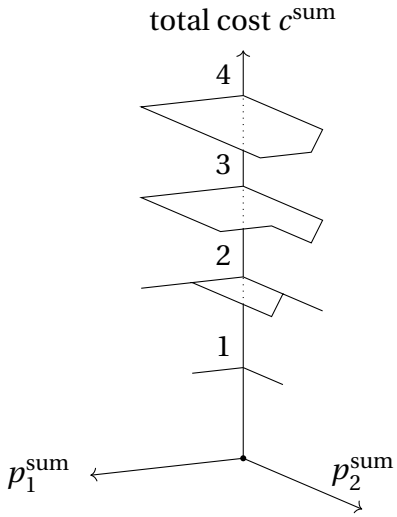

Figure 3.3: X (the origin, the line segments and the surrounded area)

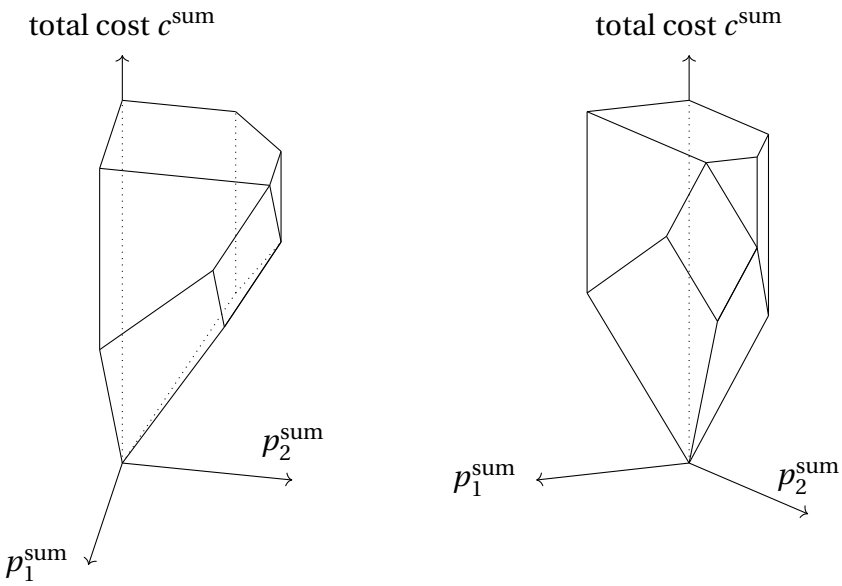

Figure 3.4: Convex hull of X (shown from two different angles)

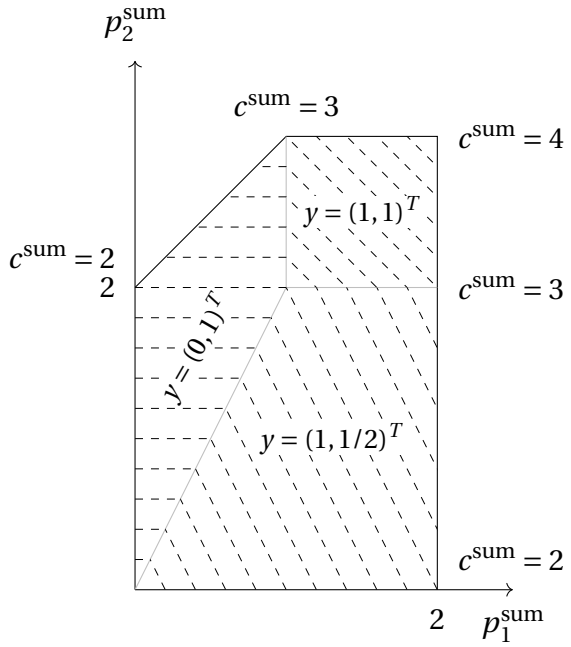

Figure 3.5: The contour plot of the optimal dual objective value

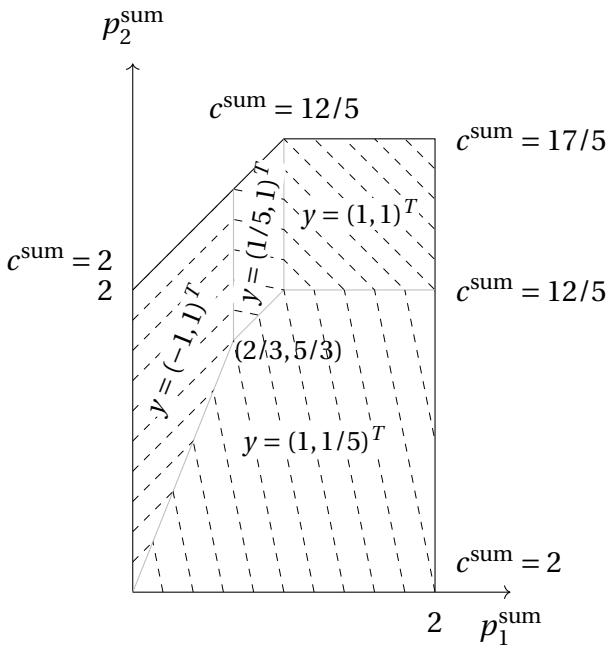

Figure 3.6: The contour plot of the optimal LPR objective value

than generator 2. Thus, when the demand is small, only generator 1 is used. When the demand is large (e.g. $P_1^{\mathrm{d}} > 1$ and $P_2^{\mathrm{d}} > 2$), generator 2 must be used to meet the demand. In both cases the gradient of the optimal dual and LPR objective values are $(1,1)$. That is, the two functions (the optimal dual and LPR objective value) is parallel. Therefore the optimal dual solution to the LPR is also optimal to the dual problem:

$$\bar{c} < q(\bar{y}) = q^*.$$

3.2 Initialisation method based on a neural network

An alternative approach, which we explore below, is to train a model which takes ω as input and outputs dual values y that provides a tight lower bound. The training approach is based on the work by Nair et al. [48] where dual decomposition was applied to solve a parametrized two-stage stochastic programming problem.

Consider a neural network $f(\omega,\theta)$ which maps problem data ω and model parameter θ to y. We aim to learn values of the parameter θ so that given ω the model outputs dual values $y = f(\omega,\theta)$ for which the lower bound $q(y)$ in (2.15) is tight. In this section we explicitly state the dependency of the lower bound on ω as $q(\omega, y)$, which was suppressed in (2.15). Assume that the distribution of ω is given (e.g. the empirical distribution based on historical data). Our goal is to maximise the expected lower bound

$$p(\theta) = \mathrm{E}_\omega[q(\omega, f(\omega,\theta))].$$

If subproblem g is sampled uniformly from $1,2,\ldots,G$ from (2.15) it follows that

$$q(\omega, y) = a^T y + \sum_{g=1}^{G} r_g(y) = \frac{1}{|G|}\sum_{g=1}^{G}(a^T y + |G| r_g(y)) = \mathrm{E}_g[\widetilde{q}(\omega, y, g)],$$

where

$$\widetilde{q}(\omega, y, g) = a^T y + |G| r_g(y).$$

Therefore we get

$$p(\theta) = \mathrm{E}_{\omega,g}[\widetilde{q}(\omega, f(\omega,\theta), g)].$$

We follow the standard approach used for training a neural network which is to use the stochastic gradient ascent method. That is, we sample ω and g, compute the gradient of $\widetilde{q}$ with respect to θ and make a single gradient ascent step. We then resample ω and g and repeat the process.

The gradient of $\widetilde{q}$ with respect to θ can be computed as follows. Fix problem data ω and subproblem index g and compute the duals value $y = f(\omega,\theta)$ and the component $\widetilde{q}(\omega, y, g)$ of the lower bound corresponding to subproblem g. Suppose that the model output $f(\omega,\theta)$ is differentiable with respect to θ and the optimal value $r_g(y)$ of the pricing subproblem (2.14) are differentiable with respect to y. Then the gradient of $\widetilde{q}(\omega, y, g)$ with respect to y is given by

$$\frac{\partial \widetilde{q}}{\partial y} = a - |G| A_g {x_g}^*$$

where ${x_g}^*$ is the solution to the pricing subproblem g (2.14). Using the chain rule, we obtain

$$\frac{\partial \widetilde{q}}{\partial \theta} = J \frac{\partial \widetilde{q}}{\partial y}, \tag{3.1}$$

where J is the Jacobian matrix of the neural network output $y = f(\omega,\theta)$ with respect to θ, which is typically given by automatic differentiation.

It is important to note that evaluating the above quantity (3.1) is much faster than solving the original problem or the MP to optimality. This is because it requires the solution of only a single pricing subproblem, which is typically substantially smaller than the original problem.

Once a model is trained, it can be used to compute initial dual values for the regularised column generation procedure. It is expected that unlike solving an approximation such as the LPR this model will run quickly and we later demonstrate that it produces near-optimal initial dual values.

3.3 Numerical experiments

In this section the proposed initialisation method based on a neural network is compared with standard approaches for the UC problem. All methods are implemented in Python. IBM ILOG CPLEX 20.1.0[1] is used as the optimisation solver (the barrier method for the RMP and the branch and bound method for the pricing subproblems) and PyTorch to implement the neural networks. The experiments are done on a workstation with 16 cores Intel® Xeon® E5-2670 with 126 GB of RAM.

3.3.1 Problem

In the experiments, we consider a setup in which UC problems are solved repeatedly with a fixed set of generators but different demand forecasts. To assess the scalability, we consider 3 different problem sizes; problems with 200, 600 and 1,000 generators. In all cases, the length of the planning horizon is 48 hours with a time resolution of 1 hour. The generator data is taken from an earlier study on UC problem by Frangioni and Gentile [25]. It is realistic synthesised data generated to conduct numerical experiments in the reference. Since their sets of generators contain 200 generators at most, we combine multiple sets to create larger ones. For example, to create a UC instance with 1,000 generators, we combine 5 distinct 200-generator sets. Each generator is unique so combining these sets does not introduce symmetry. The demand data is based on the historical demand data in the UK published by National Grid ESO.[2]

Dantzig-Wolfe decomposition with the regularised column generation procedure is used to solve the UC problems. Various initialisation methods are compared, and these are described in the next section. In each iteration of the column generation procedure, a local search primal heuristic is run to find a primal feasible solution. In the first 30 iterations we use the one proposed by Guan et al. [29], and later we run the one used by Takriti and Birge [62] as well. The detail of the primal heuristic is described in Chapter 4.

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio
[2] https://www.nationalgrideso.com/

3.3.2 Initialisation methods

We consider the following 5 methods to initialise the dual variables.

Neural Network: An initialisation method based on a neural network is implemented as described in Section 3.2. For each set of generators, a single neural network is trained for 24 hours using 8 CPU cores. We follow the procedure used by Nair et al. [48], except that only a single neural network is trained for each set of generators to reduce the computational load.

The model consists of 4 hidden layers of 1000 units per layer, with skip connections between hidden layers. The tanh function is used as an activation function. Those hyperparameters are chosen by greedy search in earlier experiments. For example, we observed that tanh performed better than ReLU, which is a more standard activation function. As De and Smith [19] proposed, the weights on the residual connections are initialised to zero. The other weights are initialised based on the methods of Glorot and Bengio [27]. All biases are initialised to zero.

Adam (Kingma and Ba [36]) is used to learn the parameters of the neural network. The learning rate is initialised to 10^{-4}. The model performance is evaluated every 5 minutes, and if it fails to improve for successive 15 minutes, the learning rate is divided by 1.5.

Random forest: As a baseline of a trainable model, a method based on a random forest (Briant et al. [16]) is also evaluated. On the same training budget (24 hours on 8 CPU cores), as many training instances as possible are solved to 0.25% optimality using Dantzig-Wolfe decomposition with the LPR initialisation and the local search primal heuristic. The parameter values and the corresponding optimal dual values of the training instances are saved. After the timelimit, a random forest is trained by supervised learning to predict the optimal dual values given a parameter value. The training took less than 30 seconds in all cases. Unlike our neural network approach, this approach requires the solution of many UC instances.

Nearest Neighbour: A nearest neighbour approach is also evaluated to provide an alternative trainable baseline. We use the dataset that was created to train the random

forest model. At runtime, given a test instance, its parameter is compared against those of the training instances. The training instance with the closest parameter value in terms of the Euclidean distance is selected, and the corresponding optimal dual values are used as initial dual values. As in the random forest approach, this requires the solution of many UC instances.

LPR: This is the method based on the LPR described in Section 3.1. Given a test instance, we first solve the LPR by CPLEX. This method does not require any training. This is the method used by Schulze et al. [61].

Coldstart: As a naive base line, column generation is run from $y = 0$. This method does not require any training.

3.3.3 Results

To test the performances of the above methods, 40 test instances of each size were created and solved by each method. The demand data to construct the test instances was sampled from a different year than those of the training instances. Each method was run sequentially on a single CPU core until a solution within 1%, 0.5% and 0.25% suboptimality was found or the time limit of 10 minutes was reached.

Table 3.3 shows the tightness of the initial lower bound obtained in the first iteration of the column generation procedure and the required time (all times in this section are wall clock times) to run the initialisation methods. For each test instance, a lower bound on the optimal objective value was also computed by running CPLEX for 4 hours. Then the initial lower bounds obtained from the column generation procedure were compared with the best lower bound found by CPLEX. For all the test instances the lower bounds found by CPLEX after 4 hours were better than the initial lower bounds from the column generation procedure. In the table the average gaps of the bounds are reported. For comparison, the objective value of the LPR (which is also a valid lower bound) is shown in the table as well (labelled as LPR alone).

Clearly, coldstart with $y = 0$ yields poor lower bounds. Although the lower bounds of the remaining 4 methods are comparable, the lower bounds obtained from the methods based on machine learning are significantly tighter than LPR. The solution

Table 3.3: Tightness of the initial lower bounds lb_1 (%) and computational time (s). The tightness of initial lower bounds lb_1 is measured by comparing it with the best known lower bound lb^* obtained by running CPLEX for 4 hours and is reported as percentages.

	size: 200		600		1000	
method	$lb^* - lb_1$	time	$lb^* - lb_1$	time	$lb^* - lb_1$	time
network	0.059	<0.1	0.051	<0.1	0.040	<0.1
forest	0.060	<0.1	0.073	<0.1	0.076	<0.1
nearest	0.048	<0.1	0.060	<0.1	0.063	<0.1
LPR	0.126	4.6	0.108	17.7	0.105	29.2
coldstart	99.854	0.0	99.831	0.0	99.873	0.0
LPR alone	0.179	4.6	0.176	17.7	0.163	29.2

time of the LPR grows and it takes significant time especially on larger problems. On the other hand, the computational time of the other methods remains small. Observe that the optimal LPR objective value gives a lower bound, but evaluating the Lagrangian using the solution to the LPR yields a tighter lower bound.

Table 3.4 shows the number of problems solved by the column generation procedure within the time limit of 10 minutes, the average computational time and the average number of iterations to close the optimality gap or to reach the time limit. The computational time reported in this table includes the time to run the initialisation methods such as solving LPR as well as the time to solve the RMP and the pricing subproblems. For the instances that are not solved within the time limit, the time is set to be 10 minutes and the number of iterations reached by the 10 minute timelimit is used. For comparison, we also solve the extensive form (2.1) to the same tolerances without decomposition (i.e. by branch and bound) using CPLEX.

These results show that solving the problems without decomposition is the slowest. Among the remaining 5 methods, which are based on the column generation procedure, solving the problem with coldstart is by far the slowest in every case. The 3 methods based on neural network, random forest and nearest neighbour are faster in all cases than the method based on the LPR. The nearest neighbour is the slowest in many cases. The neural network and random forest have a similar performance on the loose tolerances 1.0% and 0.5% but the neural network outperforms all the other methods on 0.25% tolerance.

Table 3.4: Number of problems solved, average computational time (s) and iterations.

		1% optimality			0.5% optimality			0.25% optimality		
size	method	solved	time	iters	solved	time	iters	solved	time	iters
200	network	40	4.6	1.0	40	9.0	1.9	40	**37.5**	7.4
	forest	40	**4.3**	1.0	40	**8.0**	1.8	40	42.0	8.2
	neighbour	40	4.7	1.0	40	10.0	2.1	40	43.1	8.7
	LPR	40	7.8	1.0	40	15.9	3.0	40	53.6	10.8
	coldstart	40	103.8	16.6	40	161.1	21.7	40	235.6	27.4
	CPLEX	39	235.8	-	39	244.1	-	38	336.9	-
600	network	40	**13.4**	1.0	40	**20.5**	1.5	40	**48.4**	3.4
	forest	40	14.0	1.1	40	26.5	2.0	40	55.5	4.2
	neighbour	40	13.8	1.1	40	27.6	2.0	40	57.8	4.4
	LPR	40	28.7	1.0	40	43.0	2.0	40	86.9	5.4
	coldstart	36	413.4	13.8	16	528.2	16.6	9	576.3	17.7
	CPLEX	4	594.1	-	4	594.1	-	4	594.1	-
1000	network	40	21.5	1.0	40	**34.3**	1.5	40	**74.2**	3.2
	forest	40	**19.5**	1.0	40	37.1	1.8	40	89.7	4.3
	neighbour	40	23.5	1.2	40	46.5	2.1	40	96.5	4.4
	LPR	40	45.6	1.0	40	67.4	1.8	40	120.4	4.2
	coldstart	23	515.8	12.2	7	567.8	13.6	5	584.0	14.0
	CPLEX	1	599.0	-	1	599.0	-	1	599.0	-

Table 3.5 shows the 20% and 80% quantiles of computational time. We observe that all the methods have relatively small variation of computational time. In particular, we notice that the 80% quantile of the computational time with neural network is always less than that of LPR. This shows the robustness of our method and many instances can be solved in a short time.

To observe the effect of the quality of the initial dual values on the time taken by the column generation procedure more clearly, in Figure 3.7 the total computational time (0.25% tolerance) for each 1000-generator test instance is plotted against the tightness of the initial lower bound. The initialisation methods based on a neural network and the LPR are compared in the plot. We observe a correlation between the two metrics. That is, if the lower bound computed in the first iteration of the column generation procedure is tighter then the total computational time required by the column generation procedure tends to be smaller.

The method to generate initial dual values must balance the time to train, the quality of the dual values and the computational time on a new instance. One extreme example is the LPR. It does not require any offline training. However it produces dual values with a relatively loose lower bound and the solution time grows

Table 3.5: 20% and 80% quantiles (0.2-q, 0.8-q) of computational time.

