UNIVERSITY OF ZAGREB

FACULTY OF SCIENCE

DEPARTMENT OF MATHEMATICS

Matej Jusup

NETWORK OPTIMIZATION IN RAILWAY TRANSPORT PLANNING

Master thesis

Mentor:
Prof. dr. sc. Marko Vrdoljak

Co-mentor:
Prof. dr. sc. Andreas Dress

Zagreb, February, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____ , predsjednik
2. _____ , član
3. _____ , član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Contents

Introduction

Motivation

During my studies, I got especially interested in the field of optimization immediately after my first optimization course. It was held by prof. dr. sc. Marko Vrdoljak whom I asked to be my master thesis mentor. After my first introductory optimization course, I took *Operations research* course which was held by prof. Vrdoljak as well. I saw its potential right from the beginning and after consulting with prof. Vrdoljak I decided what my topic will be. First article I read about railway planning was [6], where one can have a brief look at [1]. Problem with models introduced there is that they are mostly dealing with freight transportation in U.S..On the other hand, European railway networks are mostly dealing with a passenger transportation. Nevertheless, in [6] I found out that there are many things we should take care of while modelling railway system, e.g. *railway blocking problem*, *yard location problem*, *train scheduling problem*, *locomotive scheduling problem*, *train dispatching problem*, *crew scheduling problem* etc. Design of a complete railway system is highly complex and it is divided into sub-problems, some of which are mentioned above. These problems are not independent, indeed they highly influence each other, and it turns out that most of these problems are NP-hard (see [2], Appendix B: NP-completeness) and have a huge size. Therefore, I decided to focus my attention on *train scheduling problem* because, in my opinion, it is the most critical one. Solutions of the problems like *railway blocking problem* or *yard location problem* are hard to implement in practice, even though they can generate the biggest savings. This solutions would force countries to reallocate infrastructure in the existing railway system but, more likely than not, none of them is really willing for such a huge step (and often it is not even possible). On the other hand, *crew scheduling problem* is important one but it does not generate that high savings compared to some other problems. Thinking further one can conclude that *train scheduling problem* is in the hearth of railway system planning nowadays. It can cause substantial savings, it is not that hard to implement new timetables in the existing railway networks, most countries have issues with timetabling and it does not affect other sub-problems that much.

Pre-work assumptions

While looking for an appropriate model for describing *train timetabling problem*, we need to make many decisions, e.g. are we going to use single or double line model, continuous or discrete time variables, periodic or non-periodic timetable, in periodic case, which period is appropriate for Croatia, is model for a passenger transport going to be good enough to cover occurrence of freight trains etc. Decision about single line model is quite easy because most of European countries, including Croatia, have single line railway systems. Concerning time variable we will settle down for a discrete one because in practice arrival and departure times are always rounded in whole minute. Or in the worst case, in half a minute. We will work with periodic timetable because Croatian railways have one day period, except during the weekend but that is not causing too much difficulties. It is worth mentioning that periodical system offer faster computational time and therefore we have a huge benefit from it. Freight trains are also quite easily included in the model because in Croatia they usually have a fixed timetable like a passenger trains. Luckily for us there already exists one model which covers our reasoning and it was designed and tested for some of European railway systems. Details about this model (and much more), which we will present after theoretical discussion, can be found in [3].

Chapter 1

Basics of network flows

1.1 Introduction

Our goal is to present a solution for *train timetabling problem* and before developing any advanced mechanisms for doing that, we need to start from the basics. In this section, we will introduce the *"minimum cost flow problem"* which will be the first step in this process. The minimum cost flow problem is the most fundamental of all network flow problems. Its generalization, multi-commodity flow problem, which we will introduce later, is going to be our model for representing railway network. The minimum cost flow problem is easy to state: **We wish to determine the least cost shipment of a commodity through a network in order to satisfy demands at certain nodes from available supplies at other nodes.** We now present basic definitions of the graph theory, mathematical programming formulation of the minimum cost flow problem and some other technical results. Even though we will, at some point, say few words about linear programming and describe a simplex method into some depth, it will be mostly due to the notational purposes and for the sake of clarity and completeness of our work. Nevertheless, we are assuming that a reader is having a good understanding of linear programming, especially duality theory which we will not cover in this work.

1.2 Brief introduction to graph theory

Most of the theoretical results we are going to use in this work can be found in [2]. Particularly, this chapter is almost solely based on the following chapters:

1. Introduction

2. Paths, trees, and cycles

Only a few minor results are taken from some other chapters.

In this work, we will stick to the notation used in the above-mentioned book (see [2]). E.g. we will use arcs and nodes instead of edges and vertices, models with capacitated arcs and with exogenous supplies and demands at the nodes etc. Nevertheless, some minor adjustments were made because of [3] to have consistent notation through the whole work. In this section we have few objectives. Firstly, we will bring together some basic definitions of network flows and graph theory to have a brief summary of the full theoretical discussion which can be found in [2]. Secondly, we will discuss one of the ways to transform a network flow problem and obtain equivalent one. This equivalent model will be of a great value while dealing with multi-commodity flow problem (see section 2). Thirdly, we will describe a simplex method and its specialized version called a revised simplex method which we will need at some point.

Definition 1.2.1 (**Directed graphs and networks**). *A directed graph $G = (N, A)$ consists of a set N of nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. A directed network is a directed graph whose nodes and/or arcs have associated numerical values (typically costs, capacities and/or supplies and demands). We let n denote the number of nodes and m denote the number of arcs in G.*

From now on, we often make no distinction between graphs and networks, so we use terms *graph* and *network* synonymously.

Definition 1.2.2 (**Undirected graphs and networks**). *We define an undirected graph in the same manner as we define a directed graph except that arcs are unordered pairs of distinct nodes. In an undirected graph, we can refer to an arc joining the node pair i and j either as (i, j) or (j, i).*

One can imagine an undirected arc (i, j) as a two-way street. Following the analogy, a directed arc (i, j) can be viewed as a one-way street with entering point being node i and exit point node j. In this work, we assume that the underlying network is always directed one. Therefore, we present our subsequent notation and definitions for directed networks. The corresponding definitions for undirected networks should be transparent to a reader and we will only have a brief note about it at the end of the section.

Definition 1.2.3 (**Tails and heads**). *A directed arc (i, j) has two endpoints i and j. We refer to node i as the tail of arc (i, j) and node j as its head. We say that the arc (i, j) emanates from node i and terminates at node j. An arc (i, j) is incident to nodes i and j. The arc (i, j) is an outgoing arc of node i and an incoming arc of node j. Whenever an arc $(i, j) \in A$, we say that node j is adjacent to node i.*

Definition 1.2.4 (**Degrees**). *The indegree of a node is the number of incoming arcs of that node and its outdegree is the number of its outgoing arcs. The degree of a node is the sum of its indegree and outdegree.*

It is easy to see that the sum of indegrees of all nodes equals the sum of outdegrees of all nodes and both are equal to the number of arcs m in the network.

Definition 1.2.5 (**Adjacency list**). *The arc adjacency list $A(i)$ of a node i is the set of arcs emanating from that node, i.e., $A(i) = \{(i, j) \in A : j \in N\}$. The node adjacency list $A(i)$ is the set of nodes adjacent to that node; in this case, $A(i) = \{j \in N : (i, j) \in A\}$.*

Often, we shall omit the terms *arc* and *node* and simply refer to the adjacency list; in all cases it will be clear from the context whether we mean arc adjacency list or node adjacency list. We assume that arcs in the adjacency list $A(i)$ are arranged so that the head nodes of arcs are in increasing order. Notice that $|A(i)|$ equals the outdegree of node i. Since the sum of all node outdegrees equals m, we immediately obtain the following property:

Corollary 1.2.6. $\sum_{i \in N} |A(i)| = m$

Definition 1.2.7 (**Subgraph**). *A graph $G' = (N', A')$ is a subgraph of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. We say that $G' = (N', A')$ is the subgraph of G induced by N' if A' contains each arc of A with both endpoints in N'. A graph $G' = (N', A')$ is a spanning subgraph of $G = (N, A)$ if $N' = N$ and $A' \subseteq A$.*

Definition 1.2.8 (**Walk**). *A walk in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_1 - i_2 - a_2 - \cdots - i_{r-1} - a_{r-1} - i_r$ satisfying the property that for all $1 \leq k \leq r - 1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. Alternatively, we shall sometimes, for the sake of simplicity, refer to a walk as a set of (sequence of) arcs (or of nodes) without any explicit mention of the nodes (arcs).*

Definition 1.2.9 (**Directed walk**). *A directed walk is an oriented version of a walk in the sense that for any two consecutive nodes i_k and i_{k+1} on the walk, $(i_k, i_{k+1}) \in A$.*

Definition 1.2.10 (**Path**). *A path is a walk without any repetition of nodes. We can partition the arcs of a path into two groups: forward arcs and backward arcs. An arc (i, j) in the path is forward arc if the path visits node i prior to visiting node j, and is a backward arc otherwise.*

Definition 1.2.11 (**Directed path**). *A directed path is a directed walk without any repetition of nodes. In other words, a directed path has no backward arcs.*

Definition 1.2.12 (**Cycle**). *A cycle is a path $i_1 - i_2 - \ldots - i_r$ together with the arc (i_r, i_1) or (i_1, i_r). We shall often refer to a cycle using the notation $i_1 - i_2 - \ldots - i_r - i_1$. Just as we did for paths, we can define forward and backward arcs in a cycle.*

Definition 1.2.13 (**Directed cycle**). *A directed cycle is a directed path $i_1 - i_2 - \ldots - i_r$. together with the arc (i_r, i_1).*

Definition 1.2.14 (**Acyclic network**). *A graph is acyclic if it contains no directed cycle.*

Remark 1.2.15 (**Definitions for undirected networks**). *Definitions for directed networks easily translate into those for undirected networks. An undirected arc (i, j) has two endpoints, i and j, but its tail and head nodes are undefined. If the network contains the arc (i, j), node i is adjacent to node j, and node j is adjacent to node i. The arc adjacency list (as well as the node adjacency list) is defined similarly except that arc (i, j) appears in $A(i)$ as well as in $A(j)$. Consequently, $\sum_{i \in N} |A(i)| = 2m$. The degree of a node is the number of nodes adjacent to node i. Each of the graph theoretic concepts we have defined has essentially the same definition for undirected networks except that we do not distinguish between a path and a directed path, a cycle and a directed cycle, and so on.*

Now we are ready to introduce the *"minimum cost flow problem"*

1.3 Minimum cost flow problem

Let $G = (N, A)$ be a directed network defined by a set N of *n nodes* and a set A of *m directed arcs*. Each arc $(i, j) \in A$ has an associated *cost c_{ij}* that denotes the cost per unit flow on that arc. We assume that the flow cost varies linearly with the amount of flow. With each arc $(i, j) \in A$ we also associate a *capacity u_{ij}* that denotes the maximum amount that can flow on the arc and a *lower bound l_{ij}* that denotes the minimum amount that must flow on the arc. With each node $i \in N$ we associate an integer number $b(i)$ representing its supply/demand. If $b(i) > 0$, node i is a *supply node*; if $b(i) < 0$, node i is a *demand node* with a demand of $-b(i)$; and if $b(i) = 0$, node i is a *transshipment node*. The decision variables in the minimum cost flow problem are arc flows and we represent the flow on an arc $(i, j) \in A$ by x_{ij}. In the above formulation of network flow we defined flows on arcs. In subsection 1.4 we will introduce equivalent, flows on paths and cycles formulation.

The minimum cost flow problem is an optimization model formulated as follows:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \qquad (1.1)$$

subject to

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = b(i), \quad \forall i \in N, \qquad (1.2)$$

$$l_{ij} \leq x_{ij} \leq u_{ij}, \quad \forall(i,j) \in A, \qquad (1.3)$$

where $\sum_{i=1}^{n} b(i) = 0$. In matrix form, we represent the minimum cost flow problem as follows:

$$\min cx \qquad (1.4)$$

subject to

$$\mathcal{N}x = b, \qquad (1.5)$$

$$l \leq x \leq u. \qquad (1.6)$$

In this formulation, $\mathcal{N}$ is an $n \times m$ matrix, called *node-arc incidence matrix* of the minimum cost flow problem. Each column $\mathcal{N}_{ij}$ in the matrix corresponds to the variable x_{ij}. The column $\mathcal{N}_{ij}$ has a $+1$ in the ith row, a -1 in the jth row and the rest of its entries are 0.

We refer to the constraints (1.2) as a *mass balance constraints*. The first term in this constraint for a node represents the total *outflow* of the node (i.e., the flow emanating from the node) and the second term represents the total *inflow* of the node (i.e., the flow entering the node). The mass balance constraint states that the outflow minus inflow must equal the supply/demand of the node. If the node is a supply node, its outflow exceeds its inflow; if the node is a demand node, its inflow exceeds its outflow; and if the node is a transshipment node, its outflow equals its inflow. The flow must also satisfy the lower bound and capacity constraints (1.3), which we refer to as *flow bound constraints*. In most applications, the lower bounds on arc flows are zero and from this point on, we will assume that as well. If not stated otherwise, we will assume *integrality*, i.e. all the data are integral (all arc capacities, arc costs and supplies/demands of nodes). Nevertheless, it is always possible to transform rational data to integer data by multiplying them by a suitably large number.