		1% optimality			0.5% optimality			0.25% optimality		
size	method	0.2-q	mean	0.8-q	0.2-q	mean	0.8-q	0.2-q	mean	0.8-q
200	network	3.7	4.6	5.5	3.8	9.0	15.2	14.9	**37.5**	49.3
	forest	3.6	**4.3**	5.1	3.7	**8.0**	12.4	14.7	42.0	49.9
	neighbour	3.8	4.7	5.6	4.0	10.0	15.4	15.6	43.1	46.6
	LPR	7.5	7.8	8.1	7.5	15.9	24.4	27.0	53.6	75.1
600	network	11.3	**13.4**	14.7	11.3	**20.5**	37.1	36.2	**48.4**	62.6
	forest	10.4	14.0	13.9	10.5	26.5	43.8	36.8	55.5	73.2
	neighbour	9.7	13.8	14.9	10.4	27.6	42.9	41.7	57.8	67.5
	LPR	26.2	28.7	29.2	26.2	43.0	57.8	60.9	86.9	115.1
1000	network	17.0	21.5	22.1	17.1	**34.3**	34.9	17.9	**74.2**	122.1
	forest	15.4	**19.5**	21.0	15.7	37.1	71.2	67.2	89.7	117.1
	neighbour	15.1	23.5	21.5	15.2	46.5	82.9	68.0	96.5	117.1
	LPR	42.8	45.6	48.6	42.8	67.4	93.1	73.8	120.4	159.4

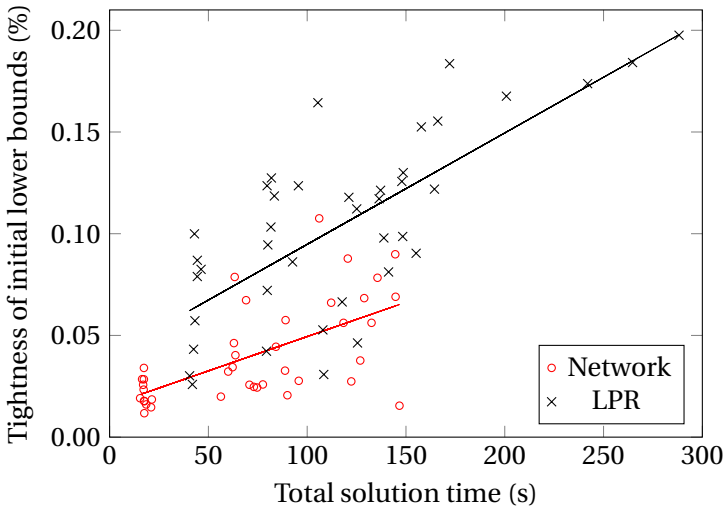

Figure 3.7: Tightness of the initial lower bound vs the total computational time required by the column generation procedure

Table 3.6: Breakdown of the average computational time (s)

tolerance	method	Initialisation	RMP	Subproblem	Primal Heuristic
1%	network	0.0	0.1	17.1	4.3
	LPR	29.2	0.1	11.8	4.5
0.5%	network	0.0	0.5	25.1	8.8
	LPR	29.2	0.8	25.2	12.2
0.25%	network	0.0	1.8	53.6	18.8
	LPR	29.2	3.8	61.4	26.1

as the problem size increases. The method based on a neural network needs off-line training, but it runs very quickly when solving a new instance and outputs dual values with a tight lower bound, resulting in significant speed up of the time to solve new instances.

Table 3.6 shows a breakdown of the average computational time of the column generation procedures based on the neural network and on the LPR for the 1000-generator instances for the 40 test cases. The column labelled as 'Initialisation' shows the time required to run the initialisation methods. In the LPR-based method, this is the time to solve the LPR, while the method based on a neural network this is the time to evaluate the neural network (but does not include the time to train the neural network). In cases of 1% tolerance, the time required to solve the LPR is longer than the sum of the time spent on the other routines. However as the tolerance becomes tighter, the number of iterations and the time spent in the other routines increases, and by 0.25% tolerance the LPR initialisation time is only 25% of the total time.

We note that all the methods find a solution of a pre-specified suboptimality (e.g. 0.1%) if they succeed. It is of interest to see whether there are qualitative differences in the solutions found by different methods. If there are many solutions whose suboptimalities are smaller than 0.1%, they may find solutions with different properties/trends. Such differences may have impacts on downstream decisions if the solver is used as a part of a larger model/system. However, analysing the quality of the solutions is outside the scope of this thesis.

Chapter 4

Primal Heuristics

Since Dantzig-Wolfe decomposition only provides lower bounds, it is necessary to combine with a primal heuristic to obtain a feasible solution. Typically primal heuristics are integrated with decomposition are used [46, 69, 29, 62]. Such primal heuristics gather information from the column generation procedure and use it as a guide to create primal feasible solutions. Empirically it has been observed that these methods usually find a primal feasible solution with small suboptimality. However, primal heuristics integrated with decomposition are not the only options in Dantzig-Wolfe decomposition. A decomposition-free primal heuristic may be advantageous since primal heuristics integrated with decomposition typically need a few iterations of the column generation procedure to collect enough data to create solutions of high quality. This requires some computational time, which may be avoided if a good feasible solution can be found earlier.

In recent years there has been a growing interest in the use of machine learning within primal heuristics. Bello et al. [4], Khalil et al. [35], Nazari et al. [49] and Kool et al. [39] apply reinforcement learning on various combinatorial problems, including the travelling salesman problems and the vehicle routine problems. Nair et al. [48] use reinforcement learning to solve two-stage stochastic programming approximately. In all of the above studies, machine learning models are designed to create primal feasible solutions directly. There are also approaches which use both machine learning and an optimisation solver. Bertsimas and Stellato [11] and Bertsimas and Stellato [12] propose a method for mixed-integer convex/quadratic

optimisation problems, where machine learning models are used to simplify the optimisation problems. They first train a model to predict the tight (active) constraints and the values of the integer variables with supervised learning. Given a new instance, the model is used to delete inactive constraints and fix all the integer variables and the reduced model is solved quickly. Xavier et al. [68] study the use of a support vector machine on a security-constrained UC problem where the model is used to reduce the problem size by fixing a subset of binary variables. They first train a separate model for each binary variable and then select all models whose accuracy in predicting their values are higher than a prescribed value. Then given a new instance the selected models are used to fix their corresponding variables and the reduced problem is then solved with an MILP solver. Pineda and Morales [56] work on the same problem but use a nearest neighbour approach. Their method is simple but achieves impressive results. Wang [67] proposes a similar approach based on a nearest neighbour method to reduce the problem size of a knapsack problem. Typically these approaches take a longer time compared to the methods which are purely based on machine learning models, however they are likely to provide solutions of better quality. We note that many of the above approaches focus on finding a primal feasible solution and cannot compute the suboptimality of the output.

In our study, we propose two primal heuristics: one integrated with decomposition and one based on machine learning. The primal heuristic integrated with decomposition uses the fractional solution to the RMP to fix a subset of integer variables. In each iteration of the column generation procedure, given the fractional solution to the RMP, the primal heuristic checks whether each binary variable satisfies the integrality constraint and fix those which do. The remaining variables are then optimised by an MILP solver with the aim of quickly finding a feasible, near-optimal solution to the original instance.

The other primal heuristic, based on machine learning, is of interest when the problems are to be solved repeatedly with different demand data but with the same problem structure. For example, this occurs when the UC problem is solved repeatedly on a daily basis by electricity generating companies. In such a case, typically the structure of the problem is the same and only some of the problem data such as demand is modified. Under this setup, the primal heuristic can use a pre-trained

model instead of the RMP to fix a subset of the integer variables. In the training phase, a model is trained to predict for any demand data and for each binary variable how likely it is that the variable takes each of its two possible values. After the training, given an instance to be solved, the prediction of the model is used with a rounding threshold to fix some binary variables and reduce the problem size. Similar approaches are used by Xavier et al. [68] and Wang [67]. However, in our approach we adjust the amount of problem size reduction adaptively depending on the solution progress so as to provide high quality solution quickly. Furthermore, we combine the primal heuristic with Dantzig-Wolfe decomposition, which allows us to find a primal feasible solution with proven suboptimality.

4.1 Local search and column combination heuristics

One of the earliest classes of primal heuristics studied in the literature is the feasibility recovery local search primal heuristic. First of them are Merlin and Sandrin [46], who solve UC by applying Lagrangian relaxation and using a subgradient method for the dual problem. In each iteration of the subgradient method, the pricing subproblems (2.14) are solved. A candidate primal solution is constructed by combining the pricing subproblem solutions obtained in this iteration and they then test if this is feasible for (2.1). If the generation capacity is not sufficient to meet the demand or the reserve at some time periods, they modify the dual step direction to increase the dual variables corresponding to the violated constraint to encourage more generators to be on at the shortage periods. We note that this modification affects the optimisation of the dual variables and the resulting lower bounds may be suboptimal.

Instead of modifying the dual step direction, Guan et al. [29] fix infeasibility of the primal solutions by applying local search. In each iteration, a primal solution is constructed from the pricing subproblem solutions. If this is infeasible, the cheapest available generators are committed until the demand and the reserve are met. After a feasible commitment decision is found, the amount of power output of each generator is optimised. That is, the values of the binary variables in the original problem are fixed to the values corresponding to the feasible commitment and the resulting LP is solved to compute the amount of power output. This is the variant

of the feasibility recovery local search primal heuristic used later in the numerical experiments in this paper.

Another heuristic integrated with decomposition is the column combination primal heuristic [65, 62, 61]. In this heuristic, the solutions to the pricing subproblems are stored in a pool. Then constraints are added to the problem (2.1) to restrict the choice of each pricing subproblem solution to one that is already in the pool: Let $\{x'_{gi} = (u'_{gi}, z'_{gi}) \in \mathbb{R}^n \times \{0,1\}^m \mid i \in J_g\}$ be the pool of solutions to pricing subproblem g. Then, binary variables w_{gi} ($g = 1,2,\ldots,G$, $i \in J_g$) are added to the original problem (2.1) together with the following constraints

$$\begin{aligned} z_g &= \sum_{i \in J_g} w_{gi} z'_{gi}, \qquad g = 1,2,\ldots,G, \\ \sum_{i \in J_g} w_{gi} &= 1, \qquad g = 1,2,\ldots,G, \\ w_{gi} &\in \{0,1\}, \qquad g = 1,2,\ldots,G, i \in J_g. \end{aligned} \tag{4.1}$$

This is referred to as the restricted master IP by Vanderbeck [65]. While this is still an MILP problem, the solution space is much smaller than the original problem and a standard MILP solver can be used to solve it. This method is used to solve the UC problem by Takriti and Birge [62] and to solve a stochastic version of the UC problems by Schulze et al. [61].

4.2 RMP partial-fixing primal heuristic

In this section we introduce a new primal heuristic which uses the RMP (2.13) to construct primal feasible solutions. In the following we assume that the RMP is feasible, which may be ensured by adding artificial columns. First we solve the RMP without regularisation and for each pricing subproblem g compute a weighted-average of columns

$$\hat{x}_g := \sum_{i \in \hat{I}_g} x_{gi} p_{gi}, \qquad g = 1,2,\ldots,G,$$

where $\hat{I}_g$ is the index set of columns $\{x_{gi} \mid i \in \hat{I}_g\}$ used to formulate the RMP for each g. Using the solution $\hat{x}_g$ for each pricing subproblem g, we construct a primal solution

$\hat{x}$. Although the integer variables in x_{gi} satisfy the integrality constraint for all g and $i \in \hat{I}_g$, those in $\hat{x}_g$ may not. In this primal heuristic, we check whether the elements of $\hat{x}_g$ that correspond to the binary decisions satisfy the integrality constraints and fix those which do. In this way we obtain a partially-fixed UC problem, which is then solved by an MILP solver. In each iteration of the column generation procedure, we repeat the above process. We refer this primal heuristic as *RMP partial-fixing primal heuristic*.

Figure 4.1 shows an example of the partial-fixing procedure. In the middle of the diagram an example solution to the RMP is shown. There are 4 variables which violates the integrality constraints. Those variables are not fixed and optimised by an MILP solver, while the other variables are fixed according to the value of the RMP solution.

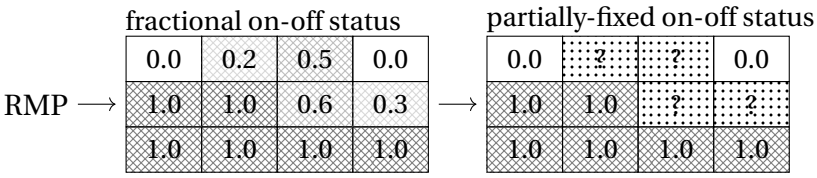

Figure 4.1: Skematic example of RMP partial-fixing on a 3-generator 4-period example

An important property of this primal heuristic is that if the RMP is solved by the Simplex Method (so yielding a basic solution) then the number of elements of $\hat{x}$ which violate the integrality constraint is bounded by a constant independent of the number of generators. Thus, the number of integer variables in the partially-fixed MILP does not grow with the number of generators, which is not true in some other primal heuristics. For example in the column combination primal heuristic the number of integer variables is proportional to the number of generators as a result the difficulty of the problem grows rapidly.

The above property can be shown using an argument similar to the one given by Bertsekas et al. [6]. Let C be the number of complicating constraints in (2.1), so $a \in \mathbb{R}^C$. If we use the formulation shown in Section 2.1, C is double the number of time periods. In the following we assume that $G > C$. We note that in the RMP (2.13), there are $G + C$ equality constraints. Thus, any basic solution to the RMP has at most

$G + C$ non-zero variables. The second constraint in (2.13) ensures that for each g at least one variable among $\{p_{gi}\}_{i \in \hat{I}_g}$ is positive. Therefore at most C generators have two or more positive values among $\{p_{gi}\}_{i \in \hat{I}_g}$ and all the other generators have exactly one positive value. In these cases, exactly one of $\{p_{gi}\}_{i \in \hat{I}_g}$ is equal to 1 and all of the others are equal to 0. Thus, $\hat{x}_g$ equals exactly same one of $\{x_{gi}\}_{i \in \hat{I}_g}$ and has integer values.

In practice the number of generators with multiple non-zeros in $\{p_{gi}\}_{i \in \hat{I}_g}$ may be smaller than C. Furthermore, even if multiple elements in $\{p_{gi}\}_{i \in \hat{I}_g}$ are positive, it may be that only a small part of $\hat{x}_g$ has fractional values.

We note that the RMP (2.13) is not necessarily feasible. Typically we need to run a few column generation iterations to gather enough columns to ensure the RMP feasible. Furthermore, even if the RMP is feasible, the partially-fixed MILP (2.1) is not necessarily feasible. However, in our experiments on UC instances with practical data (see Section 3.3), we do not observe instances where the partially-fixed schedule is infeasible and typically the above primal heuristic successfully finds a primal feasible solution with small suboptimality, such as 0.1%. On some instances, however, such tight suboptimality is not achieved. To handle such cases, we modify the method as follows. If the column generation procedure fails to terminate within a prescribed number of iterations (10 iterations in our implementation), we relax integer variables which correspond to the time periods adjacent to those with fractional values. For example, if the on-off status of generator g on time t is set free, we unfix the on-off status of the same generator in time $t-1$ and $t+1$. By relaxing more binary variables, the partially-fixed UC gets harder to solve but is more likely to provide a better solution.

Figure 4.2 shows an example of such enhancement using the same RMP solution as Figure 4.1. Although there are only 4 variables which violates the integrality constraints, we relax 3 additional variables which are 'adjacent' to the fractional variables.

We note that [24] (later extended in [20]) study primal heuristics similar to the one described here in the sense that they use the solution to the regularised RMP. Fractional solutions obtained from the regularised RMP also have a property that only a bounded number of variables can be fractional given that the specialised ac-

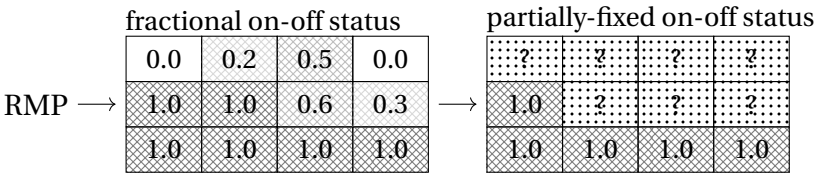

Figure 4.2: Skematic example of RMP partial-fixing with additional relaxation

tive set method studied in [37, 37] is used to solve the RMP. Our method works with the standard Simplex method and hence can be implemented easily. Furthermore, they interpret the fractional solution as a probability and apply simple rounding heuristics to obtain an integer feasible solution. We fix the fractional values using an optimiser and thus it returns a solution no worse than the one returned by their primal heuristic (up to suboptimality tolerance of the optimiser). However, since the results reported in their paper are based on different UC instances, it is not straightforward to compare the performances in terms of solution quality nor computational time. Extensive numerical experiments comparing the performances of primal heuristics are outside the scope of this thesis.

4.3 Examples

Some primal heuristics described above require more computational time than the others. In this section we consider small UC instances and compare their behaviours. To simplify the exposition we do not consider the marginal and startup costs ($C^{\mathrm{mr}} = C^{\mathrm{up}} = 0$), the reserve constraint ($P^{\mathrm{r}} = 0$), the minimum generation limit constraint ($P^{\mathrm{min}} = 0$), the ramp rate bounds constraint ($P^{\mathrm{ru}}, P^{\mathrm{rd}}, P^{\mathrm{su}}, P^{\mathrm{sd}} \gg 0$) and the minimum uptime constraint ($T^{\mathrm{u}} = 0$). We assume that the initial states of the generators are all off ($\alpha_{gt} = p_{gt} = 0$ for $t = 0$ and $g = 1, 2, \ldots, G$).

Example 4.A The first example consists of 2 generators and 1 time period. The data of the generators is given in Table 4.1. This is the same as that of Example 2.A in Section 2.7.

Figure 4.3 shows X in the space of the total cost c^{sum} and the total power output p^{sum}. Each of the solid lines and the origin corresponds to a single feasible generator

Table 4.1: Generators in Example 4.A

generator g	1	2
no-load cost C_g^{nl}	1	2
maximum generation limit $P_g^{\max}$	1	3
minimum downtime T_g^{d}	1	1

commitment decision, which is shown aside. The optimal primal and dual objective values are plotted as functions of the total power output in the same figure.

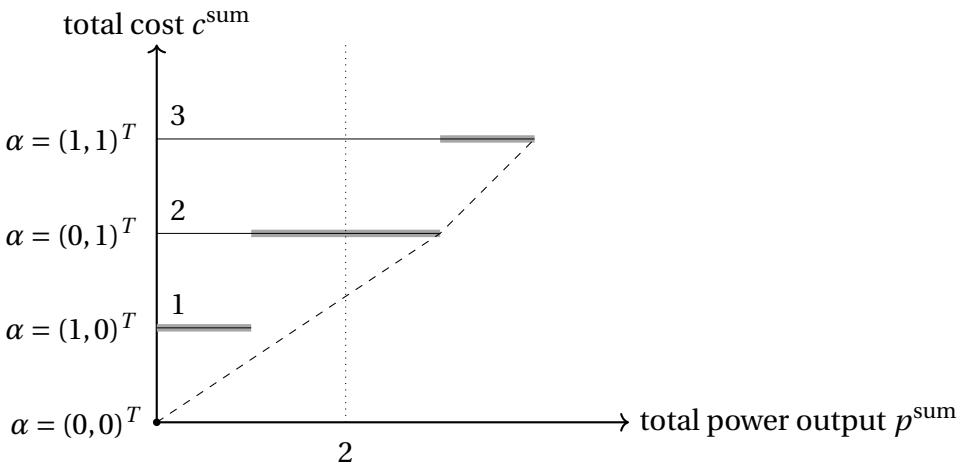

Figure 4.3: X (—, •), optimal primal (▬) and dual (- - -) objective values (α: commitment decision)

In Section 2.7 we obtained the dual function explicitly. The dual function with $P^{\mathrm{d}} = 2$ is plotted in Figure 4.4. The optimal dual bound is attained at $y = 2/3$.

In the following we apply the column generation procedure and the primal heuristics to the dual function $q(2, y)$. We assume that if there are multiple optimal solutions to a pricing subproblem, the solver returns the first solution in terms of the lexicographical order. For example, if $(1,0)^T$ and $(0,1)^T$ are the optimal solutions, the solver returns $(0,1)^T$. We also assume that the feasibility recovery local search primal heuristic commits the generators with the smallest non-load costs C^{nl}.

The output of the column generation procedure with the initial point $(0,0)^T$ and the constant step size $t_k = 1$ for all k is shown in Table 4.2. The column labelled as 'y' shows the values of the dual variables in each iteration. The next column shows

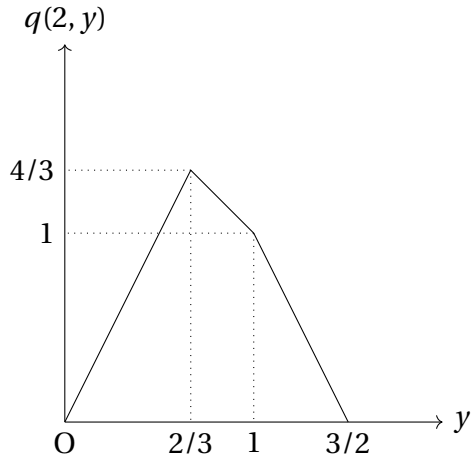

Figure 4.4: Dual function with demand $P^{\mathrm{d}} = 2$

the lower bounds obtained by evaluating the dual function at the given dual values, followed by a column showing the solution to the pricing subproblems. The next two columns are the output of the feasibility recovery local search primal heuristic. The left column shows the feasible schedule found by the primal heuristic, while the right column shows its cost. The next two columns are the output of the column combination primal heuristic displayed in the same manner. The last three columns are the output of the RMP partial-fixing primal heuristic. The left-most column is the solution to the RMP (i.e. a fractional schedule), the middle column is the solution obtained by solving the partially fixed UC and the right-most column is the upper bound computed from the feasible solution found. To reduce clutter only the values of the commitment decisions α are displayed.

In this small example the column generation procedure finds the optimal dual solution in 3 iterations. We note that in the third iteration both of $(0,0)^T$ and $(0,1)^T$ are the optimal solutions to the pricing subproblems. Since we assume the solver returns the first one in terms of the lexicographical order, we get $(0,0)^T$.

When the pricing subproblem solution is $(0,0)^T$ (that is, all the generators are shut down), there is no way to meet the demand. The feasibility recovery local search primal heuristic fixes the shortage by committing generator 1 first, since it has smaller non-load cost than generator 2. However, the capacity of generator 1 is 1 and it is still not sufficient to meet the demand. Thus, generator 2 is committed additionally and

we obtain $(1,1)^T$ as a feasible schedule, which provides upper bound 3.

In the first iteration given the single pricing subproblem solution $(0,0)^T$, the column combination primal heuristic fails to output any feasible solution. In the second iteration given the pricing subproblem solution $(1,1)^T$, all the four generator schedules satisfy the column combination constraint (4.1):

$$\left\{ \begin{pmatrix} w_{11}\cdot 0 + w_{12}\cdot 1 \\ w_{21}\cdot 0 + w_{22}\cdot 1 \end{pmatrix} \middle| (w_{g1}, w_{g2}) \in \Delta^2 \ (g = 1,2) \right\} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\},$$

where

$$\Delta^n = \{w \in \{0,1\}^n | w_1 + w_2 + \cdots + w_n = 1\}.$$

Thus the column combination primal heuristic outputs the optimal primal solution $(0,1)^T$.

The RMP partial-fixing similarly fails to find a feasible solution in the first iteration. In the second iteration and later the solution to the RMP is $(0,2/3)^T$. The RMP partial-fixing compares two solutions, $(0,0)^T$ and $(0,1)^T$, and outputs the optimal one.

Although the column combination primal heuristic and the RMP partial-fixing primal heuristic returns the same solution, the column combination primal heuristic needs to solve a larger MILP to obtain the primal feasible solution. Ignoring the variables γ, η, the problem to combine columns has 6 binary variables in total: 2 binary variables for commitment decisions α and 4 binary variables for column combination w. One of the integrality constraints on α and w can be relaxed, but at least 2 binary variables are required. On the other hand, the RMP partial-fixing primal heuristic solves a UC problem where the first generator is fixed to be off and only the second generator is to be scheduled. Thus the partially-fixed MILP has only 1 binary variables.

Example 4.B We next consider a problem with 2 generators and 2 time periods. The problem data is given in Table 4.3 and Table 4.4.

Table 4.2: Output of the column generation procedure on Example 4.A

				local search		col. comb.		RMP partial-fixing		
it	y	lb	subproblem	sol.	ub	sol.	ub	RMP	sol.	ub
1	0	0	$\begin{pmatrix}0\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	3	-	-	-	-	-
2	2	-1	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	3	$\begin{pmatrix}0\\1\end{pmatrix}$	2	$\begin{pmatrix}0\\2/3\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	2
3	2/3	4/3	$\begin{pmatrix}0\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	3	$\begin{pmatrix}0\\1\end{pmatrix}$	2	$\begin{pmatrix}0\\2/3\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	2

Table 4.3: Generators in Example 4.B

generator g	1	2
no-load cost C_g^{nl}	1	2
maximum generation limit $P_g^{\max}$	1	1
minimum downtime T_g^{d}	1	1

Table 4.4: Demand in Example 4.B

time period	1	2
demand	3/2	3/2
reserve	0	0

The only feasible, and hence optimal, generator schedule is to commit the both generators for all the time periods, which gives the optimal objective value of 6. The contour plot of the dual function with $P_1^{\mathrm{d}} = P_2^{\mathrm{d}} = 3/2$ is plotted in Figure 4.5. The plot shows that the maximum of the dual function is 4, which is attained at $y = (2,2)^T$.

The result of the column generation procedure and the primal heuristics are shown in Table 4.5. The initial point is set to $(5/2, 3/2)^T$ and the constant step size of 1 is used. The column generation procedure finds the optimal dual values in 3 iterations and output the same dual values and pricing subproblem solutions afterwards. For any iteration the feasibility recovery local search primal heuristic commit all the generators to find a feasible schedule and obtain the optimal schedule.

After 3 iterations, combination of the pricing subproblem solutions (4.1) yields the following three generator schedules

$$\left\{ \begin{pmatrix} (1\ 1) \\ w_1(1\ 0) + w_2(0\ 1) + w_3(0\ 0) \end{pmatrix} \middle| (w_1, w_2, w_3) \in \Delta^3 \right\} = \left\{ \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \right\}.$$

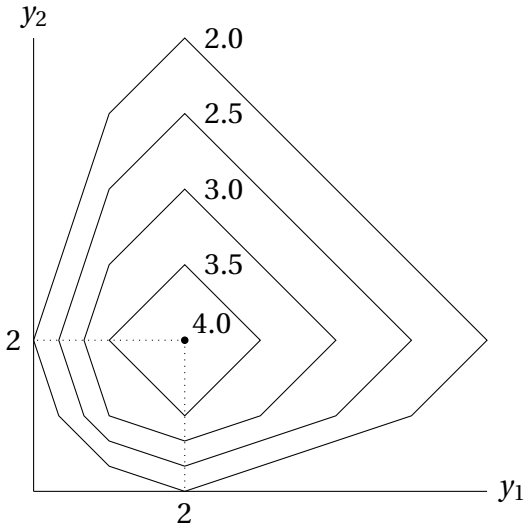

Figure 4.5: Contour plot of the dual function of Example 4.B

The column combination primal heuristic fails to find a feasible solution since any combination does not satisfy the demand. Interestingly, the RMP partial-fixing primal heuristic is able to find a fractional feasible schedule

$$\alpha = \begin{pmatrix} 1 & 1 \\ 1/2 & 1/2 \end{pmatrix} = \begin{pmatrix} (1\ 1) \\ 1/2 \cdot (1\ 0) + 1/2 \cdot (0\ 1) + 0 \cdot (0\ 0) \end{pmatrix}$$

By committing generator 2 for all the time periods the RMP partial-fixing primal heuristic finds the optimal schedule.

Table 4.5: Output of the column generation procedure on Example 4.B with initial point $y = (5/2, 3/2)^T$

				local search		col. comb.		RMP partial-fixing		
it	y	lb	subproblem	sol.	ub	sol.	ub	RMP	sol.	ub
1	$\begin{pmatrix} 5/2 \\ 3/2 \end{pmatrix}$	7/2	$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	6	-	-	-	-	-
2	$\begin{pmatrix} 3/2 \\ 5/2 \end{pmatrix}$	7/2	$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	6	-	-	$\begin{pmatrix} 1 & 1 \\ 1/2 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	6
3	$\begin{pmatrix} 2 \\ 2 \end{pmatrix}$	4	$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	6	-	-	$\begin{pmatrix} 1 & 1 \\ 1/2 & 1/2 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	6

The behaviour of the primal heuristics depends on the initial point of the column

generation procedure. The output of the column generation procedure and the primal heuristics with the initial point at $y = (2.5, 2.5)^T$ and the constant step size 1 are shown in Table 4.6. The column generation procedure again takes 3 iterations to optimise the dual function. However, in this setting the pricing subproblem solutions in the first iteration is optimal to the original problem and all of the primal heuristics find the optimal solution in 1 iteration.

Table 4.6: Output of the column generation procedure on Example 4.B with initial point $y = (5/2, 5/2)^T$

				local search		col. comb.		RMP partial-fixing		
it	y	lb	subproblem	sol.	ub	sol.	ub	RMP	sol.	ub
1	$\begin{pmatrix}5/2\\5/2\end{pmatrix}$	7/2	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6
2	$\begin{pmatrix}3/2\\3/2\end{pmatrix}$	7/2	$\begin{pmatrix}1&1\\0&0\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1/2&1/2\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6
3	$\begin{pmatrix}2\\2\end{pmatrix}$	4	$\begin{pmatrix}1&1\\0&0\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6	$\begin{pmatrix}1&1\\1/2&1/2\end{pmatrix}$	$\begin{pmatrix}1&1\\1&1\end{pmatrix}$	6

Example 4.C In the previous two examples the RMP partial-fixing primal heuristic outperformed the other primal heuristics. In this example we explore a case when it fails to find a better solution. We consider a single-period 3-generator UC instance as shown in Table 4.7. X and the optimal primal and dual objective values are plotted in the cost-power space in Figure 4.6. When demand is 3, the optimal solution is (0, 1, 0) with the optimal objective value 24. Using the similar argument as before, we obtain the dual function, which is shown in Figure 4.7.

Table 4.7: Generators in Example 4.C

generator g	1	2	3
no-load cost C_g^{nl}	12	24	42
maximum generation limit $P_g^{\max}$	2	3	6
minimum downtime T_g^{d}	1	1	1

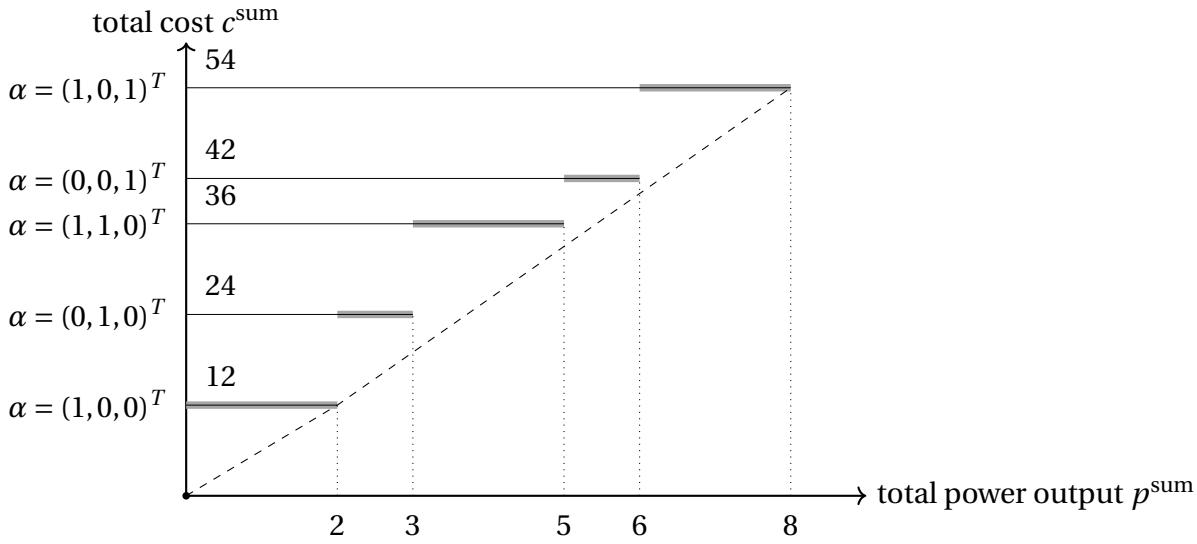

Figure 4.6: X (—, •), optimal primal (▬) and dual (- - -) objective values

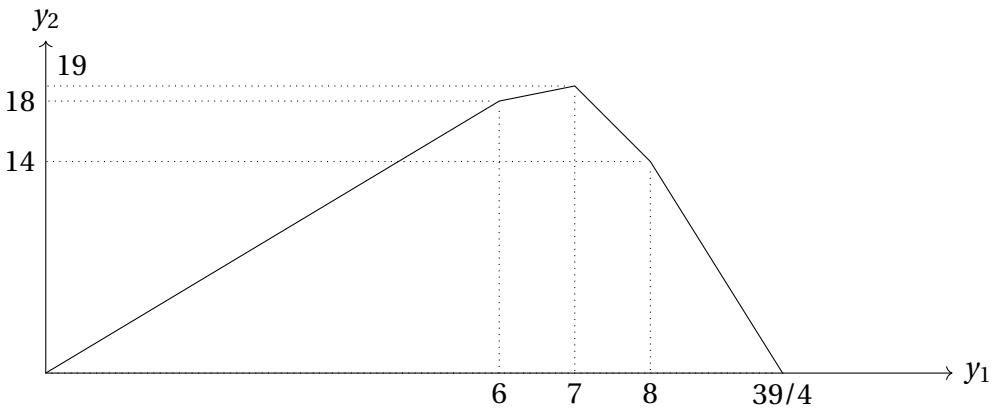

Figure 4.7: The dual function of Example 4.C with demand $P^{\text{d}} = 3$

Assume that the initial dual value is 3 and the step size is 3/2. The output of the column generation procedure is shown in Table 4.8. The column generation procedure finds the optimal dual value after 3 iterations.

Table 4.8: Output of the column generation procedure on Example 4.C

				local search		col. comb.		RMP partial-fixing		
it	y	lb	subproblem	sol.	ub	sol.	ub	RMP	sol.	ub
1	3	9	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\\0\end{pmatrix}$	36	-	-	-	-	-
2	15/2	16	$\begin{pmatrix}1\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\\1\end{pmatrix}$	54	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	42	$\begin{pmatrix}1\\0\\1/6\end{pmatrix}$	$\begin{pmatrix}1\\0\\1\end{pmatrix}$	54
3	7	19	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\\0\end{pmatrix}$	36	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	42	$\begin{pmatrix}1\\0\\1/6\end{pmatrix}$	$\begin{pmatrix}1\\0\\1\end{pmatrix}$	54

In the first iteration, the pricing subproblem solution is $(0,0,0)^T$. The feasibility recovery local search primal heuristics commits generator 1 and 2 and finds a feasible solution $(1,1,0)^T$, which gives the best upper bound 36.

On the other hand, after 3 iterations, the column combination primal heuristic gathers pricing subproblem solutions $(0,0,0)^T$, $(1,0,1)^T$ and $(1,0,0)^T$. Therefore the possible combinations of the columns are

$$\left\{ \begin{pmatrix} w_{11}\cdot 0 + w_{12}\cdot 1 \\ 0 \\ w_{31}\cdot 0 + w_{32}\cdot 1 \end{pmatrix} \middle| \, (w_{g1}, w_{g2}) \in \Delta^2 \ (g = 1,3) \right\} = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix} \right\}$$

and the column combination primal heuristic outputs $(0,0,1)^T$ as the best primal feasible solution with upper bound 42.

After the second iteration, the solution to the RMP is $(1,0,1/6)^T$. The RMP partial-fixing has two possible solutions $(1,0,0)^T$ and $(1,0,1)^T$ and it outputs $(1,0,1)^T$ as it is the only feasible solution.

The difference of the performances of the primal heuristics is due to the criteria to compare generators. The feasibility recovery local search compares generators

with the non-load cost and thus generator 2 is cheaper than generator 3. However, the pricing subproblems and the RMP tend to use generators with small average cost at their maximum power output $C_g^{\mathrm{nl}}/P_g^{\max}$. Thus, generator 3 is more likely to be used than generator 2. In fact, generator 2 never appears in the pricing subproblem solutions nor the RMP solutions.

Example 4.D In this last example we study another mode of failure of the RMP partial-fixing primal heuristic. We consider a UC instance with 6 time periods and 2 generators, whose data is given in Table 4.9 and Table 4.10. To meed the demand in time period 1, 3, 4 and 6, both of the generators must be on, while time period 2 and 5 only requires one of them. Since generator 2 cannot be switched off for 1 time period, the optimal solution is to keep generator 1 on at time period 1, 3, 4, 6 and keep generator 2 up for all the time periods

$$\alpha = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Table 4.9: Generators in Example 4.D

generator g	1	2
no-load cost C_g^{nl}	1	2
maximum generation limit $P_g^{\max}$	1	1
minimum downtime T_g^{d}	1	2

Table 4.10: Demand data of Example 4.D

time period	1	2	3	4	5	6
demand	3/2	1	3/2	3/2	1	3/2
reserve	0	0	0	0	0	0

Table 4.11 is the output of the column generation procedure used with the initial point

$$y = \begin{pmatrix} 5/2 & 3/2 & 3/2 & 5/2 & 3/2 & 3/2 \end{pmatrix}^T$$

and constant step size $t = 1$.