Remark 1.3.1. *In the applications, we necessarily need to convert irrational numbers to rational numbers to be able to save them in a database.*

Remark 1.3.2. *Biggest the network is, more zero coefficients node-arc incidence matrix $\mathcal{N}$ will have. Furthermore, a ratio of nonzero and zero coefficients will be really small. Therefore, the incidence matrix representation of a network flow is not space efficient for saving in database. Because of its inefficiency, it rarely produces efficient algorithms and there are a better ways of doing it in practice. Some introduction can be found in [2],*

but nowadays with the improvement of technology, there are even more advanced ways of doing it. Nevertheless, incidence matrix is of a great value when it comes to the theory because it represents the constraint matrix of the minimum cost flow problem, as well as it possesses some useful properties.

In [2], chapters *9. Minimum cost flows: Basic algorithms*, *10. Minimum cost flows: Polynomial algorithms* and *11. Minimum cost flows: Network simplex algorithms*, one can find an in-depth discussion about the *minimum cost flow problem*.

Shortest path problem

The shortest path problem is perhaps the simplest of all network flow problems. For this problem we wish to find a path of minimum cost (or length) from a specified *source node* s to another specified *sink node* t, assuming that each arc $(i, j) \in A$ has an associated cost (or length) c_{ij}. Some of the simplest applications of the shortest path problem are to determine a path between two specified nodes of a network that has a minimum length, or a path that takes the least time to traverse, or a path that has the maximum reliability. Regardless its simplicity, this basic model will occur as a part of our algorithm for solving multi-commodity flow problem (see section 2).

If we set $b(s) = 1$, $b(t) = -1$, and $b(i) = 0$ for all other nodes in the minimum cost flow problem (see section 1.3), the solution to the problem will send 1 unit of flow from node s to node t along the shortest path. The shortest path problem also models situations in which we wish to send flow from a single-source node to a single-sink node in an uncapacitated network. That is, if we wish to send v units of flow from node s to node t and the capacity of each arc of the network is at least v, we would send the flow along the shortest path from node s to node t. If we want to determine shortest paths from the source node s to every other node in the network, then in the minimum cost flow problem we set $b(s) = (n - 1)$ and $b(i) = -1$ for all other nodes (we can set each arc capacity u_{ij} to any number larger than $(n - 1)$). The minimum cost flow solution would then send unit flow from node s to every other node i along the shortest path. In [2] chapters *4. Shortest paths: Label-setting algorithms* and *5. Shortest paths: Label-correcting algorithms* give a deeper look at this important problem.

Before moving forward to the main results of this section, we will first describe three techniques of network transformations which we will use on a few occasions later on.

Node splitting transformation

The node splitting transformation splits each node i into two nodes i' and i'' corresponding to the node's *output* and *input* functions. This transformation replaces each original arc (i, j) by an arc (i', j'') of the same cost and capacity. It also adds an arc (i'', i') of zero

cost and with infitinite capacity for each *i*. The input side of node *i* (i.e., node *i''*) receives all the node's inflow, the output side (i.e., node *i'*) sends all the node's outflow, and the additional arc (i'', i') carries flow from the input side to the output side. We define the supplies/demands of nodes in the transformed network in accordance with the following three cases:

1. If $b(i) > 0$, then $b(i'') = b(i)$ and $b(i') = 0$.
2. If $b(i) < 0$, then $b(i'') = 0$ and $b(i') = b(i)$.
3. If $b(i) = 0$, then $b(i') = b(i'') = 0$.

It is easy to show a one-to-one correspondence between a flow in the original network and the corresponding flow in the transformed network; moreover, the flows in both networks have the same cost.

Reduced costs

In many of the network flow algorithms, we measure the cost of an arc relative to "imputed" costs associated with its incident nodes. These imputed costs typically are intermediate data that we compute within the context of an algorithm. Later on, in section 2.4, reduced costs will play one of the main roles in our algorithm. Because of its importance for us, we will introduce few basic results into some detail and whole discussion regarding reduced costs one can find in [2].

Definition 1.3.3 (**Reduced cost**). *Suppose that with each node $i \in N$ we associate a number $\pi(i)$, which we refer to as the potential of that node. With respect to the node potentials $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, we define the reduced cost c_{ij}^{π} of an arc (i, j) as*

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j). \tag{1.7}$$

In our algorithm in section 2.4 and many others, we often work with reduced costs c_{ij}^{π} instead of the actual cost c_{ij}. Consequently, it is important to understand the relationship between the objective function $z(\pi) = \sum_{(i,j) \in A} c_{ij}^{\pi} x_{ij}$ and $z(0) = \sum_{(i,j) \in A} c_{ij} x_{ij}$. Suppose, initially, that $\pi = 0$ and we then increase the node potential of node *k* to $\pi(k)$. (1.7) implies that this change reduces the reduced cost of each unit of flow leaving node *k* by $\pi(k)$. Thus the total decrease in the objective function equals $\pi(k)$ times the outflow of node *k* minus $\pi(k)$ times the inflow of node *k*. By definition, the outflow minus inflow equals supply/demand of the node. Consequently, increasing the potential of node *k* by $\pi(k)$ decreases the objective function value by $\pi(k)b(k)$ units. Repeating this argument iteratively for each node establishes that

$$z(0) - z(\pi) = \sum_{i \in N} \pi(i)b(i) = \pi b$$

For a given node potential π, πb is a constant. Therefore, a flow that minimizes $z(\pi)$ also minimizes $z(0)$. We formalize this result.

Proposition 1.3.4. *The minimum cost flow problems with arc costs c_{ij} or c_{ij}^π have the same optimal solutions. Moreover, $z(\pi) = z(0) - \pi b$*

We next study the effect of working with reduced costs on the cost of cycles and paths. Let W be a directed cycle in G. Then

$$\sum_{(i,j)\in W} c_{ij}^\pi = \sum_{(i,j)\in W} (c_{ij} - \pi(i) + \pi(j))$$

$$= \sum_{(i,j)\in W} c_{ij} + \sum_{(i,j)\in W} (\pi(j) - \pi(i))$$

$$= \sum_{(i,j)\in W} c_{ij}.$$

The last equality follows from the fact that for any directed cycle W, the expression $\sum_{(i,j)\in W} (\pi(j) - \pi(i))$ sums to a zero because for each node i in the cycle W, $\pi(i)$ occurs once with a positive sign and once with a negative sign. Similarly, if P is a directed path from node k to node l, then

$$\sum_{(i,j)\in P} c_{ij}^\pi = \sum_{(i,j)\in P} (c_{ij} - \pi(i) + \pi(j))$$

$$= \sum_{(i,j)\in P} c_{ij} - \sum_{(i,j)\in P} (\pi(i) - \pi(j))$$

$$= \sum_{(i,j)\in P} c_{ij} - \pi(k) + \pi(l),$$

because all $\pi(\cdot)$ corresponding to the nodes in the path, other than the terminal nodes k and l, cancel each other in the expression $\sum_{(i,j)\in P} (\pi(i) - \pi(j))$. We record these results.

Proposition 1.3.5.

a) *For any directed cycle W and for any node potentials π, $\sum_{(i,j)\in W} c_{ij}^\pi = \sum_{(i,j)\in W} c_{ij}$.*

b) *For any directed path P from node k to node l and for any node potentials π, $\sum_{(i,j)\in P} c_{ij}^\pi = \sum_{(i,j)\in P} c_{ij} - \pi(k) + \pi(l)$.*

Residual networks

In designing, developing, and implementing network flow algorithms, it is often convenient to measure flow not in absolute terms, but rather in terms of incremental flow about some given feasible solution. Typically, the solution at some intermediate point in an algorithm. Doing so leads us to define a new, ancillary network, known as the *residual network*, that functions as a "remaining flow network" for carrying the incremental flow. Here we will just introduce basic idea behind residual networks, define it, and state the most important result. Nevertheless, it can be shown that formulations of the problem in the original network and in the residual network are equivalent in the sense that they give a one-to-one correspondence between feasible solutions to two problems that preserve the value of the cost of solutions. Details can be found in [2].

The concept of a residual network is based on the following intuitive idea. Suppose that arc (i, j) carries x_{ij}^0 units of flow. Then we can send an additional $u_{ij} - x_{ij}^0$ units of flow from node i to node j along arc (i, j). Also notice that we can send up to x_{ij}^0 units of flow from node j to node i over the arc (i, j), which amounts to cancelling the existing flow on the arc. Whereas sending a unit flow from node i to j on arc (i, j) increases the flow cost by c_{ij} units, sending flow from node j to node i on the same arc decreases the flow cost by c_{ij} units (since we are saving the cost we used to incur in sending the flow from node i to node j). Using these ideas, we (algorithmically) define the residual network with respect to a given flow x^0 as follows:

Definition 1.3.6 (**Residual network**). *We replace each arc (i, j) in the original network with two arcs, (i, j) and (j, i). The arc (i, j) has cost c_{ij} and residual capacity $r_{ij} = u_{ij} - x_{ij}^0$, and the arc (j, i) has cost $-c_{ij}$ and residual capacity $r_{ji} = x_{ij}^0$. The residual network consists only of the arcs with a positive residual capacity and we use the notation $G(x^0)$ to represent the residual network corresponding to the flow x^0.*

We will next just state the most important result concerning residual networks and use it without a proof.

Lemma 1.3.7. *A flow x is a feasible flow in the network G if and only if its corresponding flow x', defined by $x_{ij}' - x_{ji}' = x_{ij} - x_{ij}^0$ and $x_{ij}'x_{ji}' = 0$, is feasible in the residual network $G(x^0)$. Furthermore, $cx = c'x' + cx^0$, where c' is the price vector on arcs in the residual network.*

Remark 1.3.8. *One important consequence of the lemma 1.3.7 is the flexibility it provides us. Instead of working with the original network G, we can work with the residual network $G(x^0)$ for some x^0. Once we have determined an optimal solution in the residual network, we can immediately convert it into an optimal solution in the original network.*

Now we are ready to introduce flow on paths and cycles formulation of network flows.

1.4   Equivalent representation of network flows

In formulating network flows problems, we can adopt either of two equivalent modelling approaches. We can define flow on arcs (as discussed in section 1.3) or define flow on paths and cycles. The first approach is more common, more intuitive and easier to visualize but while dealing with multi-commodity flow problem using column generation technique, the second approach will play a central role. Therefore, in this section, we will state and prove flow decomposition theorem. The proof is going to be algorithmic and using it, from flow on arcs representation we can get flow on paths and cycles representation and vice versa.

In a discussion to follow, by an *arc flow* we mean a vector $x = x_{ij}$ which satisfies the following constraints:

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = -e(i) \quad \forall i \in N, \tag{1.8}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A. \tag{1.9}$$

where $\sum_{i=1}^{n} e(i) = 0$. Notice that in this model we have replaced the supply/demand $b(i)$ of node i by another term, $-e(i)$; we refer to $e(i)$ as the node's *imbalance*. Reasons for this change would lead us outside the scope of this work. The main point being that some of the algorithms for solving the minimum cost flow problem require this change because sometimes we want to consider flows that are not feasible. Since, at some point, one might need to use these algorithms we will stick to this notation while talking about flow on paths and cycles representation. Details about it can be found in [2].

Term $e(i)$ represents the inflow minus outflow of node i. If the inflow is more than outflow, $e(i) > 0$ and we say that the node i is an *excess node*. If inflow is less than the outflow, $e(i) < 0$ and we say that node i is a *deficit node*. If the inflow equals outflow, we say that node i is a *balanced node*. Observe that if $e = -b$, the flow x is feasible for the minimum cost flow problem.

In the arcs flow formulation discussed in section 1.3, the basic decision variables are flows x_{ij} on the arcs $(i, j) \in A$. The paths and cycles flow formulation start with an enumeration of all directed paths P between any pair of nodes and all directed cycles W of the network. We let $\mathcal{P}$ denote the collection of all paths and $\mathcal{W}$ the collection of all cycles. The decision variables in the paths and cycles flow formulation are $f(P)$, the flow on path P, and $f(W)$, the flow on cycle W; we define these variables for every directed path $P \in \mathcal{P}$ and every directed cycle $W \in \mathcal{W}$.

Notice that every set of paths and cycles flow uniquely determines arcs flow in a natural way. The flow x_{ij} on arc (i, j) equals the sum of the flows $f(P)$ and $f(W)$ for all paths P and cycles W that contain this arc. We formalize this observation by defining:

$$\delta_{ij}(P) := \begin{cases} 1, & \text{if } (i, j) \in P \\ 0, & \text{otherwise} \end{cases}$$

and

$$\delta_{ij}(W) := \begin{cases} 1, & \text{if } (i, j) \in W \\ 0, & \text{otherwise} \end{cases}$$

Now we can write:

$$x_{ij} = \sum_{P \in \mathcal{P}} \delta_{ij}(P) f(P) + \sum_{W \in \mathcal{W}} \delta_{ij}(W) f(W)$$

Thus each paths and cycles flow determine arcs flow uniquely. The more complex question is can we decompose any arcs flow into, i.e., represent it as, paths and cycles flow? Our most important result so far is the following theorem which gives an affirmative answer to this question.

Theorem 1.4.1 (**Flow decomposition theorem**). *Every paths and cycles flow has a unique representation as nonnegative arcs flow. Conversely, every nonnegative arcs flow x can be represented as a paths and cycles flow (though not necessarily uniquely!) with the following two properties:*

a) *Every directed path with positive flow connects a deficit node to an excess node.*

b) *At most n + m paths and cycles have nonzero flow; out of these, at most m cycles have nonzero flow.*

Proof. In the light of our previous observations, we need to establish only the converse assertions.

As stated earlier, we give an algorithmic proof to show how to decompose any arcs flow x into a paths and cycles flow. Suppose that i_0 is a deficit node. Then some arc (i_0, i_1) carries a positive flow. If i_1 is an excess node, we stop; otherwise, the mass balance constraint (1.8) of node i_1 implies that some other arc (i_1, i_2) carries positive flow. We repeat this argument until we encounter an excess node or we revisit a previously examined node. Note that one of these two cases will occur within n steps. In the former case we obtain a directed path P from the deficit node i_0 to some excess node i_k, and in the latter case we obtain a directed cycle W. In either case the path or the cycle consists solely of arcs with positive flow. If we obtain a directed path, we let $f(P) = \min\{-e(i_0), e(i_k), \min\{x_{ij} : (i, j) \in P\}\}$ and redefine $e(i_0) = e(i_0) + f(P)$, $e(i_k) = e(i_k) - f(P)$, and $x_{ij} = x_{ij} - f(P)$, $\forall(i, j) \in P$. If we obtain a directed cycle W, we let $f(W) = \min\{x_{ij} : (i, j) \in W\}$ and redefine $x_{ij} = x_{ij} - f(W)$, $\forall(i, j) \in W$. We repeat this process with the redefined problem until all node imbalances are zero. Then we select any node with at least one outgoing arc with a positive flow as the starting

node and repeat the procedure, which in this case must find a directed cycle. We terminate when $x = 0$ for the redefined problem. Clearly, the original flow is the sum of flows on the paths and cycles identified by this method. Now observe that each time we identify a directed path, we reduce the excess/deficit of some node to zero or the flow on some arc to zero; and each time we identify a directed cycle, we reduce the flow on some arc to zero. Consequently, the path and cycle representation of the given flow x contains at most $n + m$ directed paths and cycles, and at most m of these are directed cycles. □

Let us consider a flow x for which $e(i) = 0$, $\forall i \in N$. We call such a flow a *circulation*. When we apply the flow decomposition algorithm to a circulation, each iteration discovers directed cycles consisting solely of arcs with a positive flow and subsequently reduces the flow on at least one arc to zero. Consequently, a circulation decomposes into flows along at most m directed cycles.

Corollary 1.4.2. *A circulation x can be represented as cycle flow along at most m directed cycles.*

Remark 1.4.3. *We can ask ourselves what is the time complexity of the flow decomposition algorithm described in the proof of theorem 1.4.1? The answer is that it depends on data structure we are using for storing our network. The most common method is by using doubly linked list and in that case time complexity is $O(nm)$. Details can be found in [2].*

The flow decomposition theorem 1.4.1 has a number of important consequences. As one example, it enables us to compare any two solutions of a network flows problem in a particularly convenient way and to show how we can build one solution from another by a sequence of simple operations. The augmenting cycle theorem, to be discussed next, highlights these ideas.
We begin by introducing the concept of augmenting cycles with respect to a flow x.

Definition 1.4.4 (**Augmenting cycle**). *A cycle W (not necessarily directed) in G is called an augmenting cycle with respect to the flow x if by augmenting a positive amount of flow $f(W)$ around the cycle, the flow remains feasible.*

The augmentation increases the flow on forward arcs in the cycle W and decreases the flow on backward arcs in the cycle. Therefore, a cycle W is an augmenting cycle in G if $x_{ij} \le u_{ij}$ for every forward arc (i, j) and $x_{ij} \ge 0$ for every backward arc (i, j). We next extend the notation of $\delta_{ij}(W)$ for cycles which are not necessarily directed. We define:

$$\delta_{ij}(W) := \begin{cases} 1, & \text{if arc } (i, j) \text{ is a forward arc in the cycle } W \\ -1, & \text{if arc } (i, j) \text{ is a backward arc in the cycle } W \\ 0, & \text{otherwise} \end{cases}$$

Notice that in terms of residual networks 1.3, each augmenting cycle W with respect to a flow x corresponds to a directed cycle $W \in G(x)$, and vice versa. We define the cost of augmenting cycle W as $c(W) = \sum_{(i,j) \in W} c_{ij} \delta_{ij}(W)$. The cost of an augmenting cycle represents the change in the cost of a feasible solution if we augment 1 unit of flow along the cycle. The change in flow cost for augmenting $f(W)$ units along the cycle W is $c(W)f(W)$. After this preliminary discussion we are ready to introduce the following result:

Theorem 1.4.5 (**Augmenting cycle theorem**). *Let x and x^0 be any two feasible solutions of a network flow problem. Then x equals x^0 plus the flow on at most m directed cycles in $G(x^0)$. Furthermore, the cost of x equals the cost of x^0 plus the cost of flow on these augmenting cycles.*

Proof. We will use the flow decomposition theorem 1.4.1 to prove the above result in terms of residual networks. Suppose that x and x^0 are any two feasible solutions of the minimum cost flow problem. We have seen earlier (lemma 1.3.7) that some feasible circulation $x^1 \in G(x^0)$ satisfies the property that $cx = cx^0 + cx^1$. Corollary 1.4.2 implies that we can represent the circulation x^1 as cycle flows $f(W_1), f(W_2), \ldots, f(W_r)$, with $r \leq m$. Notice that each of the cycles $W_1, W_2, \ldots, W_r$ is an augmenting cycle in $G(x^0)$. Furthermore, we see that

$$
\begin{aligned}
\sum_{(i,j) \in A} c_{ij} x_{ij} &= \sum_{(i,j) \in A} c_{ij} x_{ij}^0 + \sum_{(i,j) \in G(x^0)} c_{ij} x_{ij}^1 \\
&= \sum_{(i,j) \in A} c_{ij} x_{ij}^0 + \sum_{(i,j) \in G(x^0)} c_{ij} \left[\sum_{k=1}^{r} \delta_{ij}(W_k) f(W_k) \right] \\
&= \sum_{(i,j) \in A} c_{ij} x_{ij}^0 + \sum_{k=1}^{r} c(W_k) f(W_k)
\end{aligned}
$$

Thus we have our result. $\square$

Augmenting cycle theorem permits us to obtain the following characterization of the optimal solutions of the minimum cost flow problem:

Theorem 1.4.6 (**Negative cycle optimality conditions**). *A feasible solution x^* of the minimum cost flow problem is an optimal solution if and only if it satisfies the negative cycle optimality conditions; namely, the residual network $G(x^*)$ contains no negative cost (directed) cycle.*

Proof. Suppose that x is a feasible flow and that $G(x)$ contains a negative cycle. Then x cannot be an optimal flow since by augmenting positive flow along the cycle we can improve the objective function value. Therefore, if x^* is an optimal flow, then $G(x^*)$ cannot

contain a negative cycle. Now suppose that x^* is a feasible flow and that $G(x^*)$ contains no negative cycle. Let x^0 be an optimal flow and $x^* \neq x^0$. The augmenting cycle property stated in theorem 1.4.5 shows that we can decompose the difference vector $x^0 - x^*$ into the most m augmenting cycles with respect to the flow x^* and the sum of the costs of flows on these cycles equals $cx^0 - cx^*$. Since the lengths of all the cycles in $G(x^*)$ are nonnegative, $cx^0 - cx^* \geq 0$, or $cx^0 \geq cx^*$. Moreover, since x^0 is an optimal flow, $cx^0 \leq cx^*$. Thus $cx^0 = cx^*$, and x^* is also an optimal flow. This argument shows that if $G(x^*)$ contains no negative cycle, then x^* must be optimal, and this conclusion completes the proof of the theorem. $\square$

Multi-commodity flow problem

The minimum cost flow problem models the flow of a single commodity over a network. Multi-commodity flow problem arise when several commodities use the same underlying network. The commodities may either be differentiated by their physical characteristics or simply by their origin-destination pairs. Different commodities have different origins and destinations, and commodities have separate mass balance constraints at each node. However, **the sharing of the common arc capacities binds the different commodities together**. In fact, the essential issue addressed by the multi-commodity flow problem is the allocation of the capacity of each arc to the individual commodities in a way that minimizes overall flow costs. We will dedicate whole next chapter to the multi-commodity flow problem because it will play a central role in our modelling of a railway system.

1.5 Linear programming and simplex method

In this section, we will set a notation with respect to linear programming which we are going to use later on. Because *revised simplex method* will play a central role in the algorithm which is going to be described in section 2.4, we will try to present the main ideas behind it as concise as possible.

A linear program is an optimization problem with a linear objective function, a set of linear constraints, and a set of nonnegativity restrictions imposed upon the underlying decision variables; that is, it is an optimization model of the form

$$\min \sum_{j=1}^{q} c_j x_j \tag{1.10}$$

subject to

$$\sum_{j=1}^{q} a_{ij}x_j = b(i), \quad i = 1, 2, \dots, p, \tag{1.11}$$

$$x_j \geq 0, \quad j = 1, 2, \dots, q. \tag{1.12}$$

Remark 1.5.1. *In many texts, m denotes the number of equality constraints and n denotes the number of decision variables. This notation, unfortunately, is the reverse of the convention in network flows, since network flow system contains one constraint per node and one variable per arc. For this reason, we do not use the notation of m and n to denote the number of constraints and variables of a linear program.*

We assume, by multiplying the ith constraint by -1, if necessary, that the right hand side coefficient $b(i)$ of each constraint $i = 1, 2, \dots, p$ is nonnegative. We might note that we could formulate a linear program in several alternative ways. Since linear programming literature frequently refers to the formulation (1.10) as the *standard form* of a linear program, we will stick to it.

We can write linear programming model (1.10) in more compact, matrix form, as follows:

$$\min cx \tag{1.13}$$

subject to

$$\mathcal{A}x = b, \tag{1.14}$$

$$x \geq 0 \tag{1.15}$$

In this formulation, the matrix $\mathcal{A} = (a_{ij})$ has p rows and q columns, the vector $c = (c_j)$ is a q-dimensional row vector, vectors $x = (x_j)$ and $b = (b(i))$ are q-dimensional and p-dimensional column vectors, respectively. We let $\mathcal{A}_j$ denote the column of $\mathcal{A}$ corresponding to the variable x_j. We assume that the rows of the matrix $\mathcal{A}$ are linearly independent. For the special case of the minimum cost flow problem (see section 1.3), each component of the decision variable x corresponds to the flow on an arc and the matrix $\mathcal{A}$ has one row for each node of the underlying network. In this case the matrix $\mathcal{A}$ is the node-arc incident matrix $\mathcal{N}$.

During its execution, the simplex method modifies the original program stated in the standard form (1.13) by performing a series of one or more of the following elementary row operations:

1. Multiplying a row (i.e., constraint) by a constant, or

2. Adding one row to another row or to the objective function.

Since we have stated all constraints in the equality form, row operations do not affect a set of feasible solutions of a linear program.

Remark 1.5.2. *To model an inequality constraint $\sum_{j=1}^{q} a_{ij}x_j \leq b(i)$ as an equality constraint, we could add a new nonnegative "slack variable" y_i, with zero cost, and write the inequality as $\sum_{j=1}^{q} a_{ij}x_j + y_i = b(i)$.*

Using the above stated operations, we want to transform a linear program so that its new, equivalent, formulation satisfies the following property:

Definition 1.5.3 (**Canonical property**). *Formulation has one decision variable isolated in each constraint; a variable isolated in a given constraint has a coefficient of +1 in that constraint and does not appear in any other constraint, nor does it appear in the objective function.*

Remark 1.5.4. *A linear program typically has a large number of canonical forms since there are many ways to isolate decision variables in the constraints.*

Given a canonical form for any linear program, we obtain a *basic feasible solution* by setting a variable isolated in constraint i, called i^{th} *basic variable*, equal to the right hand side of the ith constraint, and by setting all the remaining variables, called *nonbasic variables*, to value zero. Collectively, basic variables are known as the *basis*. In general, we obtain basic feasible solution as follows. We isolate a variable in each constraint. For simplicity, assume that we have isolated the variable x_i in the ith constraint. Let $\mathbf{B} = \{1, 2, \ldots, p\}$ denote the index set of basic variables and let $\mathbf{L} = \{p+1, p+2, \ldots, q\}$ denote the index set of nonbasic variables. We refer to the pair $(\mathbf{B}, \mathbf{L})$ as a *basis structure* of a linear problem. For a given basis structure $(\mathbf{B}, \mathbf{L})$, we can compatibly partition the columns of the constraint matrix $\mathcal{A}$. Let $\mathcal{B} = \left[\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p \right]$ and $\mathcal{L} = \left[\mathcal{A}_{p+1}, \mathcal{A}_{p+2}, \ldots, \mathcal{A}_q \right]$. We refer to $p \times p$ matrix $\mathcal{B}$ as a *basis matrix*. We also let $x_{\mathbf{B}} = [x_i : i \in \mathbf{B}]$ and $x_{\mathbf{L}} = \left[x_j : j \in \mathbf{L} \right]$ be a partitioning of variables into subvectors corresponding to the index sets $\mathbf{B}$ and $\mathbf{L}$. With this notation, we can rewrite the constraint matrix $\mathcal{A}x = b$ as

$$\mathcal{B}x_{\mathbf{B}} + \mathcal{L}x_{\mathbf{L}} = b. \tag{1.16}$$

We convert (1.16) to the canonical form by premultiplying each term by $\mathcal{B}^{-1}$, the inverse of a basis matrix, giving

$$x_{\mathbf{B}} + \mathcal{B}^{-1}\mathcal{L}x_{\mathbf{L}} = \mathcal{B}^{-1}b. \tag{1.17}$$

We obtain a basic solution from (1.17) by setting each nonbasic variable to value zero. The resulting solution is

$$x_{\mathbf{B}} = \mathcal{B}^{-1}b \quad \text{and} \quad x_{\mathbf{L}} = 0. \tag{1.18}$$

We refer to this solution as a *basic feasible solution* if the value of each basic variable is nonnegative (i.e., $x_{\mathbf{B}} \geq 0$). We also say that the basis structure $(\mathbf{B}, \mathbf{L})$ is *feasible* if its associated basic solution is feasible. For some choices of the basis matrix, the corresponding basic solution will be feasible, and for some other choices it will not.