Table 4.11: Output of the column generation procedure on Example 4.D

				local search		col. comb.		RMP partial-fixing		
it	y	lb	subproblem	sol.	ub	sol.	ub	RMP	sol.	ub
1	$\begin{pmatrix}5/2\\3/2\\3/2\\5/2\\3/2\\3/2\end{pmatrix}$	9	$\begin{pmatrix}1&1\\1&0\\1&0\\1&1\\1&0\\1&0\end{pmatrix}^T$	$\begin{pmatrix}1&1\\1&1\\1&1\\1&1\\1&1\\1&1\end{pmatrix}^T$	18	-	-	-	-	-
2	$\begin{pmatrix}3/2\\3/2\\5/2\\3/2\\3/2\\5/2\end{pmatrix}$	9	$\begin{pmatrix}1&0\\1&0\\1&1\\1&0\\1&0\\1&1\end{pmatrix}^T$	$\begin{pmatrix}1&1\\1&1\\1&1\\1&1\\1&1\\1&1\end{pmatrix}^T$	18	-	-	$\begin{pmatrix}1&1/2\\1&0\\1&1/2\\1&1/2\\1&0\\1&1/2\end{pmatrix}^T$	-	-
3	$\begin{pmatrix}2\\3/2\\2\\2\\3/2\\2\end{pmatrix}$	10	$\begin{pmatrix}1&0\\1&0\\1&0\\1&0\\1&0\\1&0\end{pmatrix}^T$	$\begin{pmatrix}1&1\\1&1\\1&1\\1&1\\1&1\\1&1\end{pmatrix}^T$	18	-	-	$\begin{pmatrix}1&1/2\\1&0\\1&1/2\\1&1/2\\1&0\\1&1/2\end{pmatrix}^T$	-	-

In the pricing subproblem solutions in any iterations, generator 1 is always committed but generator 2 is not. The feasibility recovery local search primal heuristic finds a primal feasible solution by committing generator 2 for all the time periods. The column combination primal heuristics fails to find any primal feasible solutions even after 3 iterations since the only possible combinations are the following three which are all infeasible:

$$\left\{ \begin{pmatrix} (1\,1\,1\,1\,1\,1) \\ w_1(1\,0\,0\,1\,0\,0) + w_2(0\,0\,1\,0\,0\,1) + w_3(0\,0\,0\,0\,0\,0) \end{pmatrix} \middle| (w_1, w_2, w_3) \in \Delta^3 \right\}$$
$$= \left\{ \begin{pmatrix}1&1&1&1&1&1\\0&0&0&0&0&0\end{pmatrix}, \begin{pmatrix}1&1&1&1&1&1\\1&0&0&1&0&0\end{pmatrix}, \begin{pmatrix}1&1&1&1&1&1\\0&0&1&0&0&1\end{pmatrix} \right\}.$$

The unregularised RMP is feasible after 2 iterations, but the RMP partial-fixing primal heuristic fails to find a primal feasible solution. This is because generator 2 is fixed to

be off on time period 2 and 5.

4.4 Neural network partial-fixing primal heuristic

All of the primal heuristics discussed above use information gathered through the execution of the column generation procedure or the subgradient methods. In this section we consider two primal heuristics based on machine learning. We assume that the problem (2.1) is to be solved repeatedly with different demand data ω.

In the training phase, we sample ω and solve as many training instances as possible. They may come from historical data or in our case from historical data with perturbation. In this way, we obtain the dataset for each sample ω and the corresponding optimal values of the binary variables. Then, we use the dataset to create a prediction model which takes ω as input and predicts the value of the binary variables. The two alternatives we consider are a neural network and a nearest neighbour model.

The neural network is a feed-forward neural network [13]. The model takes ω as input and outputs a vector each of whose elements is a predicted probability of the corresponding binary variables being 1. The neural network is trained as a standard binary classification problem.

The nearest neighbour model compares the new instance with problem parameter ω to those in the training dataset and find a prescribed number of the closest neighbours by the Euclidean distance. The average of the values of the binary variables in their solutions of these nearest neighbours are computed and used as the prediction of their probabilities.

The output of a prediction model can be of help when constructing a feasible solution. The simplest approach is to round the prediction to the nearest integer. However, such a solution is usually infeasible when the problem is highly constrained. Instead, as described below, we use the prediction to fix only a subset of binary variables so that the problem size is reduced. Similar ideas have been explored by Xavier et al. [68], Pineda and Morales [56], and Wang [67].

Pick a threshold value $\alpha \in (0.5, 1]$. If the predicted probability of a binary variable being 1 is larger than α (smaller than $1-\alpha$), fix the corresponding binary variables to

1 (0), and leave all the other variables unfixed. Then solve the resulting partially-fixed MILP with an MILP solver.

Choosing a suitable threshold value is a subtle task. If we choose α close to 0.5 we will fix many binary variables the problem will be small and can be solved quickly. However, fixing too many variables may result in infeasibility or unacceptably large suboptimality. On the other hand, fixing fewer variables results in a harder problem which takes longer to solve. Instead of fixing the threshold value to a single value a priori, we use a sequence of increasing values (0.8, 0.9, 0.95, 0.99 and 1.0 are used in our implementation). First we try a small threshold value and solve the partially-fixed MILP. If the resulting problem is infeasible, or if the resulting problem is solved, we try a larger threshold value.

When using Dantzig-Wolfe decomposition, we can combine the primal heuristics based on machine learning with the column generation procedure as follows. At the end of each iteration of the column generation procedure, we run one of the above primal heuristics and solve partially-fixed MILPs using an optimisation solver. We impose a limit on the amount of time spent by the primal heuristic in each column generation iteration. If the solver has not found a solution within the target tolerance within this time limit, it is halted and the next iteration of the column generation procedure is executed. In the next iteration the lower bound may have improved enough to reach the tolerance gap. Otherwise, in the next run of the primal heuristic, the solver is resumed from where it was interrupted in the previous iteration, essentially splitting the execution of the solver over several column generation iterations. We refer to the method based on neural network and nearest neighbour as *neural network partial-fixing primal heuristic* and *nearest neighbour partial-fixing primal heuristic* respectively.

4.5 Numerical experiments

In this section the proposed primal heuristics are evaluated. We use the same environment and test instances as described in Section 3.3.

4.5.1 Time to close gap with Dantzig-Wolfe decomposition

Experimental Setup

In the first experiment we use the primal heuristics alongside Dantzig-Wolfe decomposition and measure computational time to find a primal feasible solution of a specified suboptimality.

To prepare the dataset used by the neural network and nearest neighbour partial-fixing, as many training instances as possible are solved to 0.25% optimality within a prescribed dataset preparation time budget. To solve each training instance we use Dantzig-Wolfe decomposition with the feasibility recovery local search primal heuristic.

After the dataset is constructed, a neural network is trained to predict the values of the binary variables. We use a feed-forward neural network with 2 hidden-layers of 400 units per layer with ReLU activation function.

The nearest neighbour method uses the same dataset as the neural network. When solving a test instance, the parameter of the instance is compared with those of training instances, and the 50 closest instances by the Euclidean distance are chosen to compute the average values of the binary variables.

To solve a test instances, Dantzig-Wolfe decomposition is used and the primal heuristics are run in the end of each column generation iteration. When the iteration number is large, the lower bound is typically tight and the upper bound provided by the primal heuristics is loose unless more time is allocated to it. Thus, we allow the primal heuristics to use more time in later iterations according to the formula

$$\text{(primal heuristic time limit)} = \text{(time spent to solve pricing subproblems)} \cdot \left(\frac{k}{10} + 2\right), \tag{4.2}$$

where k is the iteration number. We note that some primal heuristics may stop before reaching the time limit. For example, the partially-fixed MILP solved in the RMP partial-fixing primal heuristic is often solved before the time limit is reached.

For comparison, we also run CPLEX on the original MILP problem without decomposition.

Table 4.12: Statistics on the training

	200	600	1000
number of training instance	15,419	7,241	4,886
time to train a model (s)	757	1,514	1,837

Results

The training dataset was constructed with the dataset preparation time budget of 24 hours with 8 cores. Table 4.12 reports the number of solved instances as well as the time used to train a neural network.

To evaluate the primal heuristics, 40 test instances were solved. The demand data to construct the test instances was sampled from a different year than those of the instances used to train the neural network and the nearest neighbour model. Table 4.13 shows the number of instances solved within the time limit of 10 minutes, the average computational time and the average number of column generation iterations to solve the instances or to reach the time limit. For the instances that are not solved within the time limit, the time is set to be 10 minutes and the number of iterations reached by the 10 minute time limit is used.

When the tolerance is loose, such as 0.5% or 0.25%, the neural network partial-fixing primal heuristic usually performs best. We observe that the number of column generation iterations in these cases is very small and with averages all less than 2. That is, on typical instances the neural network partial-fixing very quickly find a primal feasible solution satisfying the suboptimality tolerance with 0.25% and Dantzig-Wolfe decomposition give a lower bound to assert that the suboptimality was smaller than 0.25%. The nearest neighbour partial-fixing primal heuristic is second best in half of the cases. However, for all the cases, the average performance of the neural network partial-fixing primal heuristic is better than that of the nearest neighbour partial-fixing primal heuristic, in terms of the computational time and the number of column generation iterations. The other primal heuristics are integrated with decomposition and require more column generation iterations to find primal feasible solutions of acceptable suboptimality. This results in longer computational time for many instances.

Table 4.13: Number of problems solved, computational time (s) and iterations

		tol: 0.5%			0.25%			0.1%		
size	method	solved	time	iter	solved	time	iter	solved	time	iter
200	feasibility recovery	40	15.9	3.0	40	56.0	11.3	4	551.8	68.6
	column combination	40	32.8	7.1	40	33.6	7.2	30	276.3	35.6
	RMP partial-fixing	40	22.8	5.5	40	24.0	5.8	40	**37.5**	8.4
	network partial-fixing	40	**9.3**	1.0	40	**12.0**	1.3	40	56.3	5.2
	nearest partial-fixing	40	11.6	1.2	40	15.5	1.6	40	74.4	6.3
	CPLEX	40	215.8	0.0	37	336.0	0.0	18	558.0	0.0
600	feasibility recovery	40	42.2	2.0	40	84.4	5.4	30	263.5	16.8
	column combination	40	83.9	5.2	40	86.1	5.3	40	105.6	6.4
	RMP partial-fixing	40	60.7	4.4	40	62.8	4.6	40	**76.8**	5.7
	network partial-fixing	40	**39.6**	1.3	40	**41.7**	1.3	40	128.0	4.2
	nearest partial-fixing	40	51.8	1.6	40	60.2	1.8	40	136.0	4.3
	CPLEX	5	589.7	0.0	5	589.7	0.0	1	593.5	0.0
1000	feasibility recovery	40	**66.6**	1.8	40	117.9	4.2	37	236.1	9.4
	column combination	40	114.1	3.9	40	120.1	4.1	40	166.4	5.6
	RMP partial-fixing	40	85.5	3.8	40	91.4	4.1	40	**113.7**	5.2
	network partial-fixing	40	70.8	1.3	40	**75.1**	1.4	40	181.6	3.9
	nearest partial-fixing	40	91.6	1.8	40	94.3	1.9	40	208.8	4.3
	CPLEX	1	597.4	0.0	1	597.4	0.0	0	600.0	0.0

If the target tolerance is tight (0.1%), the results are different. The RMP partial-fixing primal heuristic outperforms the other methods in all the cases. The neural network partial-fixing primal heuristic requires a smaller number of column generation iterations but needs longer computational time. This is because the RMP partial-fixing primal heuristic does not use all the time allocated to the primal heuristics but the neural network partial-fixing primal heuristic always run as long as the time limit. The neural network and nearest neighbour partial-fixing primal heuristics found a primal solution with suboptimality smaller than 0.1% for all the test instances. However, the neural network partial-fixing primal heuristic is on average slower than the RMP partial-fixing in the all cases and the nearest neighbour partial-fixing primal heuristic is even slower. The column combination primal heuristic fails to find a primal solution for some test instances for the 200-generator case. However, it successfully find a primal solution on all the test instances of size 600 and 1,000 and the average computational time is faster than the neural network partial-fixing primal heuristic. The feasibility recovery local search primal heuristic also has better performance for larger test instances but is still inferior to the other primal heuristics.

Table 4.14 shows a breakdown of the average computational time of the proposed methods applied to the 1000-generator test cases. The format is the same as Table 3.6. The column labelled as primal heuristic shows the average time spent by the primal heuristics. For RMP partial-fixing primal heuristic it includes the time to solve the unregularised RMP and the partially fixed UC problem. Since all the methods use the same initialisation of the dual variables (LPR), the time spent in initialisation is almost the same. RMP partial-fixing primal heuristic spends most of the time solving the subproblems. In other words, it is very quick to solve the partially fixed UC problem in RMP partial-fixing primal heuristic. This indicates that the number of fractional values in the solution of the unregularised RMP is very small. On the other hand the other two primal heuristics spend considerable amounts of time to solve the partially fixed UC problems. This supports the earlier observation: these two primal heuristics require fewer iterations but at each iteration they spend significant amounts of time in the primal heuristics.

Table 4.14: Breakdown of the average computational time (s)

tol	method	Initialisation	RMP	Subproblem	Primal Heuristic
0.5%	RMP partial-fixing	29.6	2.0	50.6	3.4
	network partial-fixing	29.7	0.3	17.0	36.1
	nearest partial-fixing	29.5	0.4	23.2	51.4
0.25%	RMP partial-fixing	29.6	2.3	55.7	4.0
	network partial-fixing	29.7	0.3	18.8	38.1
	nearest partial-fixing	29.5	0.5	24.7	52.7
0.1%	RMP partial-fixing	29.6	3.7	72.7	7.7
	network partial-fixing	29.7	1.4	53.7	96.9
	nearest partial-fixing	29.5	1.8	59.3	118.2

Analysis of the effect of dataset preparation time budget

In this section we study the effect of the dataset preparation time budget. We consider cases where the time budget is 6, 12, 36 or 48 hours instead of 24 hours and observe how the performance of the methods is affected. To this end, the neural network is trained using the dataset generated within each of these time budgets. The performance of the models is then evaluated as before and the results are reported in Table 4.15. In all cases, all of the test instances are solved to within 0.1% tolerance.

In many cases, both the neural network and the nearest neighbour model tend to perform better with a larger training dataset. Comparing the neural networks with 6-hour training, those with 48-hour training are all on average faster. However, there is not a systematic improvement in the performance beyond 24-hour of training. The room for additional performance gain seems limited if further dataset preparation time budget is given. The result of the nearest neighbour model are similar.

Analysis of neural network architecture

In the following the effect of the model architecture is studied. In the previous experiments, small feed-forward neural networks with 2 hidden-layers of 400 units per layer were considered. Here, we additionally train deeper neural networks and measure their performances. A deeper model consists of 4 hidden layers of 1000 units per layer. We use the same training dataset which is generated with the dataset preparation time budget of 24 hours. The performance of the models are evaluated similarly and the results are reported in Table 4.16. The difference in performance is

Table 4.15: Performance of the primal heuristics with different dataset preparation time budgets

			tol: 0.5%			0.25%			0.1%		
size	method	budget	solved	time	iter	solved	time	iter	solved	time	iter
200	network	6	40	7.7	1.1	40	9.4	1.7	40	30.5	6.8
		12	40	7.8	1.1	40	8.9	1.5	40	29.5	6.6
		24	40	7.5	1.1	40	9.3	1.6	40	27.8	6.3
		36	40	7.5	1.1	40	8.6	1.4	40	25.4	6.0
		48	40	7.3	1.0	40	8.4	1.4	40	26.3	6.3
	neighbour	6	40	8.0	1.2	40	10.2	1.9	40	41.9	8.6
		12	40	7.7	1.1	40	10.7	2.1	40	40.8	8.6
		24	40	7.6	1.1	40	9.8	1.8	40	38.2	8.2
		36	40	7.6	1.1	40	9.3	1.6	40	34.0	7.6
		48	40	7.9	1.2	40	10.2	1.9	40	31.4	7.3
600	network	6	40	31.2	1.5	40	33.7	1.8	40	71.2	5.3
		12	40	27.8	1.2	40	32.1	1.5	40	65.8	4.8
		24	40	27.5	1.1	40	30.3	1.4	40	64.6	4.7
		36	40	29.7	1.4	40	30.6	1.4	40	64.0	4.7
		48	40	30.2	1.4	40	30.2	1.4	40	64.5	4.7
	neighbour	6	40	33.6	1.8	40	37.4	2.1	40	83.7	6.0
		12	40	32.7	1.6	40	36.8	2.0	40	75.4	5.6
		24	40	32.0	1.5	40	35.0	1.9	40	76.1	5.5
		36	40	32.4	1.6	40	35.3	1.9	40	71.7	5.4
		48	40	34.2	1.8	40	37.3	2.1	40	73.8	5.6
1000	network	6	40	50.3	1.5	40	53.9	1.7	40	105.7	5.0
		12	40	50.5	1.4	40	53.2	1.6	40	97.9	4.5
		24	40	44.6	1.2	40	45.1	1.2	40	89.7	4.2
		36	40	45.5	1.2	40	45.9	1.3	40	91.3	4.3
		48	40	47.2	1.4	40	48.4	1.4	40	89.5	4.2
	neighbour	6	40	56.8	1.9	40	63.3	2.4	40	143.5	6.6
		12	40	55.2	1.9	40	59.7	2.1	40	122.4	5.7
		24	40	62.5	2.1	40	69.5	2.5	40	132.6	6.0
		36	40	47.8	1.4	40	52.4	1.6	40	110.7	5.1
		48	40	52.1	1.6	40	57.3	1.9	40	112.9	5.2

Table 4.16: Performance of the original and deeper neural networks

		tol: 0.5%			0.25%			0.1%		
size	model	solved	time	iter	solved	time	iter	solved	time	iter
200	original	40	7.5	1.0	40	9.3	1.6	40	27.8	6.4
	deeper	40	7.4	1.0	40	8.6	1.4	40	28.5	6.6
600	original	40	27.5	1.2	40	30.3	1.4	40	64.6	4.7
	deeper	40	28.6	1.2	40	33.3	1.6	40	63.1	4.6
1000	original	40	44.6	1.2	40	45.1	1.2	40	89.7	4.2
	deeper	40	43.9	1.1	40	44.3	1.2	40	97.5	4.6

relatively small. Although we observed that the performance of the neural network is noticeably better than the nearest neighbour model, there is no systematic advantage of using the deeper, more expressive model.

4.5.2 Best upper bound within time limit

Experimental Setup

All of the experiments so far are concerned with finding a primal feasible solution with guaranteed suboptimality (e.g. 0.1%). In this section we consider a case where we are only interested in obtaining as good primal feasible solution as possible within a prescribed time budget. This is of interest when we do not necessarily have enough time to achieve a given proven tolerance.

To evaluate the performance of the primal heuristics in this setup, we compare them by the quality (suboptimality) of feasible solutions found within a prescribed time limit. The neural network and nearest neighbour partial-fixing primal heuristics do not require Dantzig-Wolfe decomposition to be run. Indeed if we are not interested in lower bounds provided by Dantzig-Wolfe decomposition, these heuristics provide feasible solutions faster if they are used as stand-alone methods. We formulate the partially-fixed MILP instances and solve them sequentially (from those with small threshold values). We note that even though we are not interested in computing a lower bound, the other primal heuristics still require Dantzig-Wolfe decomposition to be run and the method used in this section is identical to the one

used in the previous section.

We use the same neural network and nearest neighbour models as in the previous experiments (with the dataset preparation time budget of 24 hours). The configuration of the other primal heuristics are the same as well.

Results

For evaluation, the same 40 test instances as in the previous section were used. The results are shown in Table 4.17. The columns labelled as 'solved' are the number of test instances where the primal heuristics found a feasible solution within the time limits of 1, 2 and 5 minutes respectively. Among the instances where the primal heuristics found a feasible solution, the suboptimality is computed as the difference between the best upper bounds found by the primal heuristics and the best lower bounds found by running CPLEX for 4 hours.

When the time limit is 1 minute, just finding a primal feasible solution may not be trivial. On the instances of size 200, all of the primal heuristics can find feasible solutions. However, on larger instances such as those of size 600 or 1000, only the neural network partial-fixing primal heuristic and the feasibility recovery local search primal heuristic found a feasible solution on all of the test instances. The column combination primal heuristic failed to find any primal feasible solutions on more than half of the test instances of size 1000 within 1 minute. We also note that the neural network partial-fixing primal heuristic on average found primal feasible solutions of smaller suboptimality compared with the feasibility recovery local search primal heuristic.

With longer time limit, the primal heuristics are more likely to find primal feasible solutions. For the 200-generator case, the neural network partial-fixing primal heuristic performs best on average on any time limits. However, for the 600-generator and 1000-generator case, given sufficiently long time limit, such as 5 minutes, the RMP partial-fixing primal heuristic is the best and the neural network partial-fixing primal heuristic finds the second best solutions. We note that on any setup, the nearest neighbour partial-fixing primal heuristic performs worse than the neural network partial-fixing primal heuristic.

Table 4.17: Suboptimality (%) of feasible solutions found within time limits

		1 minute		2 minutes		5 minutes	
size	method	solved	subopt.	solved	subopt.	solved	subopt.
200	feasibility recovery	40	0.208	40	0.159	40	0.147
	column combination	40	0.094	40	0.084	40	0.078
	RMP partial-fixing	40	0.056	40	0.050	40	0.047
	network partial-fixing	40	**0.039**	40	**0.034**	40	**0.028**
	nearest partial-fixing	40	0.054	40	0.043	40	0.032
600	feasibility recovery	40	0.317	40	0.172	40	0.088
	column combination	10	0.107	32	0.062	40	0.024
	RMP partial-fixing	22	0.117	40	**0.028**	40	**0.021**
	network partial-fixing	40	**0.067**	40	0.039	40	0.026
	nearest partial-fixing	39	0.242	40	0.046	40	0.032
1000	feasibility recovery	40	0.301	40	0.277	40	0.072
	column combination	0	-	23	0.112	40	0.033
	RMP partial-fixing	5	0.220	40	0.079	40	**0.014**
	network partial-fixing	40	**0.243**	40	**0.042**	40	0.028
	nearest partial-fixing	39	0.550	40	0.052	40	0.035

Chapter 5

Incremental Methods

The focus of this chapter is the column generation procedure. To facilitate the discussion later, we adopt the dual point of view. That is, we consider the regularised cutting-plane method applied to the dual problem (2.5), which is the sum of many convex component functions, one for each generator. The value of each component function is given as the optimal objective value of the corresponding subproblem. The regularised cutting-plane method constructs a piecewise affine model for each component. The method compute the optimal point of the model with regularisation and evaluates the components at that point to refine the models.

In this chapter, we consider an incremental variant of the regularised cutting-plane method in a sense that it only evaluates a subset of the components in each iteration. This study is motivated by the success of other incremental first-order methods such as the incremental gradient descent method and the incremental subgradient method [50, 9]. One of the most notable examples is the training of neural networks with the stochastic gradient method [10], which is closely related to the incremental gradient descent method. Empirically it has been observed that incremental methods often converge much faster than non-incremental methods, especially when the initial point is far from the solution. The incremental method is further extended to allow random selection of components (i.e. in each step the component to compute the subgradient is selected randomly based on a nonhomogeneous Markov chain) and stochastic errors in subgradient evaluation [57]. [53] study an extension of the method to handle constraints. They assume that the con-

straints are expressed as the intersection of the level sets of convex functions and consider an incremental version of the "subgradient projection". It is also expected to be advantageous for the cutting-plane method to evaluate only a subset of the components and update the models and the iterate more frequently if the number of components is large.

In a typical approach, the cutting-plane model grows indefinitely as the algorithm proceeds. That is, the model incorporates data obtained in all of the previous iterations and its size increases every time a new point is evaluated. When the model gets large, it is of interest to delete some data (i.e. delete some cuts) and keep the model size moderate. Heuristics may be used to select cuts to be deleted, but they are not supported by any convergence results [14, Remark 9.8]. We study the behaviour of the regularised cutting-plane method in the limited-memory setup under various conditions on the regularisation.

Both of the above aspects, incremental but quick steps (update) and a cheaper but less informative RMP, may affect the performance of the method in an interrelated way. This flexibility may allow the user to customise the method and improve the efficiency. However, finding the good balance is a non-trivial task and highly problem dependent.

These two ideas have been explored for bundle methods as well [31]. The bundle methods also construct a piecewise affine model of the objective function with regularisation. The bundle methods support the limited-memory setting thanks to the technique called cut aggregation. Cut aggregation can be done very cheaply using the dual solution to the RMP. If the RMP is a bottleneck, reducing the size of the RMP in this way may be beneficial. Recently, extensions of the bundle methods to handle an incremental component evaluation have been studied [64, 22, 42]. However, the current approaches have stronger requirements (e.g. evaluation of all the components before each serious step or knowledge of the Lipschitz constance). [64] is one of the latest and most flexible method in this direction. However the study lacks numerical experiments and its actual performance is yet to be seen. We refer the reader to [64] for further discussion on recent advances in incremental bundle methods.

When the underlying problem is LP, the cutting-plane method we consider be-

comes (a regularised version of) Bender's decomposition. An incremental variant of Bender's decomposition is studied in [45]. Although the method requires much stronger assumptions (e.g. identical constraint coefficients of subproblems), the study reported impressive performance. Their analysis is based on finiteness of faces of polyhedra and does not generalise to a limited-memory setup (i.e. dropping cuts).

Since the column procedure for the Dantzig-Wolfe Decomposition is the dual of the cutting-plane methods for the Lagrangian dual problem, the same technique is applicable to the column generation procedure. In the incremental column generation procedure, we only solve a subset of the pricing subproblems and generate the corresponding columns in each iteration.

The remainder of this chapter is organised as follows. In Section 5.1 we introduce the incremental cutting-plane method. Section 5.2 gives convergence analysis of the algorithm. Section 5.4 presents numerical experiments to study empirical performance of the incremental method.

5.1 Incremental regularised cutting-plane method

In this chapter we study the following convex optimisation problem as we did in Section 2.3.1:

$$\min_{y \in Y} \left\{ f(y) = \sum_{i=1}^{m} f_i(y) \right\}. \tag{5.1}$$

To facilitate the comparison with the references, in this chapter, we use different notation. The assumption is as follows.

Assumption 5.1.

1. $Y \subset \mathbb{R}^n$ is a nonempty, closed and convex set.
2. $f_i : \mathbb{R}^n \to \mathbb{R}$ is a convex function and is finite and subdifferentiable over Y for $i = 1, 2, \ldots, m$.
3. There exists $C > 0$ for which

$$\|s_i(y)\| \leq C, \qquad \forall i, y \in Y, s_i(y) \in \partial f_i(y),$$

where $\partial f_i(y)$ is the subdifferential of f_i at y for each i.

This is a standard assumption in the study of incremental subgradient method [50]. We write $f^* = \inf_{y \in Y} f(y) \in \mathbb{R} \cup \{-\infty\}$ and $Y^* = \{y \in Y \mid f(y) = f^*\}$, which may be empty.

We note that for each i, given the convexity of f_i, the boundedness of subgradients implies that f_i is C-Lipschitz continuous. For all $i \in 1, 2, \ldots, m$ and $y_1, y_2 \in Y$, it follows

$$f_i(y_1) - f_i(y_2) \leq s_i(y_1)^T (y_1 - y_2) \leq C \| y_1 - y_2 \|,$$

where $s_i(y_1)$ is a subgradient of f_i at y_1. We note that when $Y = \mathbb{R}^n$, the converse is true [3, Theorem 3.61]: If f_i is C-Lipschitz continuous, the norm of the subgradients is bounded by C.

The regularised cutting-plane method is an iterative solution method which computes candidate points $\{y_k\}$ to solve (5.1) approximately. In the kth iteration, given the previous points $y_0, y_1, \ldots, y_k$, the method computes the next point y_{k+1} by minimising a regularised model

$$y_{k+1} = \arg\min_{y \in Y} \sum_{i=1}^{m} \varphi_{k,i}(y) + \frac{1}{2t_k} \| y - y_k \|^2,$$

where $\varphi_{k,i}$ is a piecewise affine model of the ith component

$$\varphi_{k,i}(y) = \max_{l=0,1,\ldots,k} f_i(y_{k-l}) + s_i(y_{k-l})^T (y - y_{k-l}), \tag{5.2}$$

$f_i(y_l)$ and $s_i(y_l)$ are the value and any subgradient of f_i at y_l ($l = 0, 1, \ldots, k$) and t_k is a parameter to adjust the strength of the regularisation. We refer to t_k as the step size in a reason described later. The values and subgradients of the components are evaluated at the new point y_{k+1} to update the model (5.2) and the above process is repeated.

In this chapter, we consider an incremental variant of the method in a sense that it only evaluates a subset of the components f_i in each iteration. We denote the index set of components evaluated in the kth iteration by I_k. Since there are m components,

$$I_k \subset \{1, 2, \ldots, m\}, \qquad \forall k.$$

We note that the standard cutting-plane method is a special case where $I_k = \{1, 2, \ldots, m\}$ for all k. We refer to this non-incremental method as the full-step method. The requirement of our analysis is that each component must be evaluated at least once in W iterations, where W is a prescribed number.

Furthermore, it is of interest to bound the memory requirement of the method. To this end, we consider a method which may delete some cuts. Our analysis works as long as each model for each component keeps its newest cut. This allows one to limit the size of the model by a prescribed value. Deleted cuts may be put back to the model in later iterations. The algorithm is described in Algorithm 3.

Algorithm 3 Incremental cutting-plane method

select initial point $y_0 \in Y$, step size $\{t_k\}$, evaluation schedule $\{I_k\}$.
for $i = 1, 2, \ldots, m$, initialise the model $\varphi_{0,i} = 0$ and $B_{0,i} = \emptyset$.
for k in $\{1, 2, \ldots\}$ **do**
  for i in I_k **do**
    Evaluate $f_i(y_k)$ and $s_i(y_k) \in \partial f_i(y_k)$.
    Add the following cut to the model $\varphi_{k,i}$

$$f_i(y_k) + s_i(y_k)^T (y - y_k).$$

    Add k to $B_{k,i}$.
  end for
  (Optionally) **for** i in $\{1, 2, \ldots, m\}$, delete some cuts except the newest one.
  Let

$$\begin{aligned} y_{k+1} &= \arg\min_{y \in Y} \widetilde{\varphi}_k(y), \qquad (5.3) \\ \widetilde{\varphi}_k(y) &= \varphi_k(y) + \frac{1}{2t_k} \|y - y_k\|^2, \\ \varphi_k(y) &= \sum_{i=1}^{m} \varphi_{k,i}(y), \\ \varphi_{k,i}(y) &= \begin{cases} \max_{l \in B_{k,i}} f_i(y_l) + s_i(y_l)^T (y - y_l), & \text{if } B_{k,i} \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

  for each i in $\{1, 2, \ldots, m\}$, copy the model $\varphi_{k,i}(y)$ to $\varphi_{k+1,i}(y)$ and $B_{k,i}$ to $B_{k+1,i}$.
end for

Note that the above algorithm is well-defined: For each k, $\widetilde{\varphi}_k$ is a proper closed $(1/2t_k)$-strongly convex function and a minimiser of $\widetilde{\varphi}_k$ exists and is unique [3, Theorem 5.25].

We assume the following on some analysis below.

Assumption 5.2. Each component is evaluated at least once in every W iterations

$$i \in \bigcup_{l=0}^{W-1} I_{k+l}, \qquad \forall k \geq 0, i = 1,2,\ldots,m.$$

Under this assumption, each model has at least one cut after W iterations. Furthermore, each model has a cut which corresponds to a point not older than W iterations. That is,

$$\min\{k-l \mid l \in B_{k,i}\} < W, \qquad \forall k \geq W-1, i = 1,2,\ldots,m.$$

5.2 Convergence analysis

Corollary VI.4.3.2 from Hiriart-Urruty and Lemaréchal [30] shows that for $k \geq 0$ and $i = 1,2,\ldots,m$ such that $B_{k,i} \neq \emptyset$, it follows

$$\partial\varphi_{k,i}(y) \subset \operatorname{conv}\{s_i(y_l) \mid l \in B_{k,i}\}, \qquad \forall y \in Y. \tag{5.4}$$

Furthermore, by Theorem VI.4.1.1. from Hiriart-Urruty and Lemaréchal [30], we have

$$\partial\widetilde{\varphi}_k(y) = \partial\varphi_k(y) + \frac{1}{t_k}(y - y_k), \qquad \forall k \geq 0, y \in Y, \tag{5.5}$$

and

$$\partial\varphi_k(y) = \sum_{i=1}^{m} \partial\varphi_{k,i}(y), \qquad \forall k \geq 0, y \in Y. \tag{5.6}$$

Thus, it follows that

$$\|s_i{}'\| \leq C, \qquad \forall k \geq 0, i = 1,2,\ldots,m, s_i{}' \in \partial\varphi_{k,i},$$

and

$$\|s'\| \leq mC, \qquad \forall k \geq 0, s' \in \partial\varphi_k.$$

In particular, $\varphi_{k,i}$ is C-Lipschitz continuous for all $k \geq 0$ and $i = 1,2,\ldots,m$.

The next lemma allows us to write the next point using the current point and a

subgradient of the model.

Lemma 5.3. *Let Assumption 5.1 hold. For $k = 0, 1, \ldots$, there exists $\bar{s}_k \in \partial\varphi_k(y_{k+1})$, $\bar{s}_{k,i} \in \partial\varphi_{k,i}(y_{k+1})$ and $p_k \in N_Y(y_{k+1})$ such that*

$$y_{k+1} = y_k - t_k(\bar{s}_k + p_k) = y_k - t_k\left(\sum_{i=1}^{m} \bar{s}_{k,i} + p_k\right).$$

Furthermore, we have

$$y_{k+1} = P_Y(y_k - t_k\bar{s}_k),$$

where P_Y denotes the (Euclidean) projection onto the set Y.

We observe that the iteration resembles that of the projected subgradient method. (5.4) shows that $\bar{s}_{k,i}$ is a convex combination of the subgradients evaluated in the previous iterations (or $\bar{s}_{k,i} = 0$ if $B_{k,i} = \emptyset$). We sum all $\bar{s}_{k,i}$, multiply the "step size" t_k and make a "subgradient step" followed by the projection.

Proof. The optimality condition of (5.3) is

$$0 \in -\partial\widetilde{\varphi}_k(y_{k+1}) + N_Y(y_{k+1}).$$

Using (5.5) and (5.6), we get the first relation. Let $x_k = P_Y(y_k - t_k\bar{s}_k)$. By the definition of the projection, x_k is the unique minimiser of the following minimisation problem:

$$\min_{y \in Y} \|y - (y_k - t_k\bar{s}_k)\|^2.$$

The optimality condition gives

$$0 \in -(x_k - (y_k - t_k\bar{s}_k)) + N_Y(x_k).$$

In light of the first relation, we obtain $y_{k+1} = P_Y(y_k - t_k\bar{s}_k)$. □

With the preceding lemma, one can estimate the distance between iterates.

Lemma 5.4. *Let Assumption 5.1 hold. For any k_1 and k_2 with $0 \le k_1 < k_2$, we have*

$$\|y_{k_2} - y_{k_1}\| \le (k_2 - k_1)m\bar{t}_{k_1,k_2-1}C,$$

where $\bar{t}_{k_1,k_2-1} = \max\{t_{k_1}, t_{k_1+1}, \ldots, t_{k_2-1}\}$.

Proof. Using Lemma 5.3 and the non-expansion property of the projection, for all $k \geq 0$, it follows

$$\begin{aligned} \|y_{k+1} - y_k\| &= \|P_Y(y_k - t_k \bar{s}_k) - y_k\| \\ &\leq \|y_k - t_k \bar{s}_k - y_k\| \\ &\leq m t_k C. \end{aligned}$$

Let k_1 and k_2 be integers such that $0 \leq k_1 < k_2$. Combining the preceding inequality with the triangle inequality, we obtain the desired inequality

$$\begin{aligned} \|y_{k_2} - y_{k_1}\| &= \left\| \sum_{l=0}^{k_2-k_1-1} (y_{k_1+l+1} - y_{k_1+l}) \right\| \\ &\leq \sum_{l=0}^{k_2-k_1-1} \|y_{k_1+l+1} - y_{k_1+l}\| \\ &\leq \sum_{l=0}^{k_2-k_1-1} m t_{k_1+l} C \\ &\leq (k_2 - k_1) m \bar{t}_{k_1,k_2-1} C. \end{aligned}$$

□

Using the notation in Lemma 5.3, define the following affine function

$$\bar{f}_k(y) = \varphi_k(y_{k+1}) + \bar{s}_k^T (y - y_{k+1}). \tag{5.7}$$

Since $\bar{s}_k \in \partial\varphi_k(y_{k+1})$, using the definition of the subgradient,

$$\bar{f}_k(y) \leq \varphi_k(y), \qquad \forall k \geq 0, y \in Y,$$

with equality at y_{k+1}. In particular, we get

$$\bar{f}_k(y) \leq \varphi_k(y) \leq f(y), \qquad \forall k \geq W_0, y \in Y. \tag{5.8}$$

Lemma 5.5. *Let Assumption 5.1 hold. For any $k \geq 0$,*

$$\bar{f}_k(y_k) \geq \bar{f}_k(y_{k+1}).$$

Proof. By definition of $\bar{f}_k$ (5.7)

$$\bar{f}_k(y_k) - \bar{f}_k(y_{k+1}) = \bar{s}_k^T(y_k - y_{k+1}).$$

Using Lemma 5.3, we get

$$\begin{aligned}
\bar{s}_k^T(y_k - y_{k+1}) &= \left(\frac{1}{t_k}(y_k - y_{k+1}) - p_k\right)^T (y_k - y_{k+1}) \\
&= \frac{1}{t_k}\|y_k - y_{k+1}\|^2 - p_k^T(y_k - y_{k+1}) \\
&\geq -p_k^T(y_k - y_{k+1}).
\end{aligned}$$

Since $p_k \in N_Y(y_{k+1})$ and $y_k \in Y$,

$$p_k^T(y_k - y_{k+1}) \leq 0.$$

Combining the preceding two relations, we obtain

$$\bar{f}_k(y_k) - \bar{f}_k(y_{k+1}) \geq 0.$$

□

Now we prove a fundamental inequality which plays a key role in the convergence analysis of the method.

Lemma 5.6. *Let Assumption 5.1 and 5.2 hold. For any $x \in Y$ and any $k \geq W$, we have*

$$\|y_{k+1} - x\|^2 \leq \|y_k - x\|^2 - 2t_k(f(y_k) - f(x)) + 3m^2 t_k^2 C^2 + 4m^2 t_k \bar{t}_{k-W+1,k} C^2 W.$$

Proof. Using Lemma 5.3, the non-expansion property of the projection and defini-

tion of $\bar{f}_k$ (5.7), for all $k \geq 0$, it follows

$$\begin{aligned}
&\|y_{k+1} - x\|^2 - \|y_k - x\|^2 \\
&\quad = \|P_Y(y_k - t_k \bar{s}_k) - x\|^2 - \|y_k - x\|^2 \\
&\quad \leq \|y_k - t_k \bar{s}_k - x\|^2 - \|y_k - x\|^2 \\
&\quad = -2t_k \bar{s}_k^T (y_k - x) + t_k^2 \|\bar{s}_k\|^2 \\
&\quad = 2t_k \left(\bar{f}_k(x) - \bar{f}_k(y_k)\right) + t_k^2 \|\bar{s}_k\|^2.
\end{aligned}$$

We shall bound the two terms in the last line.