Converting a linear program (1.13) to the canonical form (1.5.3) requires that we invert basis matrix, which is possible only if the columns associated with basic variables are linearly independent. If the associated columns are linearly dependent, basis matrix is singular and we cannot invert it. We shall therefore henceforth refer to a basis as a subset of p variables whose corresponding columns are linearly independent.

During its execution, the simplex method requires information about $\mathcal{B}^{-1}\mathcal{A}_j$, the updated column corresponding to nonbasic variable x_j. We let $\bar{A}_j = \mathcal{B}^{-1}\mathcal{A}_j$ and call this vector the *representation* of $\mathcal{A}_j$ with respect to the basis matrix $\mathcal{B}^{-1}$ (or, alternatively, basis $\mathbf{B}$). For notational convenience, $\bar{A}_{\mathbf{L}}$ denote the matrix $\mathcal{B}^{-1}\mathcal{L}$ containing the column representation of all nonbasic variables and let $\bar{b} = \mathcal{B}^{-1}b$; we refer to the vector $\bar{b}$ as the modified right hand side. Finally, let $c_{\mathbf{B}} = (c_1, c_2, \dots, c_p)$ and $c_{\mathbf{L}} = (c_{p+1}, c_{p+2}, \dots, c_q)$ denote the cost vectors associated with basic and nonbasic variables, respectively.

In a canonical form of a linear program, each basic variable has a zero coefficient in the objective function. We can obtain this special form of the objective function by performing a sequence of elementary row operations (i.e., multiplying constraints by some multipliers and subtracting them from the objective function). Any sequence of elementary row operations is equivalent to the following: multiply each constraint i by a number $\pi(i)$ and subtract it from the objective function. This operation gives the equivalent objective function $\sum_{j=1}^{q} c_j x_j - \sum_{i=1}^{p} \pi(i) \left[\sum_{j=1}^{q} a_{ij} x_j - b(i) \right]$, or, collecting terms and letting $z_0 = \sum_{i=1}^{p} \pi(i)b(i)$,

$$z(x) = \sum_{j=1}^{p} \left[c_j - \sum_{i=1}^{p} \pi(i)a_{ij} \right] x_{ij} + \sum_{j=p+1}^{q} \left[c_j - \sum_{i=1}^{p} \pi(i)a_{ij} \right] x_{ij} + z_0. \tag{1.19}$$

To obtain a canonical form, we select the vector π so that

$$c_j - \sum_{i=1}^{p} \pi(i)a_{ij} = 0 \quad , \forall j \in \mathbf{B}. \tag{1.20}$$

In the matrix notation we select π so that

$$\pi\mathcal{B} = c_{\mathbf{B}}$$

In this expression, $c_{\mathbf{B}} = (c_1, c_2, \dots, c_p)$ is the cost vector associated with basic variables. We refer to

$$\pi = c_{\mathbf{B}}\mathcal{B}^{-1}$$

as the *simplex multipliers* associated with the basis $\mathbf{B}$ and refer to $c_j^\pi = c_j - \sum_{i=1}^p \pi(i)a_{ij}$ as the *reduced cost* (see subsection 1.3) of the variable x_j. Note that $z_0 = \pi b = c_{\mathbf{B}}\mathcal{B}^{-1}b = c_{\mathbf{B}}x_{\mathbf{B}}$, which (since $x_{\mathbf{L}} = 0$) is the value of the objective function corresponding to the basis $\mathbf{B}$. The simplex method maintains a basic feasible solution at every step. Given a basic feasible solution, the method first applies the optimality criteria to test the optimality of the current solution. If the current solution does not fulfill this condition, the algorithm performs an operation, known as *pivot operation*, to obtain another basis structure with a lower or identical cost. The simplex method repeats this process until the current basic feasible solution satisfies the optimality criteria.

Optimality criteria

Let $(\mathbf{B}, \mathbf{L})$ denote a feasible basis structure of a linear program. Assume, for simplicity, that $\mathbf{B} = \{1, 2, \ldots, p\}$. Consider the canonical form associated with this basic structure. In this canonical form the objective function is

$$\min z(x) = z_0 + \sum_{j=p+1}^q c_j^\pi x_j. \tag{1.21}$$

The coefficient $c_j^\pi = c_j - \sum_{i=1}^p \pi(i)a_{ij}$ is the reduced cost of the nonbasic variable x_j with respect to the current simplex multipliers π. We claim that if $c_j^\pi \geq 0$ for all nonbasic variables x_j, the current basic feasible solution x is an optimal solution of the linear program. To see this, observe that in any feasible solution of the linear program, $x_j \geq 0$, $j \in \mathbf{L}$. Therefore, if $c_j^\pi \geq 0$ for all nonbasic variables x_j, then z_0 is a lower bound on the optimal objective function value. Therefore, because the current solution x, which sets $x_j = 0$, $\forall j \in \mathbf{L}$, achieves this lower bound, it must be optimal.

Pivot operation

If $c_j^\pi < 0$ for some nonbasic variable x_j, the current basic feasible solution might not be optimal. The expression (1.21) implies that c_j^π is the rate of decrease in the objective function value per unit increase in the value of x_j. The simplex method selects one such nonbasic variable, say x_s, as the *entering variable* and tries to increase its value. As we will see, when the simplex method increases x_s, as much as possible while keeping all the other nonbasic variables at value zero, some basic variable, say x_r, reaches value zero. The simplex method replaces the basic variable x_r by x_s, defining a new basis structure. It then updates the inverse of the basis and repeats the computations.

If we increase the value of the entering variable x_s, to value θ and keep all the other nonbasic variables at zero value, expression (1.17) implies that the basic variables $x_{\mathbf{B}}$ change in the following manner:

$$x_{\mathbf{B}} + \theta \bar{\mathcal{A}}_s = \bar{b} \tag{1.22}$$

In this expression $\bar{b} = \mathcal{B}^{-1}b \geq 0$, $\bar{\mathcal{A}}_s = \mathcal{B}^{-1}\mathcal{A}_s$, and $x_{\mathbf{B}} = \begin{bmatrix} x_1, x_2, \ldots, x_p \end{bmatrix}$. Let $\bar{\mathcal{A}}_s = \begin{bmatrix} \bar{a}_{1s}, \bar{a}_{2s}, \ldots, \bar{a}_{ps} \end{bmatrix}$. We can restate (1.22) as

$$x_i = \bar{b}(i) - \theta \bar{a}_{is} \quad \forall i = 1, 2, \ldots, p. \tag{1.23}$$

If $\bar{a}_{is} \leq 0$ and we increase θ, then x_i either remains unchanged or increases. If $\bar{a}_{is} > 0$, then the scalar θ must satisfy the condition $\theta \leq \bar{b}(i)\bar{a}_{is}$ in order for x_i to remain nonnegative. As a result,

$$\theta = \min_{1 \leq i \leq p} \{ \bar{b}(i)/\bar{a}_{is} : \bar{a}_{is} > 0 \}$$

is the largest value of θ that we can assign to x_s while remaining feasible. What if $\bar{a}_{is} \leq 0$, $\forall i = 1, 2, \ldots, p$? Then we can assign arbitrarily large values to the entering variable x_s and the solution remains feasible. Since $c_s^\pi < 0$, by setting x_s as large as we like, we can make the objective function arbitrarily small, and make it approach $-\infty$. In this instance, we say that the linear program has an *unbounded solution*.

We next focus on situations in which θ is finite. If we set $x_s = \theta$, one of basic variables, say x_r, becomes zero. Note that $\bar{b}(r)/\bar{a}_{rs} = \theta$. We refer to x_r as the *leaving variable* and refer to the rule we have described for identifying θ and the corresponding leaving variable as the *minimum ratio rule*. Next, we designate x_s a basic variable, x_r a nonbasic variable, and update the canonical form of the linear program so that it satisfies the canonical property with respect to a new basis. We do so by performing a sequence of elementary row operations.

Updating the simplex tableau - pivot operation

Recall that in the canonical form with respect to the basis $\mathbf{B}$, the equations of the linear program assume the form $x_{\mathbf{B}} + \mathcal{B}^{-1}\mathcal{L}x_{\mathbf{L}} = \mathcal{B}^{-1}b$. In a new basis, the entering variable x_s becomes the basic variable for the rth row, which requires that in a new canonical form the variable x_s should have a coefficient $+1$ in the rth row, and a coefficient 0 in all other rows. We achieve this new canonical form by first dividing the rth row by $\bar{a}_{rs}$; the variable x_r then has a $+1$ coefficient in this row. Then, for each $1 \leq i \leq p$, $i \neq r$, we multiply the rth row by the constant $-\bar{a}_{is}$ and add it to the ith row so that the updated values of $\bar{a}_{is}$ becomes zero. We also multiply the rth row by a constant $-c_s^\pi$ and add it to the objective function so that the objective function coefficient of x_s becomes zero. We refer to this set of computations as a *pivot operation*.

Revised simplex method

One way to perform the pivot operation is by updating the full matrix $\bar{\mathcal{A}}$, i.e., by performing the explicit set of pivot computations iteratively on the matrix $\mathcal{A}$. This set of computations can be very expensive for a linear program that contains many variables (as will be the case with our problem). The *revised simplex method* is a particular implementation of the simplex method that permits us to avoid many of these computations. To describe a basic approach of the revised simplex method, suppose that we (conceptually) append a set of fictitious variables y to the original linear program and form a new linear program with the constraints:

$$\mathcal{A}x + \mathcal{I}y = b,$$

$$x \geq 0, \quad y \geq 0.$$

In this formulation, $\mathcal{I}$ is an identity matrix. Then to obtain the canonical form (1.22) with respect to the basis $\mathcal{B}$, we premultiply this system by the basis inverse $\mathcal{B}^{-1}$. With the fictitious variables y, the system becomes

$$x_{\mathbf{B}} + \mathcal{B}^{-1}\mathcal{L}x_{\mathbf{L}} + \mathcal{B}^{-1}y = \mathcal{B}^{-1}b.$$

As shown by this expression, if we were to perform a pivot operation on the entire matrix $\mathcal{A}$ from step to step, the coefficients of fictitious variables y would be the basis inverse. This observation shows that we need not carry out pivot operations on the entire matrix $\mathcal{A}$. Instead, we can perform these operations on the columns associated with the initial identity matrix $\mathcal{I}$ (we do not formally introduce the variables y). Since the resulting computations give us the basis inverse $\mathcal{B}^{-1}$, we can use this matrix to compute the simplex multipliers $\pi = c_{\mathbf{B}}\mathcal{B}^{-1}$ and then use them to compute the reduced cost of each variable. Once we have determined the variable x_s to introduce into the basis, we compute its representation $\bar{\mathcal{A}}_s = \mathcal{B}^{-1}\mathcal{A}_s$ and then use this information to perform the ratio test to identify the variable x_r to leave the basis. We next perform row elementary operations on the current basis $\mathcal{B}^{-1}$ matrix and obtain updated basis inverse.

The advantage of this approach is that we use only the original data $\mathcal{A}$ in computing the reduced costs (using the formula $c_j^\pi = c_j - \sum_{i=1}^{p} \pi(i)a_{ij}$) and determine the modified data $\bar{\mathcal{A}}$ for only one column s of $\mathcal{A}$ and perform row elementary operations only on a basis inverse matrix. This apparently modest change in the algorithm often has dramatic effects on its efficiency because the original data $\mathcal{A}$ for most problems met in practice is very sparse in the sense that most (90% or more) of its coefficients are zero. On the other hand, the modified matrix $\bar{\mathcal{A}}$ typically becomes very dense as we perform the iterations of the simplex algorithm. In the revised simplex method, by using appropriate data structures, we can avoid all the computations corresponding to zero elements. As a consequence, by implementing the revised simplex method, we usually achieve great savings in our computations.