For $k \geq W-1$, using (5.8), Lemma 5.5 and definition of $\bar{f}_k$ (5.7),

$$\begin{aligned}
\bar{f}_k(x) - \bar{f}_k(y_k) &\leq f(x) - \bar{f}_k(y_k) \\
&\leq f(x) - \bar{f}_k(y_{k+1}) \\
&= f(x) - \varphi_k(y_{k+1}) \\
&= f(x) - \sum_{i=1}^{m} \varphi_{k,i}(y_{k+1})
\end{aligned}$$

Fix i and let $l_i = \max B_{k,i} \geq k - W + 1$. By the construction of the model, $f_i(y_{l_i}) = \varphi_{k,i}(y_{l_i})$. Since $\varphi_{k,i}$ and f_i are C-Lipschitz continuous, we have

$$\begin{aligned}
\varphi_{k,i}(y_{k+1}) &= \varphi_{k,i}(y_{k+1}) - \varphi_{k,i}(y_{l_i}) + \varphi_{k,i}(y_{l_i}) - f_i(y_k) + f_i(y_k) \\
&\geq f_i(y_k) - C\|y_{k+1} - y_{l_i}\| - C\|y_k - y_{l_i}\| \\
&\geq f_i(y_k) - C\|y_k - y_{k+1}\| - 2C\|y_k - y_{l_i}\| \\
&\geq f_i(y_k) - m t_k C^2 - 2m \bar{t}_{k-W+1,k} C^2 W.
\end{aligned}$$

We have invoked Lemma 5.4 in the last inequality. Combining the preceding two relations,

$$\bar{f}_k(x) - \bar{f}_k(y_k) \leq f(x) - f(y_k) + m^2 t_k C^2 + 2m^2 \bar{t}_{k-W+1,k} C^2 W.$$

Now we shall bound the other term. With the triangle inequality, for all $k \geq 0$,

$$\|\bar{s}_k\|^2 = \left\| \sum_{i=1}^{m} \bar{s}_{k,i} \right\|^2 \leq \left(\sum_{i=1}^{m} \|\bar{s}_{k,i}\| \right)^2 \leq m^2 C^2.$$

Using the two bounds, we obtain the desired relation. □

The preceding lemma is an analogue of Lemma 2.1 from Nedic [50], which is a basis of the convergence analysis of the incremental subgradient method. Following their analysis of the incremental subgradient method, now we can obtain various convergence properties. The three propositions below correspond to Proposition 2.1, 2.4 and 2.6 from Nedic [50], respectively.

Proposition 5.7. *Let Assumption 5.1 and 5.2 hold. Assume that step size* $\{t_k\}$ *is bounded from both below and above as* $0 < t_{\min} \leq t_k \leq t_{\max}$.

1. *If* $f^* = -\infty$, *then*

$$\liminf_{k\to\infty} f(y_k) = -\infty.$$

2. *If* $f^* > -\infty$, *then*

$$\liminf_{k\to\infty} f(y_k) \leq f^* + \frac{1}{2t_{\min}} \left(3m^2 t_{\max}^2 C^2 + 4m^2 t_{\max}^2 C^2 W\right).$$

The above lemma estimates the worst-case suboptimality the method may suffer when the step size is kept bounded away from 0 and bounded from above. The bound gets smaller as the interval of the evaluation W decreases. Also, when the step size is constant ($t_{\min} = t_{\max}$), the bound gets improved as the step size gets smaller.

Proof. We follow the proof of Proposition 2.1 in [51].

Suppose for contradiction that the result does not hold. Then there exists an $\epsilon > 0$ such that

$$\liminf_{k\to\infty} f(y_k) - \frac{1}{2t_{\min}} \left(3m^2 t_{\max}^2 C^2 + 4m^2 t_{\max} C^2 W\right) - 2\epsilon > f^*.$$

Let $\hat{x} \in Y$ be such that

$$\liminf_{k\to\infty} f(y_k) - \frac{1}{2t_{\min}} \left(3m^2 t_{\max}^2 C^2 + 4m^2 t_{\max} C^2 W\right) - 2\epsilon \geq f(\hat{x})$$

and let $k_0 \geq W$ be large enough so that for all $k \geq k_0$, we have

$$f(y_k) \geq \liminf_{k' \to \infty} f(y_{k'}) - \epsilon.$$

By combining the preceding two relations, we obtain

$$f(y_k) - f(\hat{x}) \geq \frac{1}{2t_{\min}} \left(3m^2 t_{\max}^2 C^2 + 4m^2 t_{\max} C^2 W\right) + \epsilon, \qquad \forall k \geq k_0.$$

Using Lemma 5.6, where $x = \hat{x}$, together with the preceding relation, we see that

$$\|y_{k+1} - \hat{x}\|^2 \leq \|y_k - \hat{x}\|^2 - 2\epsilon t_{\min}, \qquad \forall k \geq k_0 (\geq W),$$

implying that

$$\|y_{k+1} - \hat{x}\|^2 \leq \|y_k - \hat{x}\|^2 - 2\epsilon t_{\min} \leq \cdots \leq \|y_{k_0} - \hat{x}\|^2 - 2\epsilon(k + 1 - k_0) t_{\min},$$

which cannot hold for a sufficiently large k. □

With a diminishing step size rule, the method is capable of solving the problem asymptotically.

Proposition 5.8. *Let Assumption 5.1 and 5.2 hold. Assume that step size* $\{t_k\}$ *is such that*

$$t_k > 0 \ (\forall k), \ t_k \geq t_{k+1} \ (\forall k), \ \lim_{k \to \infty} t_k = 0, \ \sum_{k=0}^{\infty} t_k = \infty.$$

Then, we have

$$\liminf_{k \to \infty} f(y_k) = f^*.$$

Proof. We follow the proof of Proposition 2.4 in [51].

Suppose for contradiction that there exists an $\epsilon > 0$ such that

$$\liminf_{k \to \infty} f(y_k) + 2\epsilon > f^*.$$

Let $\hat{x} \in Y$ be such that

$$\liminf_{k \to \infty} f(y_k) \geq f(\hat{x}) + 2\epsilon,$$

and let $k_0 \geq W$ be large enough so that for all $k \geq k_0$, we have

$$f(y_k) \geq \liminf_{k' \to \infty} f(y_{k'}) - \epsilon.$$

Then, we have

$$f(y_k) - f(\hat{x}) \geq \epsilon, \qquad \forall k \geq k_0.$$

Using Lemma 5.6 with $x = \hat{x}$, we get

$$\|y_{k+1} - \hat{x}\|^2 \leq \|y_k - \hat{x}\|^2 - t_k(2\epsilon - 3m^2 t_k C^2 - 4m^2 t_{k-W+1} C^2 W).$$

Because $t_k \to 0$, without loss of generality, we may assume that k_0 is large enough so that $\epsilon \geq 3m^2 t_k C^2 + 4m^2 t_{k-W+1} C^2 W$ for all $k \geq k_0$, implying that

$$\|y_{k+1} - \hat{x}\|^2 \leq \|y_k - \hat{x}\|^2 - t_k \epsilon m \leq \cdots \leq \|y_{k_0} - \hat{x}\|^2 - \epsilon \sum_{j=k_0}^{k} t_k,$$

which cannot hold for a sufficiently large k. □

Proposition 5.9. *Let Assumption 5.1 and 5.2 hold. Assume that step size* $\{t_k\}$ *is such that*

$$t_k > 0 \ (\forall k), \ t_k \geq t_{k+1} \ (\forall k), \ \sum_{k=0}^{\infty} t_k = \infty, \ \sum_{k=0}^{\infty} t_k^2 < \infty,$$

and assume that the set of optimal solutions Y^* *is nonempty. Then, the sequence* $\{y_k\}$ *converges to some optimal solution.*

Proof. We follow the proof of Proposition 2.6 in [51].

By Lemma 5.6, where $x = y^*$ with $y^* \in Y^*$, we have

$$\begin{aligned} \|y_{k+1} - y^*\|^2 &\leq \|y_k - y^*\|^2 - 2t_k(f(y_k) - f^*) + 3m^2 t_k^2 C^2 + 4m^2 t_k t_{k-W+1} C^2 W \\ &\leq \|y_k - y^*\|^2 - 2t_k(f(y_k) - f^*) + 3m^2 t_k^2 C^2 + 4m^2 t_{k-W+1}^2 C^2 W. \end{aligned} \tag{5.9}$$

Since $f(y_k) - f^* \geq 0$ for all k and $\sum_{k=0}^{\infty} t_k^2 < \infty$, it follows that the sequence $\{y_k\}$ is bounded. Furthermore, by Proposition 5.8, we have that

$$\liminf_{k \to \infty} f(y_k) = f^*.$$

Let $\{y_{k_j}\}$ be a subsequence of $\{y_k\}$ along which the above liminf is attained, so that

$$\lim_{j\to\infty} f(y_{k_j}) = f^*.$$

The sequence $\{y_{k_j}\}$ is bounded, so it has limit points. Let $\bar{y}$ be one of them, and without loss of generality we can assume that $y_{k_j} \to \bar{y}$. By continuity of f, we have that $\bar{y} \in Y^*$, so from (5.9) with $y^* = \bar{y}$, we obtain for any j and any $k \geq k_j$ such that $k_j \geq W$,

$$\begin{aligned}\|y_{k+1} - \bar{y}\|^2 &\leq \|y_k - \bar{y}\|^2 + 3m^2 t_k^2 C^2 + 4m^2 t_{k-W+1}^2 C^2 W \\ &\leq \|y_{k_j} - \bar{y}\|^2 + 3m^2 C^2 \sum_{l=k_j}^{k} t_l^2 + 4m^2 C^2 W \sum_{l=k_j}^{k} t_{l-W+1}^2.\end{aligned}$$

Taking first the limit as $k \to \infty$ and then the limit as $j \to \infty$, from the preceding relation we obtain

$$\limsup_{k\to\infty} \|y_{k+1} - \bar{y}\|^2 \leq \lim_{j\to\infty} \|y_{k_j} - \bar{y}\|^2 + 3m^2 C^2 \lim_{j\to\infty} \sum_{l=k_j}^{\infty} t_l^2 + 4m^2 C^2 W \lim_{j\to\infty} \sum_{l=k_j}^{\infty} t_{l-W+1}^2,$$

which by $y_{k_j} \to \bar{y}$ and $\sum_{k=0}^{\infty} t_k^2 < \infty$, implies that

$$\limsup_{k\to\infty} \|y_{k+1} - \bar{y}\|^2 = 0,$$

and consequently, $y_k \to \bar{y}$ with $\bar{y} \in Y^*$. □

5.3 Pratical considerations

As we studied in the previous chapter, the incremental regularised cutting-plane method finds a solution asymptotically given the step size is controlled appropriately. However, when we apply the method to the dual problem to the UC problem, there are a few points to be considered.

First, the RMP partial-fixing primal heuristic is still applicable with the incremental method. The primal heuristic does not require any modification and the property on the boundedness of the number of fractional values still holds. Since the

incremental steps take a much shorter time than the full-step method, the primal heuristic may be run less frequently.

Second, the method does not gives a valid lower bound to the UC problem. When the incremental regularised cutting-plane method is used to solve the dul problem to the UC problem, a lower bound is necessary to compute a suboptimality gap. To obtain lower bounds one needs to run full-steps. Alternatively one can use a model to compute valid lower bounds as discussed in [64] (the reference studies a minimisation problem and the model is called the upper model).

In the numerical experiments in the next section we consider a simplified setting. The study on the use of primal heuristics with the incremental method is postponed until Chapter 6.

5.4 Numerical experiments

In this section the incremental cutting-plane method is applied to the dual problems of UC problem and is compared with the full-step method. We use the test instances of size 200. The data is described in Section 3.3. To simplify the discussion we do not consider primal heuristics in the numerical experiments below.

The incremental cutting-plane method used in this section is the one described in Algorithm 3. For simplicity, we consider the constant step size rule ($t_k = t$ for all k). Furthermore, although in the previous analysis we consider a setting where we delete cuts, in our numerical experiments we do not delete any cuts. When adding a new cut the model checks whether it already contains the same cut or not. The cut is only added when it is not yet present in the model. When the full-step method is used the objective value at each point is readily available and this can be used to monitor the progress. In our numerical experiments we only update the regularisation centre y_k when the objective value gets improved. If the objective value of the point computed by the model is worse than that of the current point, only the model gets updated and the regularisation centre is kept at the same point. This modification improves the performance of the full-step method and is often used in practice (e.g. [61]).

An evaluation schedule $\{I_k\}$ is chosen based on a permutation. In our experiments the number of evaluated components per iteration, denoted by p, is fixed to be

the same in all iterations. Namely, $|I_k| = p$ for all k. We consider a sequence of component functions where every m elements are a permutation of $f_1, f_2, \ldots, f_m$

$$\underbrace{f_{h_1^1}, f_{h_2^1}, \ldots, f_{h_m^1}}_{\text{1st permutation}}, \underbrace{f_{h_1^2}, f_{h_2^2}, \ldots, f_{h_m^2}}_{\text{2nd permutation}}, \ldots.$$

In every iteration, the first p components are removed from the sequence and evaluated. This is a generalisation of the approach taken in the study on the incremental subgradient method [50], which corresponds to $p = 1$ where a single component is evaluated per iteration. Setting $p = m$ we obtain the full-step method which evaluates all of the components in each step.

For later use, we introduce an adjusted step size $\widetilde{t} = mt/p$ where t is the actual step size. We note that the method with a small value of p makes more steps (i.e. solves model (5.3) more often) than the method with a large value of p. If all of the component functions are affine, the method makes roughly the same progress per component evaluation no matter which p value is used, as long as the adjusted step size $\widetilde{t}$ is the same. Thus, the adjusted step size $\widetilde{t}$ facilitates the comparison of experiments with different values of p.

Two approaches to initialise the cutting-plane method are considered: warmstart based on the LPR and coldstart with $y = 0$. Coldstart has to be used when there is no prior knowledge of the problem to be solved. However, as shown in Chapter 3, warmstart may significantly reduce the computational time if available. We will observe how the incremental method works on these two setups.

To observe the effect of the incremental update the cutting-plane method is applied to the test instances with various values for the number of component evaluations per iteration $|I_k| = p$. Table 5.1 shows the average computational time, the average number of component evaluations and the average number of iterations (the average number of the solutions of model (5.3)) needed to find a solution within 0.1% and 0.05% tolerance. All times in this section are wall-clock times. If the method is warmstarted, the computational time includes the time spent to solve the LPR, which is approximately 5 seconds. For each value of p, the adjusted step size $\widetilde{t}$ is set to 0.01 (i.e. the step size is $t = 0.01p/m$). The incremental method does not compute

the function value $f(y_k)$ for each iteration so they are evaluated after the experiments and the time to do this is not included.

Table 5.1: Performance of the cutting-plane method with various evaluation schedule sizes p and with $\widetilde{t} = 0.01$

		tol: 0.1%			0.05%		
initialiser	p/m	time	comp. eval.	iter.	time	comp. eval.	iter.
warmstart	0.05	14.4	345.0	34.5	19.8	531.2	53.1
	0.10	**13.8**	347.5	17.4	**18.9**	537.5	26.9
	0.20	14.4	375.0	9.4	20.1	585.0	14.6
	0.30	15.5	420.0	7.0	20.0	607.5	10.1
	0.40	16.7	480.0	6.0	23.3	740.0	9.2
	0.50	19.2	587.5	5.9	25.3	825.0	8.2
	1.00	22.2	700.0	3.5	30.7	1025.0	5.1
coldstart	0.05	23.5	911.2	91.1	39.0	1226.2	122.6
	0.10	20.0	910.0	45.5	32.7	1232.5	61.6
	0.20	**19.4**	950.0	23.8	**29.2**	1235.0	30.9
	0.30	19.9	1005.0	16.8	29.5	1305.0	21.8
	0.40	22.3	1110.0	13.9	31.8	1400.0	17.5
	0.50	25.6	1237.5	12.4	36.1	1537.5	15.4
	1.00	51.2	2075.0	10.4	66.9	2450.0	12.2

As shown in Table 5.1, for any value of p the warmstarted method solves the problem quicker than the coldstarted method. The reason for this is that the LPR gives a dual value close to the optimal one which helps the method to find a solution in a shorter time.

We also note that whichever initialisation method is used, the incremental method ($p/m < 1$) performs better than the full-step method ($p/m = 1$). In general, an incremental first-order method is expected to be more efficient than the full-step method when the initial point is far from the solution, but the convergence to a solution of high precision is expected to be slower [9]. However, the incremental cutting-plane method successfully finds a solution of high precision (e.g. 0.05%) in a shorter time than the full-step method. We note that as the number of component evaluations per iteration p gets smaller, the method tends to find a solution with fewer component evaluations in total. However, at the same time, the number of iterations and hence the number of solution of model (5.3) gets larger. If p is too small, the time spent to solve (5.3) becomes significant and the overall computational time increases. The

best value of p balances the number of component evaluations in total and the growth of model solution time.

Table 5.2 shows the performances of the methods with various step sizes. As shown in the table, the performance of the methods depends on the choice of the step size. When the step size is very small, the numbers of component evaluations required by the incremental method and the full-step method tend to be close. With a small step size of $\widetilde{t} = 0.001$, the regularisation in the model (5.3) is strong and the methods do not make much progress per iteration. The behaviours of the two methods are similar in this situation. However, in the setup in Table 5.2, the incremental method with $p/m = 0.1$ has to solve the model ten times more often than the full-step method per component evaluation. As a result, it requires slightly longer computational times. On the other hand, an excessively large step size slows down the method as well. Although this holds both for the incremental method and for the full-step method, the growth of the computational time is more significant in the full-step method. In other words, the incremental method is more robust and tends to work relatively well even if the step size is set larger than the best value.

Figure 5.1 shows the objective values at the points computed by the method on one of the test instances. Its (near-optimal) objective value is computed by running the warmstarted full-step method for 30 minutes and the plotted values are normalised so that the computed objective value becomes 1. The left and right figures correspond to the warmstart and the coldstart respectively. In each plot, the full-step method and incremental method are shown in solid and dashed lines respectively. The warmstarted method first solves the LPR, which approximately take 5 seconds, while the coldstarted method starts outputting the iterates immediately. In both cases, the full-step method with step size 0.1 often outputs points whose objective values are out of the range plotted and do not appear in the figures.