With this result, we will finish our discussion about simplex method but we advise inter-ested reader to check results about *termination of simplex method* and *bounded variable simplex method.* Basic introduction, with respect to these methods, can be found in [2].

At this point, all of our preliminary work is done so we can move forward to describe multi-commodity flow problem and introduce column generation approach for solving it.

Chapter 2

Multi-commodity flows

2.1   Introduction

If the commodities do not interact in any way, then to solve problems with several commodities, we would solve each single-commodity problem separately. However, in other situations because the commodities do share common facilities, the individual single commodity problems are not independent, so to find an optimal flow, we need to solve problems in concern with each other. In this section, we study one such model, known as *multi-commodity flow problem*, in which individual commodities share common arc capacities. That is, each arc has a capacity u_{ij} that restricts a total flow of all commodities on that arc.

Let x_{ij}^k denote a flow of the commodity k on the arc (i, j), and let x^k and c^k denote flow vector and per unit cost vector for commodity k. Using this notation we can formulate *multi-commodity flow* problem as follows:

$$\min \sum_{1 \le k \le K} c^k x^k \tag{2.1}$$

subject to

$$\sum_{1 \le k \le K} x_{ij}^k \le u_{ij}, \quad \forall (i, j) \in A, \tag{2.2}$$

$$\mathcal{N} x^k = b^k, \quad k = 1, 2, \dots, K, \tag{2.3}$$

$$0 \le x_{ij}^k \le u_{ij}^k \quad \forall (i, j) \in A, \text{ and } k = 1, 2, \dots, K. \tag{2.4}$$

This formulation has a collection of K ordinary *mass balance constraints* (2.3), modelling flow of each commodity $k = 1, 2, \dots, K$. The *bundle* constraint (2.2) tie together commodities by restricting a total flow $\sum_{1 \le k \le K} x_{ij}^k$ of all commodities on each arc (i, j) to at most u_{ij}. Note that we also impose individual flow bounds u_{ij}^k on the flow of commodity k on the

24

arc (i, j). Many applications do not impose these bounds, so for these applications we set each bound to $+\infty$. Although we might formulate a variety of alternative multi-commodity models with different assumptions, we will refer to this model as *multi-commodity flow problem*.

Sometimes it is more convenient to state the bundle constraints (2.2) as equalities instead of inequalities. In these instances we introduce nonnegative *slack* variables s_{ij} and write the bundle constraints as

$$\sum_{1 \leq k \leq K} x_{ij}^k + s_{ij}^k = u_{ij}, \quad \forall (i, j) \in A, \tag{2.5}$$

The slack variable s_{ij} for the arc (i, j) measures unused bundle capacity on that arc.

Note that model (2.1) imposes capacities on arcs but not on nodes. This modelling assumption imposes no loss of generality, since by using node splitting technique (see subsection 1.3), we can use this formulation to model situations with node capacities as well. Three other features of this model are worth nothing.

Remark 2.1.1 (**Homogeneous goods assumption**). *We are assuming that every unit flow of each commodity uses* 1 *unit of capacity of each arc. A more general model would permit the unit flow of each commodity k to consume a given amount ρ_{ij}^k of the capacity (or some other resource) associated with each arc (i, j), and replace the bundle constraints with a more general resource availability constraint $\sum_{1 \leq k \leq K} \rho_{ij}^k x_{ij}^k \leq u_{ij}$. With minor modifications, the solution techniques that we will be discussing here can be applied to this more general model as well.*

Remark 2.1.2 (**No congestion assumption**). *We are assuming that we have a hard (i.e., fixed) capacity on each arc and that the cost on each arc is linear in the flow on that arc. In some applications encountered in communication, transportation, and other problem domains, commodities interact in a more complicated fashion in a sense that as the flow of any commodity increases on an arc, we incur an increasing and nonlinear cost on that arc. This type of model arises frequently, e.g., in traffic networks where the objective function is to find a flow pattern of all commodities that minimizes overall system delay. In this setting, because of queuing effects, the greater the flow on an arc, the greater is the queuing delay on that arc. For example, a "congestion" model for multi-commodity flow might contain the individual flow constraints and , no bundle constraints, but a nonlinear objective function of the form*

$$\min \sum_{(i,j) \in A} \frac{x_{ij}}{u_{ij} - x_{ij}}.$$

In this model u_{ij} is the nominal capacity of the arc (i, j); as the total flow $x_{ij} = \sum_{(i,j) \in A} x_{ij}^k$

on any arc approaches the arc's nominal capacity, the delay approaches $+\infty$. Practitioners often use this type of model in the context of "performance modelling" to see how overall system delay, or performance, varies as a function of various system designs (e.g., in response to change in the network topology).

Remark 2.1.3 (**Indivisible goods assumption**). *Above model assumes that the flow variables can be fractional. In some applications encountered in practice, this assumption is appropriate; in other contexts, however, the variables must be integer valued. In these instances the model that we are considering might still prove to be useful, since linear programming model might either be a good approximation of integer programming model (we will deal with it later), or we can use linear programming model as linear programming relaxation of an integer program and embed it within branch-and-bound or some other type of enumeration approach. We note that the integrality of solutions is one very important distinguishing feature between single and multi-commodity flow problems. One very nice feature of single-commodity network flow problems is that they always have integer solutions, whenever the supply/demand and capacity data are integer valued. Multi-commodity flow problems do not satisfy this integrality property.*

2.2 Solution approaches

There are several approaches for solving *multi-commodity flow* problem, including:

1. Price-directive decomposition

2. Resource-directive decomposition

3. Partitioning methods

Price-directive decomposition methods place Lagrangian multipliers (or prices) on the bundle constraints and bring them into the objective function so that the resulting problem decomposes into the separate minimum cost flow problem for each commodity *k*. That is, these methods remove the capacity constraint and instead "charge" each commodity for the use of the capacity of each arc. These methods attempt to find appropriate prices so that some optimal solution to the resulting "pricing problem" or Lagrangian subproblem also solves the overall multi-commodity flow problem. Several methods are available for finding appropriate prices. Details about finding correct prices applying Lagrangian relaxation reader can find in [2].

Dantzig-Wolfe decomposition is another approach for finding a correct prices; that method is general-purpose approach for decomposing problems that have a set of *easy* constraints and also a set of *hard* constraints (i.e., constraints that make problem much more difficult to solve). For multi-commodity flow problems, the network flow constraints are the easy constraints and the bundle constraints are the hard constraints. The approach begins, like Lagrangian relaxation, by ignoring or imposing prices on the bundle constraints and solving Lagrangian subproblems with only the single-commodity network flow constraints. The resulting solutions need not satisfy the bundle constraints, and the method uses linear programming to update the prices so that the solutions generated from the subproblems satisfy the bundle constraints. The method iteratively solves two different problems: a Lagrangian subproblem and a price-setting linear program. This method has played an important role in the field of optimization both because the algorithm itself has proven to be very useful, and also because it has stimulated many other approaches to problem decomposition. Moreover, the algorithm and its associated underlying theory have had a significant influence on the filed of economics since this type of price decomposition formalizes ideas of transfer pricing and coordination that lie at the heart of planned economics. Later on we will introduce and describe the related *column generation technique* for solving *multi-commodity flow problem*.

An alternative way of viewing multi-commodity flow problem is as a capacity allocation problem. All commodities are competing for the fixed capacity u_{ij} of every arc (i, j) of the network. Any optimal solution to multi-commodity flow will prescribe a specific flow on

each arc (i, j) for each commodity which is the appropriate capacity to allocate to that commodity. If we started by allocating these capacities to the commodities and then solved the resulting (independent) single-commodity flow problems, we would be able to solve the problem quite easily as a set of independent single-commodity flow problems. Resource-directive methods provide a general solution approach for implementing this idea. They begin by allocating capacities to commodities and then use information gleaned from the solution to the resulting single-commodity problems to reallocate the capacities in a way that improve the overall system cost.

Partitioning methods exploit the fact that multi-commodity flow problem is a specially structured linear program with embedded network flow problems. To solve any single-commodity flow problem, we can use the network simplex method. Using similar approach we can solve multi-commodity flow problem but trying to describe it any further would lead us outside the scope of this work.

2.3 Lagrangian relaxation technique

Lagrangian relaxation procedure is together with linear programming relaxation technique, one of the most widely used relaxation techniques. Linear programming relaxation replaces constraints with the requirement that variables are integer by an appropriate continuous constraints. It usually serves as a good approximation to the integer programming problem but it can be shown that Lagrangian relaxation technique offers even better approximation of the same problem. Furthermore, Lagrangian relaxation procedure can be used in a wider range of the optimization problems. Nevertheless, Lagrangian relaxation procedure is more complicated and in this section we will only introduce it because of notational purposes, and state the results which we will need in subsection 2.4 to determine a lower bounds of a column generation approach. These results are non trivial and to prove them one need a lot of side results. A full length discussion about Lagrangian relaxation technique can be found in [2].

To describe the general form of Lagrangian relaxation procedure, suppose that we consider the following generic optimization model formulated in terms of the vector x of decision variables:

$$z^* = \min cx$$

subject to

$$\mathcal{A}x = b, \tag{P}$$

$$x \in X.$$

Model (P) has a linear objective function cx and a set $\mathcal{A}x = b$ of explicit linear constraints. The decision variables x are also constrained to lie in a given constraint set X which often models embedded network flow structure. For example, the constraint set $X = \{x : \mathcal{N}x = q, 0 \le x \le u\}$ might be all feasible solutions to network flow problem with a supply/demand vector q. Furthermore, we assume that the set X is finite.

Lagrangian relaxation procedure uses the idea of relaxing the explicit linear constraints by bringing them into the objective function with associated Lagrange multipliers μ (this idea might be familiar one from calculus in the context of solving nonlinear optimization problems): We refer to the resulting problem

$$\min cx + \mu(\mathcal{A}x - b)$$

subject to

$$x \in X,$$

as *Lagrangian relaxation* or *Lagrangian subproblem* of the original problem, and refer to the function

$$L(\mu) = \min\{cx + \mu(\mathcal{A}x - b) : x \in X\},$$

as *Lagrangian function*. Note that since in forming Lagrangian relaxation, we have eliminated the constraints $\mathcal{A}x = b$ from the problem formulation, the solution of Lagrangian subproblem need not be feasible for the original problem (P). Nevertheless, we can obtain useful information about the original problem even when the solution to Lagrangian subproblem is not feasible in the original problem (P). The following elementary observation is the key result that motivates use of Lagrangian relaxation technique in general.

Lemma 2.3.1 (**Lagrangian bounding principle**). *For any vector μ of Lagrangian multipliers, the value $L(\mu)$ of Lagrangian function is a lower bound on the optimal objective function z^* of the original optimization problem (P).*

To obtain the sharpest possible lower bound, we would need to solve the following optimization problem

$$L^* = \max_{\mu} L(\mu)$$

which we refer to as *Lagrangian multiplier problem* associated with the original optimization problem (P). Lagrangian bounding principle has the following immediate implication:

Proposition 2.3.2 (**Weak duality**). *The optimal objective function L^* of Lagrangian multiplier problem is always a lower bound on the optimal objective function value of the problem (P) (i.e., $L^* \le z^*$).*

Our preceding discussion provides us with valid bounds for comparing objective function values of Lagrange multiplier and optimization (P) problems for any choice of Lagrange multipliers μ and any feasible solution x of (P):

$$L(\mu) \leq L^* \leq z^* \leq cx.$$

These inequalities furnish us with a guarantee when Lagrange multiplier μ to Lagrange multiplier problem or a feasible solution x to the original problem (P) are optimal.

Proposition 2.3.3 (**Optimality test**).

a) *Suppose that μ is a vector of Lagrangian multipliers and x is a feasible solution of the optimization problem (P) satisfying the condition $L(\mu) = cx$. Then $L(\mu)$ is an optimal solution of Lagrangian multiplier problem (i.e., $L^* = L(\mu)$) and x is an optimal solution to the optimization problem (P).*

b) *If for some choice of Lagrangian multiplier vector μ, the solution x^* of Lagrangian relaxation is feasible in optimization problem (P), then x^* is an optimal solution of the optimization problem (P) and μ is an optimal solution to Lagrangian multiplier problem.*

As indicated by proposition 2.3.3, the bounding principle immediately implies one advantage of the Lagrangian relaxation approach. The method can give us a *certificate* (in the form of the equality $L(\mu) = cx$ for some Lagrange multiplier μ) for guaranteeing that a given feasible solution x of the optimization problem (P) is an optimal solution. Even if $L(\mu) < cx$, having the lower bound permits us to state a bound on how far a given solution is from optimality. Let say if $[cx - L(\mu)]/L(\mu) \leq 0.05$, we know that the objective function value of the feasible solution x is no more than 5% from optimality. This type of bound is very useful in practice. It permits us to assess the degree of suboptimality of given solutions and it permits us to terminate our search for an optimal solution when we have a solution that we know is close enough to optimality (in objective function value) for our purposes. This idea will prove very useful in our algorithm later on.

Remark 2.3.4. *In the optimization model (P), constraints $\mathcal{A}x = b$ are all equality constraints. In case with inequality constraints $\mathcal{A}x \leq b$, Lagrangian multiplier problem is a slight variant of the one we have just introduce. It becomes*

$$L^* = \max_{\mu \geq 0} L(\mu)$$

*Later on it implies one substantial difference in optimality test (proposition 2.3.3). To be optimal for an optimization problem (P), the solution x^* of Lagrangian subproblem, in addition to being feasible, needs to satisfy **complementary slackness condition** $\mu(\mathcal{A}x^* - b) =$*

0. *Since proper discussion on this matter will just complicate and prolong our discussion without any major benefits, we will move forward to state the results of our interest.*

Suppose next that we apply Lagrangian relaxation to a discrete optimization problem (P) defined as $\min\{cx : \mathcal{A}x = b, x \in X\}$. We assume that discrete set X is specified as $X = \{x : \mathcal{D}x \leq q, , x \geq 0$ and integer$\}$ for an integer matrix $\mathcal{D}$ and an integer vector q. Consequently, problem (P) becomes

$$z^* = \min\{cx : \mathcal{A}x = b, \mathcal{D}x \leq q, \ x \geq 0 \text{ and integer}\}.$$

We incur essentially no loss of generality by specifying set X in this manner because we can formulate almost all real-life discrete optimization problems as integer programming problems. Let

$$z^\circ = \min\{cx : \mathcal{A}x = b, \mathcal{D}x \leq q, \ x \geq 0\}. \tag{LP}$$

be the linear programming relaxation of the problem (P).
Clearly, $z^\circ \leq z^*$ because set of feasible solutions of (LP) lies within set of feasible solutions of (P). Therefore, linear programming relaxation provides a valid lower bound on the optimal objective function value of (P). Proposition 2.3.2 says us that $L^* \leq z^*$. Next theorem says which of these two lower bounds is sharper.

Theorem 2.3.5. *When applied to an integer program stated in minimization form, the lower bound obtained by the Lagrangian relaxation technique is always as large (or sharp) as the bound obtained by linear programming relaxation of the problem; that is, $z^\circ \leq Ł^*$.*

To be consistent with a notation in the discussion to follow, after these general observations, we will now introduce notation connected with Lagrangian relaxation for *multi-commodity flow problem*.

Lagrangian relaxation for multi-commodity flow problem

To apply Lagrangian relaxation to multi-commodity flow problem, we associate nonnegative Lagrange multipliers w_{ij} with the bundle constraints (2.1.2), creating the following Lagrangian subproblem:

$$L(w) = \min \sum_{1 \leq k \leq K} c^k x^k + \sum_{(i,j) \in A} w_{ij} \left(\sum_{1 \leq k \leq K} x_{ij}^k - u_{ij} \right) \tag{2.6}$$

or, equivalently,

$$L(w) = \min \sum_{1 \leq k \leq K} \sum_{(i,j) \in A} (c_{ij}^k + w_{ij}) x_{ij}^k - \sum_{(i,j) \in A} w_{ij} u_{ij} \tag{2.7}$$

subject to

$$\mathcal{N}x^k = b^k, \quad k = 1, 2, \ldots, K, \tag{2.8}$$

$$x_{ij}^k \geq 0 \quad \forall (i, j) \in A, \text{ and } k = 1, 2, \ldots, K. \tag{2.9}$$

Note that since the term $-\sum_{(i,j)\in A} w_{ij}u_{ij}$ in the objective function of Lagrangian subproblem is a constant for any given choice of Lagrange multipliers, we can ignore it for any fixed value of these multipliers. The resulting objective function for Lagrangian subproblem has a cost of $c_{ij}^k + w_{ij}$ associated with every flow variable x_{ij}^k. Since none of the constraints in this problem contains flow variables for more than one of commodities, problem decomposes into separate minimum cost flow problems, one for each commodity.

Remark 2.3.6. *At the end it is worth mentioning **subgradient pptimization technique**. It is quite useful and reliable technique for solving Lagrange multiplier problem. Basic idea behind this method comes from gradient method in nonlinear programming. The major difference is that Lagrangian function need not be differentiable in every point.*

2.4 Column generation approach

To simplify our discussion in this section, we consider a special case of multi-commodity flow problem. We assume that each commodity k has a single source node s^k, a single sink node t^k and a flow requirement of d^k units between these source and sink nodes. We also assume that we impose no flow bounds on the individual commodities other than the bundle constraints (2.1.2). Therefore, for each commodity k, subproblem constraints $\mathcal{N}x^k = b^k$, $x^k \geq 0$ define a shortest path problem (see subsection 1.3). For this model, for any choice w_{ij} of Lagrange multipliers for the bundle constraints, Lagrangian relaxation requires solution of a series of shortest path problems, one for each commodity.

Reformulation with path flows

To begin our discussion in this subsection, let us first reformulate multi-commodity flow problem using paths and cycles flows instead of the arc flows. Recall from section 1.4 that we can formulate any network flow problem using paths and cycles flows. To simplify our discussion even further, let us assume that for every commodity cost of every cycle W in the underlying network is nonnegative. For example, the problem satisfies this condition if the arc flow costs are all nonnegative. If we impose this nonnegative cycle cost condition, then in some optimal solution of the problem, flow on every cycle is zero, so we can eliminate cycle flow variables. Therefore, throughout this section, we assume that we can represent any potentially optimal solution as the sum of flows on directed paths. Let us recall our notation from section 1.4 concerning paths and cycles decomposition, tailored a bit for

multi-commodity flow problem.

For each commodity k, let $\mathbf{P^k}$ denote a collection of all directed paths from the source node s^k to the sink node t^k in the underlying network $G = (N, A)$. In paths flow formulation, each decision variable $f(P)$ is the flow on some path P and for the kth commodity, we define this variable for every directed path $P \in \mathbf{P^k}$.

As in section 1.4, let $\delta_{ij}(P)$ be an arc-path indicator variable, i.e., $\delta_{ij}(P)$ equals 1 if arc (i, j) is contained in the path P, and is 0 otherwise. Flow decomposition theorem 1.4.1 of network flows states that we can always decompose some optimal arcs flow x_{ij}^k into path flows $f(P)$ as follows:

$$x_{ij}^k = \sum_{P \in \mathbf{P^k}} \delta_{ij}(P)f(P), \quad \forall (i, j) \in A$$

Let $c^k(P) = \sum_{(i,j) \in A} c_{ij}^k \delta_{ij}(P) = \sum_{(i,j) \in P} c_{ij}^k$ denote per unit cost of flow on the path $P \in \mathbf{P^k}$ with respect to the commodity k. Note that for each commodity k, if we substitute for arc flow variables in the objective function, interchange the order of summations, and collect terms, we find that

$$\sum_{(i,j) \in A} c_{ij}^k x_{ij}^k = \sum_{(i,j) \in A} c_{ij}^k \left[\sum_{P \in \mathbf{P^k}} \delta_{ij}(P)f(P) \right] = \sum_{P \in \mathbf{P^k}} c^k(P)f(P).$$

This observation shows that we can express cost of any solution as either cost of arc flows or the cost of path flows.

By substituting path variables in multi-commodity flow formulation, we obtain the following equivalent paths flow formulation of the problem:

$$\min \sum_{1 \leq k \leq K} \sum_{P \in \mathbf{P^k}} c^k(P)f(P) \tag{2.10}$$

$$\sum_{1 \leq k \leq K} \sum_{P \in \mathbf{P^k}} \delta_{ij}(P)f(P) \leq u_{ij}, \quad \forall (i, j) \in A, \tag{2.11}$$

$$\sum_{P \in \mathbf{P^k}} f(P) = d^k, \quad k = 1, 2, \ldots, K, \tag{2.12}$$

$$f(P) \geq 0, \quad \forall P \in \mathbf{P^k} \text{ and } k = 1, 2, \ldots, K. \tag{2.13}$$

In formulating this problem we have invoked the flow decomposition theorem 1.4.1 stating that we can decompose any feasible arcs flow of the system $\mathcal{N} x^k = b^k$ into a set of paths and cycles in such a way that the paths flow satisfy the mass balance condition (2.12).

Note that paths flow formulation of multi-commodity flow problem as a very simple constraint structure. The problem has a single constraint for each arc (i, j) which states that the sum of path flows passing throughout the arc is at most u_{ij}, the capacity of the arc.

Moreover, the problem has a single constraint (2.12) for each commodity k which states that the flow on all paths connecting the source node s^k and sink node t^k of commodity k must equal the demand d^k for this commodity.

Example 2.4.1. *For a network with n nodes, m arcs, and K commodities, paths flow formulation contains m + K constraints (in addition to nonnegativity restrictions imposed on the path flow values). In contrast, the arcs flow formulation (2.1) contains m + nK constraints since it contains one mass balance constraint for every node and commodity combination. For example, a network with n = 1.000 nodes and m = 5.000 arcs and with a commodity between every pair of nodes has approximately K $\approx$ n² = 1.000.000 commodities. Therefore, paths flow formulation contains about 1.005.000 constraints. In contrast, arcs flow formulation (2.1) contains about 1.000.005.000 constraints. But the difference is even more pronounced; because no path appears in more than one of the constraints (2.12), we can apply a specialized version of a simplex method, known as generalized upper bounding simplex method (see [2]), to solve paths flow formulation very efficiently. Even though linear programming basis for our example has size 1.005.000 times 1.005.000, generalized upper bounding simplex method is able to perform all of its matrix computations on a much smaller basis of size 5.000 times 5.000. This method essentially solves the problem as though as it contained only m bundle constraints, which, for this sample data, means that we can essentially solve a linear program with only 5.000 constraints instead of over 1 billion constraints in arcs flow formulation.*

However, this savings in the number of constraints does come at a cost since paths flow formulation has a variable for every path connecting a source and sink node for each of the commodities. The number of variables will typically be enormous, growing exponentially in size of a network. On the other hand, we might expect that only very few paths will carry flow in the optimal solution of the problem. In fact, linear programming theory permits us to show that at most K + m paths carry positive flow in some optimal solution of the problem (see [2]). Therefore, for a problem with 1.000.000 commodities and 5.000 arcs we could, in principle, solve paths flow formulation using 1.005.000 paths. Since the problem contains 1.000.000 commodities, this solution would use two or more paths for at most 5.000 of them and one path for at least the 995.000 remaining ones. If we knew the optimal set of paths, or a very good set of paths, we could obtain an optimal solution (i.e., values for path flows) by solving a linear program containing just commodities with two or more sets of paths. Generalized upper bounding linear programming procedure for solving linear programs permits us to exploit this observation.

Optimality conditions

Since paths flow formulation (2.10) contains one bundle constraint (2.11) for each arc and one demand constraint (2.12) for each node-commodity combination, dual linear program

has a dual variable w_{ij} for each arc and another dual variable σ^k for each commodity $k = 1, 2, \ldots, K$. With respect to these dual variables, the reduced cost $c_P^{\sigma,w}$ for each path flow $f(P)$ is

$$c_P^{\sigma,w} = c^k(P) + \sum_{(i,j)\in P} w_{ij} - \sigma^k.$$

and the complementary slackness conditions has the following form:

Proposition 2.4.2 (**Path flow complementary slackness conditions**). *Commodity path flows $f(P)$ are optimal in the paths flow formulation (2.10) of multi-commodity flow problem if and only if for some arc prices w_{ij} and commodity prices σ^k, the reduced costs and arc flows satisfy the following complementary slackness conditions:*

$$w_{ij}\left[\sum_{1\le k\le K}\sum_{P\in \mathbf{P^k}}\delta_{ij}(P)f(P) - u_{ij}\right] = 0, \quad \forall (i,j) \in A. \tag{2.14}$$

$$c_P^{\sigma,w} \ge 0 \quad \forall k = 1, 2, \ldots, K \text{ and } \forall P \in \mathbf{P^k}. \tag{2.15}$$

$$c_P^{\sigma,w} f(P) = 0 \quad \forall k = 1, 2, \ldots, K \text{ and } \forall P \in \mathbf{P^k}. \tag{2.16}$$

These optimality conditions have a very appealing and intuitive interpretation. Condition (2.14) states that the price w_{ij} of arc (i, j) is zero if the optimal solution $f(P)$ does not use all of the capacity u_{ij} of the arc. That is, if optimal, a solution does not fully use the capacity of that arc and we could ignore the constraints (place no price on it).

Since cost $c^k(P)$ of path P is just the sum of costs of arcs contained in that path, i.e., $c^k(P) = \sum_{(i,j)\in P} c_{ij}^k$, we can write the reduced cost of path P as

$$c_P^{\sigma,w} = \sum_{(i,j)\in P}(c_{ij}^k + w_{ij}) - \sigma^k.$$

That is, the reduced cost of path P is just the cost of that path with respect to the modified costs $c_{ij}^k + w_{ij}$ minus the commodity cost σ^k. The complementary slackness condition (2.15) states that the modified path cost $\sum_{(i,j)\in P}(c_{ij}^k + w_{ij})$, for each path connecting the source node s^k and the sink node t^k of commodity k, must be at least as large as the commodity cost σ^k. The condition (2.16) implies that the reduced cost $c_P^{\sigma,w}$ must be zero for any path P that carries flow in the optimal solution (i.e., for which the flow $f(P)$ is positive); that is, the modified cost $\sum_{(i,j)\in P}(c_{ij}^k + w_{ij})$ of this path must equal the commodity cost σ^k. Therefore, conditions (2.15) and (2.16) imply:

Corollary 2.4.3. *σ^k is the shortest path distance from node s^k to node t^k with respect to the modified costs $c_{ij}^k + w_{ij}$ and in the optimal solution every path from node s^k to node t^k that carries a positive flow must be the shortest path with respect to the modified costs.*

This result shows that the arc costs w_{ij} permit us to decompose multi-commodity flow problem into a set of independent "modified" cost shortest path problems.