The figure on the left shows that the strength of the regularisation (i.e. the reciprocal of the step size) has a significant effect on the behaviour of the full-step method, which is plotted in the solid lines. We note that all of the cases use the same initial point which is close to the optimal solution, computed by the LPR. We see that the full-step method with the weakest regularisation (step size 0.1) is quite unstable and outputs dual values far away from the optimal one. This is expected behaviour since

Table 5.2: Performance of the cutting-plane method with various step sizes

			tol: 0.1%			0.05%		
initialiser	p/m	$\widetilde{t}$	time	comp. eval.	iter.	time	comp. eval.	iter.
warmstart	0.1	0.100	26.1	677.5	33.9	34.0	907.5	45.4
		0.050	20.4	530.0	26.5	28.1	772.5	38.6
		0.010	13.8	347.5	17.4	**18.9**	537.5	26.9
		0.005	**13.8**	360.0	18.0	22.4	690.0	34.5
		0.001	27.9	940.0	47.0	77.1	2735.0	136.8
	1.0	0.100	65.0	1775.0	8.9	101.1	2650.0	13.2
		0.050	46.2	1350.0	6.8	66.0	1950.0	9.8
		0.010	22.2	700.0	3.5	30.7	1025.0	5.1
		0.005	**18.7**	600.0	3.0	**27.7**	975.0	4.9
		0.001	26.6	950.0	4.8	70.5	2775.0	13.9
coldstart	0.1	0.100	28.9	975.0	48.8	38.9	1195.0	59.8
		0.050	24.4	900.0	45.0	**32.7**	1095.0	54.8
		0.010	**20.0**	910.0	45.5	32.7	1232.5	61.6
		0.005	25.2	1145.0	57.2	56.9	1902.5	95.1
		0.001	143.3	4507.5	225.4	416.3	8847.5	442.4
	1.0	0.100	267.9	4300.0	21.5	347.9	4850.0	24.2
		0.050	186.9	3750.0	18.8	218.8	4050.0	20.2
		0.010	51.2	2075.0	10.4	66.9	2450.0	12.2
		0.005	**35.3**	1725.0	8.6	**56.5**	2350.0	11.8
		0.001	98.5	4500.0	22.5	235.2	88r5.0	44.4

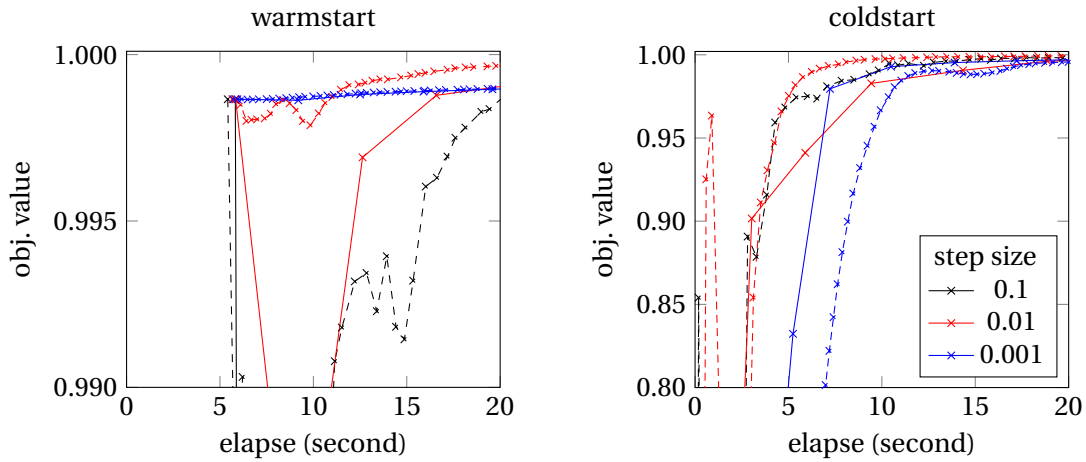

Figure 5.1: Objective values of the iterates by the warmstarted (left) and coldstarted (right) method. Solid and dashed lines correspond to the full-step method and the incremental method respectively.

the model does not have enough cuts in the first few iterations and without suitable regularisation the model tends to output points far from the solution, even if it is initialised at a near-optimal point. When the step size is 0.01, the full-step method still shows instability at first but makes constant progress afterwards. If the step size decreased to the further smaller value 0.001, the full-step method does not show such unstable behaviour at all but the entire progress becomes significantly slower. As a result, the full-step method with step size 0.001 takes longer than that of step size 0.01.

The same discussion applies to the warmstarted incremental method. However, the instability of the incremental method observed with step size 0.1 and 0.01 is much less than with the full-step method. The incremental method does also show unstable behaviour at first but becomes stable more quickly than the full-step method, and this leads to the reduction of the overall computational time. However, the incremental update does not overcome the slow progress caused by the too small step size 0.001, and the full-step method and incremental method behave quite similarly.

Similar trends are observed with the coldstart case, which is plotted on the right. We still observe the advantage of the incremental update with the larger step sizes (0.1 and 0.01). Namely the incremental update mitigate the instability of the cutting-plane method. However, the incremental method with the small step size (0.001) is somewhat slower than the full-step method.

Chapter 6

Integrated Approach

In Chapter 3 we found the initialisation method based on a neural network performs better than the other initialisation methods, such as the one based on the LPR. Furthermore, the numerical experiments in Chapter 4 showed that the neural network partial-fixing primal heuristic performs well when the tolerance is loose (0.5% and 0.25%) while the RMP partial-fixing primal heuristic outperforms others when the suboptimality tolerance is small (0.1%). In Chapter 5 the incremental column generation procedure showed superior performance compared with the full-step column generation procedure. A natural question is whether combining these techniques leads to further speed-up or not. This chapter aims to address this question.

To reduce clutter we refer to a method by combining its initialisation method name and primal heuristic name. For example, the method which uses the initialisation method based on the LPR and the RMP partial-fixing primal heuristic is referred to as the LPR-RMP method. If not specified, the full-step column generation procedure is used. When it may cause confusion, we explicitly state the type of the column generation procedure (e.g. the full network-network method, the incremental network-RMP method).

6.1 Initialisation method and primal heuristic

In this section we study combinations of the initialisation methods and the primal heuristics. We consider the initialisation methods based on a neural network and

the LPR, and the neural network and RMP partial-fixing primal heuristics. All the methods in this section use the full-step column generation procedure.

The performance of the methods was evaluated on the 40 test instances as described before and the result is shown in Table 6.1. In this experiment all the methods could find a solution with suboptimality smaller than 0.1% on all the test instances within 10 minutes.

Table 6.1: Average computational time (s) and iterations

			0.5% opt.		0.25% opt.		0.1% opt.	
size	initialiser	primal heuristic	time	iter	time	iter	time	iter
200	LPR	RMP partial-fixing	22.8	5.5	24.0	5.8	37.5	8.4
		network partial-fixing	9.3	1.0	12.0	1.3	56.3	5.2
	network	RMP partial-fixing	25.2	6.0	25.8	6.1	42.2	9.2
		network partial-fixing	**6.0**	1.0	**6.3**	1.0	**35.8**	3.0
600	LPR	RMP partial-fixing	60.7	4.4	62.8	4.6	76.8	5.7
		network partial-fixing	39.6	1.3	41.7	1.3	128.0	4.2
	network	RMP partial-fixing	52.5	4.6	54.6	4.8	**70.5**	5.8
		network partial-fixing	**23.7**	1.1	**25.4**	1.2	76.5	2.6
1000	LPR	RMP partial-fixing	85.5	3.8	91.4	4.1	113.7	5.2
		network partial-fixing	70.8	1.3	75.1	1.4	181.6	3.9
	network	RMP partial-fixing	56.8	3.3	59.5	3.4	**85.7**	4.6
		network partial-fixing	**42.5**	1.2	**42.6**	1.2	87.6	2.0

For each primal heuristic the initialisation method based on a neural network is better than the method based on the LPR in terms of the computational time with the one exception of the 200-generator test instances where the LPR-RMP method is better than the network-RMP method. It is less clear cut but the initialisation method based on a neural network usually requires fewer iterations compared with the initialisation method based on the LPR.

The preferable choice of the primal heuristic depends on the suboptimality tolerance. When the tolerance is loose, such as 0.5% or 0.25%, the neural network partial-fixing primal heuristic is better than the RMP partial-fixing primal heuristic. Among all the combinations the initialisation method based on a neural network combined with the neural network partial-fixing primal heuristic performs best if the suboptimality tolerance is 0.5% or 0.25%. On the other hand, when the tolerance is small (0.1%), the RMP partial-fixing primal heuristic often outperforms the

neural network partial-fixing primal heuristic. As a result the network-RMP method often performs best in this case. In no cases does the optimal combination use the initialisation method based on the LPR.

6.2 Incremental regularised column generation

In the previous section we observed that the network-RMP method, which uses the initialisation method based on a neural network and the RMP partial-fixing primal heuristic, tends to have the best performance when the tolerance is small (0.1%). In the remainder of this chapter we combine this method with the incremental column generation with the aim of improving the performance of the method for the small suboptimality gap of 0.1%.

Early experiments showed that naive integration of the incremental column generation procedure based on cyclic column generation does not speed up the method. We note that the numerical experiments in Chapter 5 studied the performance of the incremental method to achieve a dual solution of small suboptimality, such as 0.05%, while here we are interested in solving the UC problem with 0.1% suboptimality. Furthermore, since we cannot compute a lower bound after the incremental column generation, we need to run the full column generation at a certain frequency to obtain a lower bound, which introduces some overhead.

However, if we are allowed to run offline preparation, we can train a model to decide which pricing subproblems to solve to generate columns. One of the simplest approaches would be to train a model for each pricing subproblem to predict whether it gives a new column or not and only solve those that are predicted to give new columns. Assume that there are some pricing subproblems that do not give new columns. This is the case when the pricing subproblem solutions are identical to one of those obtained in previous iterations. The solutions of such pricing subproblems do not modify the RMP at all since the identical columns are already included in the RMP. Hence skipping these pricing subproblem does not affect the optimisation of the dual variables at all.

Whether this approach is effective or not depends on the property of the problem to be solved. If the pricing subproblems generate many duplicated columns, it is

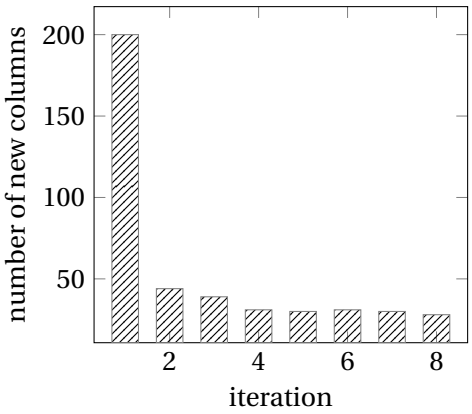

Figure 6.1: Number of new columns generated by the full column generation

of interest to avoid redundant computation. However, if the pricing subproblems generate few duplicated columns, then there is little interest to apply the aforementioned approach since all the pricing subproblems need to be solved. To examine the potential of the aforementioned approach, the number of new columns generated in each iteration is plotted in Figure 6.1. The result is for the full network-RMP method applied to one of the 200-generator test instances. Since there are no columns in the RMP at first, all the pricing subproblems generate new columns in the first iteration. However, after the first iteration, only approximately 30% of the pricing subproblems generate new columns in each iteration. This indicates that we may be able to save some computational time by avoiding redundant solution of the pricing subproblems.

Algorithm 4 shows the outline of our approach. Assume that we have a model to predict the number of iterations required a and probability p_{ki} of pricing subproblem i to give a new column in iteration k. In iteration 1 run the full column generation since all columns are new. Then in iteration $2,3,\ldots a-1$, we run the incremental column generation. If p_{ki} is larger than a prescribed threshold, we then solve pricing subproblem i in iteration k and skip it otherwise. In iteration a and all the subsequent iterations we run the full column generation and all the pricing subproblems are solved.

To predict whether a pricing subproblem gives a new column or not, we use a nearest neighbour approach. In the training phase the full-step column generation procedure is applied to as many training instances as possible for a predefined

Algorithm 4 Data-driven incremental column generation procedure

select suboptimality tolerance ϵ, probability thresholds $\bar{p}$, initial dual values y_0.
Initialise the dual values $y = y_0$, the lower and upper bound $l = -\infty, u = \infty$.
Predict the number of iterations required a.
for k in $\{1,2,\ldots\}$ **do**
  if $2 \le k \le a-1$ **then**
    // Run the incremental column generation procedure.
    for i in $\{1,2,\ldots,m\}$ **do**
      Predict probability p_{ki} of component i to give a new column.
      If $p_{ki} > \bar{p}$, **then** evaluate $q_i(y)$ and add a column to the RMP.
    end for
  else
    // Run the full-step column generation procedure.
    for i in $\{1,2,\ldots,m\}$ **do**
      Evaluate $q_i(y)$ and add a column to the RMP.
    end for
    Compute the lower bound $q(y)$ and update l.
  end if
  Update and solve the regularised RMP and set y to the dual solution.
  Run primal heuristics and update u.
  If $u - l \le \epsilon$, **then** terminate.
end for

number of iterations. For each training instance we record the number of iterations required to find a solution whose suboptimality is smaller than the prescribed tolerance (0.1% in our implementation). Furthermore, for each combination of training instance, iteration and pricing subproblem, we check whether the generated column is new or not.

Given an instance to be solved, we use the dataset of the training instances to predict the required number of iterations a and probability p_{ki} of pricing subproblem i to give a new column in iteration k as follows. First compare the problem parameters (i.e. demand) of the new instance with those in the training dataset and retrieve the closest N instances in Euclidean distance. Then take the average of the required number of iterations of the N nearest neighbours and use the average as the prediction of the number of required iteration a to solve the new instance. To predict probability p_{ki} for each iteration k and pricing subproblem i, find the number of nearest neighbours N_{ki} whose pricing subproblem i gave a new column in iteration k and use the ratio N_{ki}/N as the prediction.

6.2.1 Numerical experiments

Numerical experiments were conducted to evaluate the performance of the aforementioned approach. In the training phase we ran the full network-RMP method for 10 iterations on as many training instances as possible for 24 hours with 8 cores. Table 6.2 shows the number of training instances gathered in each case. To solve test instances we use $N = 50$ and threshold $\bar{p} = 0$. That is, we only skip pricing subproblem solution if it returned a duplicated column for all of the 50 nearest neighbours.

Table 6.2: Number of training instances

size	training instances
200	12,506
600	4,280
1000	2,756

The performance of the proposed approach was evaluated on the 40 test instances with suboptimality tolerance 0.1% as shown in Table 6.3. Every method successfully found a solution with suboptimality smaller than 0.1% on all the test instances within 10 minutes.

Table 6.3: Average iterations (column labelled as "iters"), number of subproblem solution ("eval") and computational time (s) with its breakdown

		summary			time breakdown			
size	method	iters	eval	total time	init.	RMP	subproblem	PH
200	network-network	3.0	600.0	**35.8**	0.0	0.1	12.3	23.5
	network-RMP	9.2	1845.0	42.2	0.0	0.5	37.4	4.4
	inc. network-RMP	10.1	1231.4	38.5	0.0	0.5	32.5	5.4
600	network-network	2.6	1530.0	76.5	0.0	0.3	27.5	48.8
	network-RMP	5.8	3510.0	70.5	0.0	1.2	64.2	5.1
	inc. network-RMP	6.0	2350.2	**57.7**	0.0	1.2	51.4	5.1
1000	network-network	2.0	2025.0	87.6	0.0	0.5	32.5	54.8
	network-RMP	4.6	4625.0	85.7	0.0	2.7	74.5	8.6
	inc. network-RMP	4.8	3524.4	**75.7**	0.0	2.7	64.4	8.6

Compared to the full network-RMP method, the incremental network-RMP method has shorter computational time while the incremental method requires slightly more iterations than the full-step method. Recall that if every pricing sub-

problem which gives a new column is solved, the incremental method computes the same dual values as the full column generation and requires the same number of iterations. The discrepancy in the numbers of iterations between the two methods indicates that the incremental method sometimes skips pricing subproblems which would give new columns. However, such errors were not observed frequently and their effect on the number of iterations required were on average minimal. On the other hand, as shown in the column labelled as "eval", there is a significant difference between the two methods in the number of pricing subproblem solutions: The incremental method requires a much smaller number of pricing subproblem to be solved, and by solving fewer pricing subproblems the incremental method requires significantly less computational time.

We note that the behaviour of the incremental regularised cutting-plane method is significantly different from that of the incremental subgradient method. The incremental subgradient method typically requires more iterations than the full-step method. However, since the number of component evaluations per iteration is smaller, the method requires fewer component evaluations in total and as a result the method finds a near optimal solution in a shorter time. The incremental regularised cutting-plane method requires almost the same number of iterations as the full-step method. The method is accelerated since it skips component evaluation which seems redundant.

The result of this is that when the tolerance is 0.1% the incremental network-RMP method is the fastest on test instances of size 600 and 1000. However, on the 200-generator test instances the network-network method slightly better than the network-RMP method. The incremental network-RMP method is faster than the full network-RMP, but the full network-network method is faster than both of them.

We note that the network-network method requires fewer iterations and function evaluations even on 600- and 1000-cases where the method is slower than the others. This is because the network-network method uses the neural network partial-fixing primal heuristic, which takes considerably longer time than the RMP partial-fixing primal heuristic. The RMP partial-fixing solves a partially-fixed UC instance in each iteration but the solution time of such an instance is very short. On the other hand, the neural network partial-fixing primal heuristic solves a much harder instance and

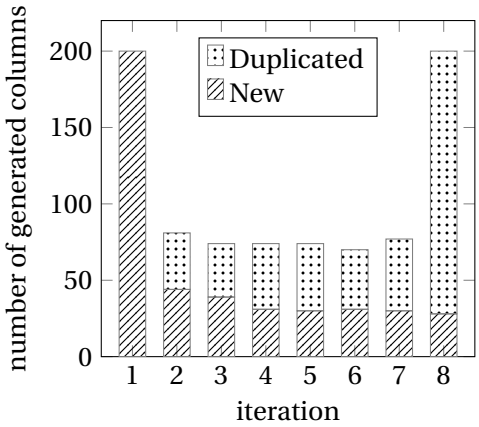

Figure 6.2: Number of new and duplicated columns generated by the incremental method. The model predicted the problem can be solved (the gap can be closed) in iteration 8 so all the subproblems are evaluated in the 8 iteration to compute a lower bound.

spends all the allocated time (4.2) in each iteration.

To gain better understanding of the performance of the incremental column generation, the number of pricing subproblems solved in each iteration is shown in Figure 6.2. A single test instance of size 200 is used to produce this plot. The model predicts that this test instance is likely to be solved in the 8th iteration. Thus, the algorithm runs the incremental method until the 7th iteration and switches to the full-step method in the 8th iteration. Between iteration 2 and 7, it only solved around 80 pricing subproblems per iteration. On this instance, indeed tight lower and upper bounds are found in the 8th iteration and the algorithm is terminated.

For each generated column we examined which were new and which were duplicated. Figure 6.2 shows the numbers of new and duplicated columns by iteration. In iteration 2 to 7 only about half of the subproblems solved gave new columns, so there remains some space for improvement in the prediction model. After solving the test instance, we also solved the pricing subproblems which were skipped by the incremental method and examined whether they actually gave duplicated columns or not. On this test instance, none of the skipped pricing subproblems indeed generated new columns.

Chapter 7

Conclusions

This section reviews the earlier chapters and draws conclusions of this thesis. Section 7.1 summarises the contents and conclusions of each chapter and Section 7.2 discusses further research.

7.1 Summary of the contents and conclusions

Chapter 2 described the UC problem and its solution methods. Lagrangian relaxation was studied and the subgradient methods and the cutting-plane methods were considered as solution methods for the dual problem. Then the duality between the cutting-plane methods and Dantzig-Wolfe decomposition with column generation procedures was discussed. The chapter was concluded with some examples to demonstrate the properties of the dual problem.

Chapter 3 studied approaches to compute the initial dual values to warmstart the column generation procedure. An analysis on the strength of the initialisation method based on the LPR was given, which was followed by numerical examples of small UC instances to verify the results. Then the initialisation method based on a neural network and an approach to train a model was introduced. The training was done efficiently by exploiting the decomposable structure of the UC problem. The following numerical experiments showed that once the training was completed, the initialisation method based on a neural network was able to find dual values that yielded a tight lower bound. The initialisation method yielded equally good or

even slightly better lower bounds than the method based on the LPR. What's more, in contrast to the method based on the LPR, the method based on a neural network output dual values instantly even on large instances. We also used the dual values output from the neural network to warmstart the column generation procedure. The overall computational time was successfully reduced compared with the initialisation method based on the LPR.

In Chapter 4 primal heuristics were examined. First some existing primal heuristics were reviewed and the RMP partial-fixing primal heuristic, a novel primal heuristic integrated with decomposition, was introduced. The behaviour and performance of these primal heuristics were compared on small examples. One of the most notable takeaway is that none of the primal heuristics always outperforms the others. In other words, for each primal heuristic we found an UC instance on which the other primal heuristics worked better. After the examples the neural network partial-fixing primal heuristic, a primal heuristic based on a neural network, was proposed. All the above primal heuristics were evaluated on practical, large-scale test instances. The numerical experiments showed that the RMP partial-fixing was able to find a primal feasible solution of suboptimality less than 0.1% in a short time. When the target tolerance is 0.5% or larger, the neural network partial-fixing primal heuristic was superior.

We also considered the situation where we were only interested in the best primal solution found in a prescribed time budget. The neural network partial-fixing primal heuristic was used as a standalone method without Dantzig-Wolfe decomposition and typically outperformed other primal heuristics when the timelimit was tight (2 minutes or less). If the timelimit was long, such as 5 minutes, the RMP partial-fixing primal heuristic often performed better.

Chapter 5 studied an extension of the column generation procedure to handle incremental column generation. To simplify the presentation, we discussed the dual of the column generation procedure, the cutting-plane methods applied to the dual problem. The analysis showed the worst-case suboptimality of the incremental method might suffer if the step size (regularisation parameter) was kept bounded both from above and from below. However, the convergence in the function value was shown to be guaranteed when diminishing step size rules are used. One notable

feature of our analysis is that the results hold when all cuts are kept but also for any strategy of deleting cuts as long as the latest cut are kept in the model. Such cut deletion was done heuristically with little theoretical justification. The following numerical experiments showed the effectiveness of the incremental method to optimise the dual function of large-scale UC instances. The incremental method greatly improved the stability of the iterates especially in the early iterations, and this led to a reduction in the overall computational time.

Finally, Chapter 6 considered integration of the aforementioned techniques. First, combination of the initialisation method based on a neural network and the two novel primal heuristics was studied. The numerical experiments showed that the novel primal heuristics usually works better with the initialisation method based on a network network compared with the initialisation methods based on the LPR. For example, when the tolerance is tight, i.e. 0.1%, the combination of the initialisation method based on a neural network and the RMP partial-fixing primal heuristic often performs best. Finally, the incremental column generation was used within this method. A data-driven approach to select subproblems to be solved was proposed. By skipping presumably redundant solution of pricing subproblems the column generation procedure was successfully accelerated and found a feasible solution with suboptimality smaller than 0.1% in a shorter time than the full-step counterpart.

7.2 Further research

The experiments conducted in this thesis are preliminary. A more comprehensive computational study is required to see the performance of the proposed methods in a more practical setting. For example, the solver for the pricing subproblem has a non-negligible impact on the overall performance. Combining the proposed methods with tailored subproblem solvers (see Section 2.5) is of interest but left as a future work. It is also of interest to compare the performances of the proposed primal heuristics with those proposed recently (see Section 4.2).

This thesis focused on the deterministic UC problem, which assumes that the demand is given. Recently, there is a significant interest to extend the UC problem to incorporate more realistic factors. For example, the stochastic UC problem assumes

that multiple forecasts of demand are given instead of a single, accurate forecast. These problems are modelled as two-stage or multi-stage stochastic programmes. Another example is a security-constrained UC problem. In this extension, we seek a solution which remains feasible even after some contingency happens, such as transmission failure.

It is of interest to apply the techniques considered in this thesis to improve solution methods for a UC problem with such extensions. Typical solution methods to stochastic or security-constrained UC problem are decomposition-based: the problem is split into multiple deterministic UC problems and they are solved repeatedly. For example, one typical solution method to the stochastic UC is scenario decomposition, which relax the non-anticipativity constraints to bind decision variables across multiple scenarios. In this approach the deterministic UC problems are solved with perturbed cost coefficient repeatedly. Some of the techniques studied in this thesis are readily applicable to deterministic UC problems, while others require some modification to handle the perturbation in the cost.

References

[1] J. F. Bard. “Short-Term Scheduling of Thermal-Electric Generators Using Lagrangian Relaxation”. In: *Operations Research* 36.5 (Sept. 1988), pp. 756–766 (cit. on p. 4).

[2] C. Barnhart et al. “Branch-and-Price: Column Generation for Solving Huge Integer Programs”. In: *Operations Research* 46.3 (May 1998), pp. 316–329 (cit. on p. 5).

[3] A. Beck. *First-Order Methods in Optimization*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2017. DOI: 10.1137/1.9781611974997 (cit. on pp. 17, 81, 82).

[4] I. Bello et al. “Neural Combinatorial Optimization with Reinforcement Learning”. In: *CoRR* abs/1611.09940 (2016). arXiv: 1611.09940 (cit. on p. 50).

[5] Y. Bengio, A. Lodi, and A. Prouvost. “Machine Learning for Combinatorial Optimization: A Methodological Tour d’Horizon”. In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421. DOI: https://doi.org/10.1016/j.ejor.2020.07.063 (cit. on p. 2).

[6] D. Bertsekas et al. “Optimal Short-term Scheduling of Large-Scale Power Systems”. In: *IEEE Transactions on Automatic Control* 28.1 (1983), pp. 1–11 (cit. on pp. 3, 4, 54).

[7] D. P. Bertsekas. *Convex Optimization Algorithms*. Belmont, Massachusetts: Athena Scientific, 2015 (cit. on p. 15).

[8] D. P. Bertsekas. *Convex Optimization Theorem*. Belmont, Massachusetts: Athena Scientific, 2009 (cit. on p. 35).

[9] D. P. Bertsekas. “Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey”. In: *Optimization for Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2011, pp. 85–119 (cit. on pp. 78, 94).

[10] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Optimization and neural computation series ; 3. Belmont, Massachusetts: Athena Scientific, 1996 (cit. on p. 78).

[11] D. Bertsimas and B. Stellato. “Online Mixed-Integer Optimization in Milliseconds”. In: (2021). arXiv: 1907.02206 (cit. on p. 50).

[12] D. Bertsimas and B. Stellato. “The Voice of Optimization”. In: (2020). arXiv: 1812.09991 (cit. on p. 50).

[13] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. New York: Springer, 2006 (cit. on p. 67).

[14] J. F. Bonnans et al. *Numerical Optimization Theoretical and Practical Aspects*. Second edition. Universitext. Berlin: Springer, 2006 (cit. on pp. 34, 79).

[15] A. Borghetti et al. "Lagrangian Heuristics Based on Disaggregated Bundle Methods for Hydrothermal Unit Commitment". In: *IEEE Power Engineering Review* 22.12 (2002), pp. 60–60 (cit. on p. 33).

[16] O. Briant et al. "Comparison of Bundle and Classical Column Generation". In: *Mathematical programming* 113.2 (2008), pp. 299–344 (cit. on pp. 5, 16, 44).

[17] E. W. Cheney and A. A. Goldstein. "Newton's Method for Convex Programming and Tchebycheff Approximation". eng. In: *Numerische Mathematik* 1.1 (1959), pp. 253–268. DOI: https://doi.org/10.1007/BF01386389 (cit. on p. 16).

[18] G. Dantzig and P. Wolfe. "Decomposition Principle for Linear Programs". In: *Operations Research* 8.1 (1960), p. 101 (cit. on p. 20).

[19] S. De and S. Smith. "Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 19964–19975 (cit. on p. 44).

[20] L. Dubost, R. Gonzalez, and C. Lemaréchal. "A Primal-Proximal Heuristic Applied to the French Unit-Commitment Problem". In: *Mathematical programming* 104.1 (2005), pp. 129–151 (cit. on p. 55).

[21] N. Freitas E. Brochu V. M. Cora. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (2010). arXiv: 1012.2599 (cit. on p. 34).

[22] G. Emiel and C. Sagastizábal. "Incremental-Like Bundle Methods with Application to Energy Planning". In: *Computational optimization and applications* 46.2 (2010), pp. 305–332. DOI: 10.1007/s10589-009-9288-8 (cit. on p. 79).

[23] W. Fan, X. Guan, and Q. Zhai. "A New Method for Unit Commitment with Ramping Constraints". In: *Electric power systems research* 62.3 (2002), pp. 215–224 (cit. on p. 23).

[24] S. Feltenmark and C. K. Kiwiel. "Dual Applications of Proximal Bundle Methods, Including Lagrangian Relaxation of Nonconvex Problems". eng. In: *SIAM journal on optimization* 10.3 (2000), pp. 697–721 (cit. on p. 55).

[25] A. Frangioni and C. Gentile. "Solving Nonlinear Single-Unit Commitment Problems with Ramping Constraints". In: *Operations research* 54.4 (2006), pp. 767–775 (cit. on pp. 23, 43).

[26] L. L. Garver. "Power Generation Scheduling by Integer Programming — Development of Theory". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 81.3 (1962), pp. 730–734. DOI: 10.1109/AIEEPAS.1962.4501405 (cit. on p. 1).

[27] X. Glorot and Y. Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Journal of Machine Learning Research - Proceedings Track* 9 (Jan. 2010), pp. 249–256 (cit. on p. 44).

[28] J. Gruhl, F. Schweppe, and M. Ruane. “Unit Commitment Scheduling of Electric Power Systems”. In: *Systems Engineering for Power: Status and Prospects*. Aug. 1975, pp. 116–129 (cit. on p. 2).

[29] X. Guan et al. “An Optimization-Based Method for Unit Commitment”. In: *International journal of electrical power & energy systems* 14.1 (1992), pp. 9–17 (cit. on pp. 5, 43, 50, 52).

[30] J. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms I*. Grundlehren der mathematischen Wissenschaften 305. Berlin: Springer, 1993. DOI: 10.1007/978-3-662-06409-2 (cit. on p. 83).

[31] J. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II*. Grundlehren der mathematischen Wissenschaften 306. Berlin: Springer, 1993. DOI: 10.1007/978-3-662-06409-2 (cit. on p. 79).

[32] B. F. Hobbs et al. “Why this Book? New Capabilities and New Needs for Unit Commitment Modeling”. In: *The Next Generation of Electric Power Unit Commitment Models*. Ed. by B. F. Hobbs et al. Boston, MA: Springer US, 2001, pp. 1–14. DOI: 10.1007/0-306-47663-0_1 (cit. on pp. 1, 2).

[33] D. Jones, M. Schonlau, and W. Welch. “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492 (cit. on p. 34).

[34] J. E. Kelley Jr. “The Cutting-Plane Method for Solving Convex Programs”. In: *Journal of the Society for Industrial & Applied Mathematics* 8.4 (1960), pp. 703–712 (cit. on p. 16).

[35] E. Khalil et al. “Learning Combinatorial Optimization Algorithms over Graphs”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 6348–6358 (cit. on p. 50).

[36] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *The International Conference on Learning Representations*. ULR https://arxiv.org/abs/1606.01885. May 2015 (cit. on p. 44).

[37] K. C. Kiwiel. “A Dual Method for Certain Positive Semidefinite Quadratic-Programming Problems”. In: *SIAM journal on scientific and statistical computing* 10.1 (1989), pp. 175–186 (cit. on p. 56).

[38] B. Knueven, J. Ostrowski, and J. Watson. “On Mixed-Integer Programming Formulations for the Unit Commitment Problem”. In: *INFORMS Journal on Computing* 32.4 (2020), pp. 857–876 (cit. on p. 8).

[39] W. Kool, H. van Hoof, and M. Welling. “Attention, Learn to Solve Routing Problems!” In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on p. 50).

[40] G. S. Lauer et al. “Solution of Large-Scale Optimal Unit Commitment Problems”. eng. In: *IEEE Transactions on Power Apparatus and Systems* PAS-101.1 (1982), pp. 79–86 (cit. on p. 4).

[41] C. Lemaréchal and A. Renaud. "A Geometric Study of Duality Gaps, with Applications". In: *Mathematical programming* 90.3 (2001), pp. 399–427. DOI: https://doi.org/10.1007/PL00011429 (cit. on p. 5).

[42] J. Linderoth and S. Wright. "Decomposition Algorithms for Stochastic Programming on a Computational Grid". eng. In: *Computational Optimization and Applications* 24.2 (2003), pp. 207–250 (cit. on p. 79).

[43] P. G. Lowery. "Generating Unit Commitment by Dynamic Programming". In: *IEEE Transactions on Power Apparatus and Systems* PAS-85.5 (1966), pp. 422–426. DOI: 10.1109/TPAS.1966.291679 (cit. on p. 3).

[44] M. Madrigal and V. H. Quintana. "An Interior-Point/Cutting-Plane Method to Solve Unit Commitment Problems". In: *IEEE Transactions on Power Systems* 15.3 (2000), pp. 1022–1027 (cit. on p. 5).

[45] N. Mazzi et al. "Benders Decomposition with Adaptive Oracles for Large Scale Optimization". eng. In: *Mathematical Programming Computation* 13.4 (2020), pp. 683–703 (cit. on p. 80).

[46] A. Merlin and P. Sandrin. "A New Method for Unit Commitment at Electricite De France". In: *IEEE Transactions on Power Apparatus and Systems* PAS-102.5 (1983), pp. 1218–1225. DOI: 10.1109/TPAS.1983.318063 (cit. on pp. 4, 50, 52).

[47] J. A. Muckstadt and S. A. Koenig. "An Application of Lagrangian Relaxation to Scheduling in Power-Generation Systems". In: *Operations Research* 25.3 (1977), pp. 387–403 (cit. on p. 3).

[48] V. Nair et al. "Learning Fast Optimizers for Contextual Stochastic Integer Programs". In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*. 2018, pp. 591–600 (cit. on pp. 34, 41, 44, 50).

[49] M. Nazari et al. "Reinforcement Learning for Solving the Vehicle Routing Problem". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 9839–9849 (cit. on p. 50).

[50] A. Nedic and D. P. Bertsekas. "Incremental Subgradient Methods for Nondifferentiable Optimization". In: *SIAM Journal on Optimization* 12.1 (2001), pp. 109–138. DOI: 10.1137/S1052623499362111 (cit. on pp. 78, 81, 88, 93).

[51] A. Nedić. "Subgradient Methods for Convex Minimization". PhD thesis. Massachusetts Institute of Technology, 2002 (cit. on pp. 88–90).

[52] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. New York: Wiley, 1988 (cit. on p. 13).

[53] E. S. H. Neto and A. R. Pierro. "Incremental Subgradients for Constrained Convex Optimization: A Unified Framework and New Methods". In: *SIAM journal on optimization* 20.3 (2009), pp. 1547–1572 (cit. on p. 78).

[54] S. Nowozin and C. H. Lampert. "Structured Learning and Prediction in Computer Vision". In: *Foundations and Trends in Computer Graphics and Vision* 6.3–4 (2011), pp. 185–365. DOI: 10.1561/0600000033 (cit. on p. 33).

[55] J. Ostrowski, M. F. Anjos, and A. Vannelli. "Tight Mixed Integer Linear Programming Formulations for the Unit Commitment Problem". In: *IEEE Transactions on Power Systems* 27.1 (2012), pp. 39–46 (cit. on p. 8).

[56] S. Pineda and J. M. Morales. "Is Learning for the Unit Commitment Problem a Low-Hanging Fruit?" In: (2021). arXiv: 2106.11687 (cit. on pp. 51, 67).

[57] S. S. Ram, A. Nedić, and V. V. Veeravalli. "Incremental Stochastic Subgradient Algorithms for Convex Optimization". eng. In: *SIAM journal on optimization* 20.2 (2009), pp. 691–717 (cit. on p. 78).

[58] N. J. Redondo and A. J. Conejo. "Short-Term Hydro-Thermal Coordination by Lagrangian Relaxation: Solution of the Dual Problem". In: *IEEE Transactions on Power Systems* 14.1 (1999), pp. 89–95 (cit. on p. 5).

[59] R. T. Rockafellar. *Convex Analysis*. Princeton mathematical series ; 28. Princeton, N.J: Princeton University Press, 1970 (cit. on p. 13).

[60] B. Saravanan et al. "A Solution to the Unit Commitment Problem — A Review". In: *Frontiers in Energy* 7.2 (2013), pp. 223–236 (cit. on p. 2).

[61] T. Schulze, A. Grothey, and K. McKinnon. "A Stabilised Scenario Decomposition Algorithm Applied to Stochastic Unit Commitment Problems". In: *European Journal of Operational Research* 261.1 (Aug. 2017), pp. 247–259. DOI: 10.1016/j.ejor.2017.02.005 (cit. on pp. 17, 33, 45, 53, 92).

[62] S. Takriti and J. R. Birge. "Using Integer Programming to Refine Lagrangian-Based Unit Commitment Solutions". In: *IEEE Transactions on Power Systems* 15.1 (2000), pp. 151–156 (cit. on pp. 43, 50, 53).

[63] S. Takriti, J. R. Birge, and E. Long. "A Stochastic Model for the Unit Commitment Problem". In: *IEEE Transactions on Power Systems* 11.3 (1996), pp. 1497–1508 (cit. on pp. 5, 33).

[64] W. van Ackooij and A. Frangioni. "Incremental Bundle Methods Using Upper Models". In: *SIAM Journal on Optimization* 28.1 (2018), pp. 379–410. DOI: 10.1137/16M1089897 (cit. on pp. 79, 92).

[65] F. Vanderbeck. "Implementing Mixed Integer Column Generation". In: *Column Generation*. Springer US, 2005, pp. 331–358 (cit. on p. 53).

[66] F. Vanderbeck and M. W. P. Savelsbergh. "A Generic View of Dantzig–Wolfe Decomposition in Mixed Integer Programming". In: *Operations Research Letters* 34.3 (2006), pp. 296–306 (cit. on p. 20).

[67] Q. Wang. "Knowledge-Based Approach for Dimensionality Reduction Solving Repetitive Combinatorial Optimization Problems". In: *Expert Systems with Applications* 184 (2021), p. 115502. DOI: https://doi.org/10.1016/j.eswa.2021.115502 (cit. on pp. 51, 52, 67).

[68] A. S. Xavier, F. Qiu, and S. Ahmed. "Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems". In: *INFORMS Journal on Computing* 0.0 (2020). DOI: https://doi.org/10.1287/ijoc.2020.0976 (cit. on pp. 51, 52, 67).

[69] F. Zhuang and F. D. Galiana. “Towards a More Rigorous and Practical Unit Commitment by Lagrangian Relaxation”. In: *IEEE Transactions on Power Systems* 3.2 (1988), pp. 763–773. DOI: 10.1109/59.192933 (cit. on pp. 4, 50).

List of Figures

List of Tables