Column generation solution procedure

Up to this point we have restated *multi-commodity flow problem* as a large-scale linear program with an enormous number of columns, with one flow variable for each path connecting the source and the sink of any commodity. We have also shown how to characterize any optimal solution of this formulation in terms of linear programming dual variables w_{ij} and σ^k, interpreting these conditions as shortest path conditions (subsection 1.3) with respect to the modified arc costs $c_{ij}^k + w_{ij}$. We next show how to solve the problem by using a solution procedure known as *column generation*.

The key idea in column generation is to never explicitly list all columns of the problem formulation, but rather to generate them only "as needed". The revised simplex method of linear programming is perfectly suited for carrying out this algorithmic strategy. Recall from subsection 1.5 that the revised simplex method maintains a basis $\mathcal{B}$ at each iteration. It uses this basis to define a set of simplex multipliers π via the matrix computation $\pi\mathcal{B} = c_B$ (in our application, the multipliers are w and σ). That is, the method defines simplex multiplier so that the reduced costs c_B^π of the basic variables are zero, i.e., $c_B^\pi = c_\mathcal{B} - \pi\mathcal{B} = 0$. To find simplex multipliers, the method requires no information about columns (variables) which are not in the basis. It then uses the multipliers to *price-out* nonbasic columns, i.e., compute their reduced costs. If any reduced cost is negative (assuming a minimization formulation), the method will introduce one nonbasic variable into the basis in place of one of the current basic variables, recompute simplex multipliers π, and then repeat these computations. To use column generation approach, columns should have structural properties that permit us to perform the pricing out operations without explicitly examining every column.

When applied to paths flow formulation of multi-commodity flow problem, with respect to the current basis at any step (which is composed of a set of columns, or path variables, for the problem), the revised simplex method defines simplex multipliers w_{ij} and σ^k so that the reduced cost of every variable in the basis is zero. Therefore, if a path P connecting the source s^k and sink t^k for commodity k is one of the basic variables, then $c_P^{\sigma,w} = 0$, or equivalently, $\sum_{(i,j)\in P}(c_{ij}^k + w_{ij}) = \sigma^k$. Therefore, the revised simplex method determines simplex multipliers w_{ij} and σ^k so that they satisfy the following equations:

$$\sum_{(i,j)\in P}(c_{ij}^k + w_{ij}) = \sigma^k \quad \text{for every path } P \text{ in the basis.}$$

Notice that since each basis consists of $K + m$ paths, each basis gives rise to $K + m$ of these equations. Moreover, the equations contain $K + m$ variables (i.e., m arc prices w_{ij} and K

shortest path distances σ^k). The revised simplex method uses matrix computations to solve $K + m$ equations and determines the unique values of simplex multipliers.

The complementary slackness condition (2.16) dictates that $c_P^{\sigma,w} f(P) = 0$ for every path P in the network. Since each path P in the basis satisfies the condition $c_P^{\sigma,w} = 0$, we can send any amount of flow on it and still satisfy the condition (2.16). To satisfy this condition for a path P which is not in the basis, we set $f(P) = 0$. Next consider the complementary slackness condition (2.14). If the slack variable $s_{ij} = \left[\sum_{1 \le k \le K} \sum_{P \in \mathbf{P^k}} \delta_{ij}(P) f(P) - u_{ij} \right]$ is not in the basis, $s_{ij} = 0$, so the solution satisfies condition (2.14). On the other hand, if the slack variable s_{ij} is in the basis, its reduced cost, which equals $0 - w_{ij}$, is zero, implying that $w_{ij} = 0$ and the solution satisfies condition (2.14). We have thus shown that the solution defined by the current basis satisfies conditions (2.14) and (2.16); it is optimal if it satisfies condition (2.15) (i.e., the reduced cost of every path flow variable is nonnegative). How can we check this condition? That is, how can we see if for each commodity k,

$$c_P^{\sigma,w} = \sum_{(i,j) \in P} (c_{ij}^k + w_{ij}) - \sigma^k \ge 0 \quad \forall P \in \mathbf{P^k},$$

or, equivalently,

$$\min_{P \in \mathbf{P^k}} \sum_{(i,j) \in P} (c_{ij}^k + w_{ij}) \ge \sigma^k?$$

As we have noted, the left-hand side of this inequality is just the length of the shortest path connecting the source and sink nodes, s^k and t^k, of commodity k with respect to the modified costs $c_{ij}^k + w_{ij}$. Thus, to see whether the arc prices w_{ij} together with current path distances σ^k satisfy the complementary slackness conditions, we solve the shortest path problem for each commodity k. If for all commodities k, the length of the shortest path for that commodity is at least as large as σ^k, we satisfy the complementary slackness condition (2.15).

Otherwise, if for some commodity k, Q denotes the shortest path with respect to the current modified cost $c_{ij}^k + w_{ij}$ and the modified cost of the path Q is less than the length σ^k of the minimum cost path from the set $\mathbf{P^k}$, then

$$c_Q^{\sigma,w} = \sum_{(i,j) \in Q} (c_{ij}^k + w_{ij}) - \sigma^k < 0.$$

In terms of a linear program (2.10), the path Q has a negative reduced cost, so we can profitably use it in the linear program in place of one of the paths P in the current basis $\mathcal{B}$. That is, using the usual steps of the simplex method, we would perform a basis change introducing the path Q into the current basis. Doing so would permit us to determine a new set of arc prices w_{ij} and a new modified shortest path distance σ^k between the source and sink nodes of commodity k. We choose the values of these variables so that the reduced

cost of every basic variable is zero. That is, using matrix operations, we would once again solve the system $c_P^{\sigma,w} = \sum_{(i,j)\in P}(c_{ij}^k + w_{ij}) - \sigma^k = 0$ in the variables w_{ij} and σ^k. We would then, as before, solve the shortest path problem for each commodity k and see whether any path has a shorter length than σ^k. If so, we would introduce this path into the basis and continue by alternately (1) finding new values for the arc prices w_{ij} and for the path lengths σ^k, and (2) solving shortest path problems.

This discussion shows us how we would determine variable to introduce into the basis at each step. The rest of the steps for implementing the simplex method (e.g., determining the variable to remove from the basis at each step) are the same as those of the usual implementation, so we do not specify any further details.

Determining lower bounds

Let z^* denote the optimal objective function value of multi-commodity flow problem (2.10) and let z^{lp} denote the objective function value at any step in solving paths flow formulation of the problem (2.10) by the simplex method. Since z^{lp} corresponds to a feasible solution to the problem, $z^* \leq z^{lp}$. As we have noted in theorem 2.3.2, for any choice of the arc prices w, the optimal value $L(w)$ of Lagrangian subproblem is a lower bound on z^*. Therefore, suppose that at any point during the course of the algorithm, we solve Lagrangian subproblem with respect to the current arc prices w_{ij}. That is, we solve for the shortest path lengths $l^k(w)$ for all commodities k with respect to the modified costs $c_{ij}^k + w_{ij}$. (notice that this is the same computation that we perform in pricing out columns for the simplex method). Then from (2.7) the value $L(w)$ of Lagrangian subproblem is

$$L(w) = \sum_{k=1}^{K} l^k(w) - \sum_{(i,j)\in A} w_{ij}u_{ij}$$

and by the theory of Lagrangian relaxation,

$$L(w) \leq z^* \leq z^{lp}.$$

Therefore, as a by-product of finding the shortest path distances $l^k(w)$, as we are pricing out columns in implementing column generation procedure, we obtain a lower bound on the objective function value. This lower bound allows us to judge the quality of the current solution in *column generation technique* and often terminate this procedure without further computation if the difference between the solution value z^{lp} and the lower bound $L(w)$ is sufficiently small. We might note that since at each step of the simplex method, the objective value z^{lp} of the problem stays the same or decreases, the upper bound is monotonically nonincreasing from step to step. On the other hand, the objective function value $L(w)$ of Lagrangian subproblem need not decrease from step to step, so at any point in the

algorithm we would use the largest of the values $L(w)$ generated in all previous steps as the best lower bound.

After this theoretical discussion we are finally ready to move forward to introduce and solve train timetabling problem.

Chapter 3

Train timetabling problem

3.1 Introduction

By this point we have all necessary results for introducing and solving *train timetabling problem*, but before doing that we are going to present a motivation behind the introduced model. This whole chapter is based on Ph. D. thesis written by Valentina Cacchiani (see [3]). There she is dealing with, not only *train timetabling problem*, but with few other railway planning problems as well. For each problem, she stated a model, offered few different approaches for solving it, made computations and compared the results. In this work, we will only deal with *train timetabling problem* and introduce solution based on *column generation technique*.

The general aim of train timetabling problem is to provide a timetable for a number of trains on a certain part of railway network. According to the current situation, an infrastructure manager is handling railway network and receiving requests from train operators, concerning trains to be operated for a given time horizon. Each of these requests specifies a path for a train along with the arrival and departure times for all stations along the path. Given that these requests are mutually incompatible, the infrastructure manager has to solve *train timetabling problem*, modifying the arrival and/or departure times of some trains (and possibly cancelling some other trains), in order to come up with proposed feasible scenario for train operators, who may either accept it (given that they will pay less for the requests that were modified), or come up with a new proposals. The process is iterated until the scenario proposed by the infrastructure manager is accepted by all train operators.

Remark 3.1.1. *In Croatia this process is much simpler since trains are owned by the company owned by the country. Nevertheless, we will introduce a general approach since it might, and probably will, change at some point.*

Given practical relevance of the problem and different organization rules of the various infrastructure managers, some variants of the problem were formulated and solved. They can be roughly classified according to the following criteria:

a) the method focuses on a single line or on a general network

b) the method takes advantage, or not, of the fact that the solution to be constructed is periodic.

As to a), the reason for focusing on a single (main) line, often called corridor, is that, in many cases, once the timetable for trains on the corridor has been determined, it is relatively easy to find a convenient timetable for trains on the other lines of the network. On the other hand, there are other situations in which there are few congested lines, and solving problem on each line separately would be a too rough approach. As to b), in some real cases (nearly) all trains are repeated every period (typically one hour or one day). When this holds, not only the size of the problem is widely reduced by considering only one period, but also mathematical formulation can be stronger. In this work, we will introduce a model described in [3], chapter 2, and illustrate solution method for train timetabling problem on a single line. This solution can be also used when the problem is non-periodic, although the case on which we focus is periodic with period one day. The method uses an integer linear programming formulation and is based on the explicit solution of linear programming relaxation of integer the linear formulation. Before [3], decision variables were almost exclusively associated with nodes/arcs. This new integer linear programming associates variables with paths in a *space-time graph*, which we will introduce in section 3.3. Solution of its linear programming relaxation by separation and column generation techniques will be described in sections 3.6 and 3.7, respectively.

3.2 Train timetabling problem

We consider a single, one-way track corridor linking two major stations, with a number of intermediate stations in between, together with a set of trains that are candidates to be run every day of a given time horizon along the corridor. Let $S = \{1, \ldots, s\}$ represent a set of stations, numbered according to the order in which they appear along the corridor for running direction considered, and $T = \{1, \ldots, t\}$ denote the set of candidate trains. For each train $j \in T$, the *first* (departure) station f_j and the *last* (destination) station l_j ($l_j > f_j$) are given. Let $S^j := \{f_j, \ldots, l_j\} \subseteq S$ be ordered set of stations visited by train j ($j \in T$). The *track capacity constraints* impose that overtaking between trains occurs only within a station. Furthermore, for each station $i \in S$, there are lower bounds a_i and d_i on the time interval between two consecutive arrivals and two consecutive departures, respectively. A *timetable* defines, for each train $j \in T$, departure time from f_j, arrival time at l_j, and arrival

and departure times for the intermediate stations $f_j + 1, \ldots, l_j - 1$. Each train is assigned, by train operator, an ideal timetable, representing the most desirable timetable for the train, that may be modified in order to satisfy the track capacity constraints. In particular, with respect to the ideal timetable, one is allowed to modify (anticipate or delay) departure time of each train from its first station, and to increase (but not decrease) stopping time interval at the intermediate stations. Moreover, one can also *cancel* the train. A timetable for train j in the final solution will be referred to as the *actual timetable*.

Remark 3.2.1. *Observe that, differently from most of other models, we consider version of the problem in which a travel time between consecutive stations must be the same in the ideal and actual timetables. In this new version one can write the overtaking constraints in integer linear programming in a form that is much stronger for linear programming relaxation than in the old versions. Interested reader can check [4] and [5] for models where the time is not constant.*

The *objective* is to maximize sum of the profits of scheduled trains, defined as follows. The *profit* achieved for each train $j \in T$ is given by $\pi_j - \alpha_j \nu_j - \gamma_j \mu_j$, where π_j is the *ideal profit*, that is profit that is achieved if the train travels according to its ideal timetable, ν_j is the *shift*, that is the absolute difference between a departure times from station f_j in the ideal and actual timetables, μ_j is the *stretch*, that is the (nonnegative) difference between a travel time from f_j to l_j in the actual and ideal timetables (equal to the sum of the stopping time increases over all intermediate stations), and α_j, γ_j are given nonnegative parameters.

Remark 3.2.2. *Note that it would be easy to impose additional constraints on a train timetables in the model, such as the requirement that a train does not arrive at a station later than a certain time instant, to guarantee a connection with a train on a different line whose timetable is fixed. However, we do not consider these constraints in our work.*

For convenience, from now on, we will use the acronym TTP to denote train timetabling problem, ILP to denote integer linear programming and LP to denote linear programming.

3.3 Space-time graph representation

We next outline the representation of the problem on a graph. Times are here discretized and expressed as integers from 1 to $q := 1440$ (the number of minutes in a day), though a finer discretization would also be possible (e.g., 1/2, 1/4 of a minute) without changing the model, although time and space complexity of the associated algorithms could increase considerably.

Let $G = (N, A)$ be a (directed, acyclic) space-time multigraph in which nodes represent arrivals/departures at a station in a given time instants, and paths represent feasible timetables for trains, defined as follows. The node set N has the form $\{\sigma, \tau\} \cup (U^2 \cup \ldots \cup U^s) \cup (W^1 \cup \ldots \cup W^{s-1})$, where σ and τ are an *artificial source node* and an *artificial sink node*, respectively. Whereas sets U^i, $i \in S \setminus \{1\}$, and W^i, $i \in S \setminus \{1\}$, represent a set of time instants in which some train can arrive at and depart from station i, respectively. We call a nodes in $U^2 \cup \ldots \cup U^s$ and $W^1 \cup \ldots \cup W^{s-1}$ *arrival* and *departure* nodes, respectively.
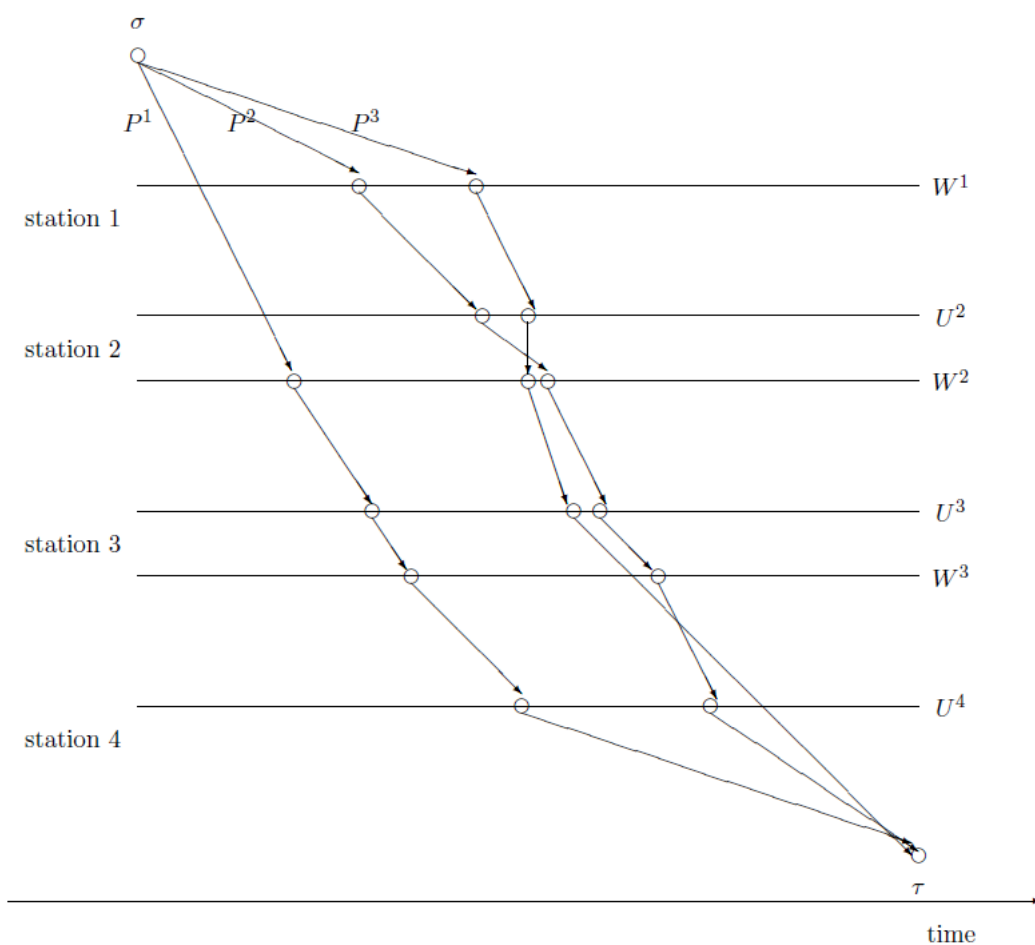

Figure 3.1: An example of train paths in graph G (with $s = 4$, $t = 3$, $f_1 = 2$, $l_1 = 4$, $f_2 = 1$, $l_2 = 4$, $f_3 = 1$, $l_3 = 3$)

Let $\theta(v)$ be time instant associated with a given node $v \in N$. Moreover, let

$$\Delta(u, v) := \begin{cases} \theta(v) - \theta(u), & \text{if } \theta(v) \geq \theta(u) \\ \theta(v) - \theta(u) + q, & \text{otherwise} \end{cases}$$

Because of the "cyclic" nature of time horizon, we say that node u *precedes* node v (i.e., $u \preceq v$) if $\Delta(v, u) \geq \Delta(u, v)$ (i.e., if the cyclic time interval between $\theta(v)$ and $\theta(u)$ is not smaller than the cyclic time interval between $\theta(u)$ and $\theta(v)$).

Note that not all time instants correspond to possible arrivals/departures of train j at the station $i \in S^j$. Accordingly, let $N^j \subseteq \{\sigma, \tau\} \cup (U^2 \cup \ldots \cup U^s) \cup (W^1 \cup \ldots \cup W^{s-1})$ denote set of nodes associated with time instants corresponding to possible arrivals/departures of train j in a positive-profit timetable. The arc set A is partitioned into sets $A^1, \ldots, A^t$, one for each train $j \in T$. In particular, for every train $j \in T$, A^j contains:

- a set of *starting* arcs (σ, v), for each $v \in W^{f_j} \cap N^j$, whose profit is $p_{(\sigma, v)} := \pi_j - \alpha_j \nu(v)$, with $\nu(v) := \min\{\Delta(v^*, v), \Delta(v, v^*)\}$, where v^* is node associated with a departure of train j from station i in the ideal timetable;

- a set of *station* arcs (u, v), for each $i \in S^j \setminus \{f_j, l_j\}$, $u \in U^i \cap N^j$ and $v \in W^i \cap N^j$ such that $\Delta(u, v)$ is at least equal to the minimum stop time of train j in station i, whose profit is $p_{(u,v)} := -\gamma_j \mu(u, v)$, with $\mu(u, v) := \Delta(u, v) - \Delta(u^*, v^*)$, where u^* and v^* are the nodes associated, respectively, with the arrival and departure of train j at station i in the ideal timetable;

- a set of *segment* arcs (v, u), for each $i \in S^j \setminus \{l_j\}$, $v \in W^i \cap N^j$ and $u \in U^{i+1} \cap N^j$ such that $\Delta(v, u)$ is equal to the travel time of train j from station i to station $i + 1$, whose profit $p_{(v,u)} := 0$;

- a set of *ending* arcs (u, τ), for each $u \in U^{l_j} \cap N^j$, whose profit is $p_{(u,\tau)} := 0$.

By construction, for each train $j \in T$, any path from σ to τ in G which uses only arcs in A^j and has profit p, corresponds to a feasible timetable for train j having profit p.

To satisfy the track capacity constraints, one should impose that certain pairs of arcs, associated with different trains, cannot be selected in the overall solution. This is discussed in a detail in the following section.

3.4 Integer linear programming model

Solution approaches in the literature that are based on a graph representation of the previous section define ILP formulations in which variables are associated with nodes and/or arcs of the graphs. As the associated LP relaxations turn out to be extremely expensive to solve exactly, even by the state-of-the-art LP solvers, a heuristic solution of their dual is

obtained by combining Lagrangian relaxation of the track capacity constraints with sub-gradient optimization. In this section we illustrate an alternative solution approach, based on an alternative (and equally natural) ILP formulation in which variables are associated with paths in G.

In a new ILP model there is a binary variable for each possible path for a train, indicating if the path is chosen or not in the solution. Considering that for each train $j \in T$ the corresponding path can be shifted and/or stretched with respect to the ideal path, i.e., the path associated with the ideal timetable of train j, there is an exponentially large number of possible paths for a train, as will be discussed next. Therefore, our solution approach is based on column generation technique (see section 2.4), that is easy to apply as column generation problem calls for an optimal path in the (acyclic) graph considered. Moreover, we will have a very large number of constraints, as was the case for the previous ILP formulations. These constraints will be handled by separation algorithms (introduced in section 3.7), which in this case have to deal with fractional solutions, differently from the case in which Lagrangian relaxation is used.

Before introducing ILP model we should extend definitions introduced in section 3.3. Given two consecutive stations i and $i + 1$ along with two trains j and k such that i, $i + 1 \in S^j \cap S^k$, let

$$b_i^{jk} := \max\{d_i, a_{i+1} + r_j - r_k\}$$

denote the minimum time interval between the departure of j from i and the departure of k from i (in this order) in a feasible solution, where r_j and r_k are the travel times of j and k from i to $i + 1$, respectively. Note that if j and k depart from i within a time interval smaller than b_i^{jk}, either their departures are too close in time, or their arrivals are too close in time, or k overtakes j between i and $i + 1$.

For each $j \in T$, let $\mathcal{P}^j$ be the collection of possible paths for train j, each associated with a path from σ to τ in G containing only arcs in A^j (paths are seen as arc subsets) and having positive profit. Furthermore, let $\mathcal{P} := \mathcal{P}^1 \cup \ldots \cup \mathcal{P}^t$ be the overall (multi-)collection of paths, denoting by $p_P := \sum_{a \in P} p_a$ the actual profit for path $P \in \mathcal{P}$. Finally, let $\mathcal{P}_w^j \subseteq \mathcal{P}^j$ be a (possibly empty) subcollection of paths for train j that visit node $w \in N$. Now we have notation needed for defining our ILP model.

Letting x_P, $P \in \mathcal{P}$, be a binary variable that is equal to 1 if and only if the path $P \in \mathcal{P}$ is selected in an optimal solution, ILP model is the following:

$$\max \sum_{P \in \mathcal{P}} p_P x_P \tag{3.1}$$

subject to

$$\sum_{P \in \mathcal{P}^j} x_P \leq 1, \quad j \in T, \tag{3.2}$$

$$\sum_{w \in U^i : w \geq u, \Delta(u,w) < a_i} \sum_{P \in \mathcal{P}_w} x_P \leq 1, \quad i \in S \setminus \{1\}, \ u \in U^i, \tag{3.3}$$

$$\sum_{w \in W^i : w \geq v, \Delta(u,w) < d_i} \sum_{P \in \mathcal{P}_w} x_P \leq 1, \quad i \in S \setminus \{s\}, \ v \in W^i, \tag{3.4}$$

$$\sum_{w \in W^i \cap N^j : v_1 \leq w < v_2} \sum_{P \in \mathcal{P}_w^j} x_P \ + \sum_{w \in W^i \cap N^j : w \geq v_2, \Delta(v_2,w) < b_i^{kj}} \sum_{P \in \mathcal{P}_w^j} x_P \ +$$
$$+ \sum_{w \in W^i \cap N^k : w \geq v_2, \Delta(v_1,w) < b_i^{jk}} \sum_{P \in \mathcal{P}_w^k} x_P \leq 1, \tag{3.5}$$

$$i \in S \setminus \{s\}, \ j, k \in T(\text{with } j \neq k; i, i+1 \in S^j \cap S^k),$$

$$v_1, v_2 \in W^i(\text{with } v_1 \leq v_2, \Delta(v_1, v_2) < b_i^{jk}),$$

$$x_P \geq 0, \quad P \in \mathcal{P} \tag{3.6}$$

$$x_P \text{ integer}, \quad P \in \mathcal{P}. \tag{3.7}$$

The interpretation of the objective function (3.1) and the constraints (3.2), that impose that at most one path for each train is selected, poses no problem. The *arrival time* constraints (3.3) and the *departure time* constraints (3.4), prevent two consecutive arrivals and departures at the same station i to be too close in time. The *overtaking* constraints (3.5) are formally more complex. These constraints are defined by specifying a station $i \in S \setminus \{s\}$, two trains $j, k \in T$ such that $i, i + 1 \in S^j \cap S^k$, and two nodes $v_1, v_2 \in W^i$ such that $v_1 \geq v_2$ and $\Delta(v_1, v_2) < b_i^{jk}$. By definition of b_i^{jk}, we must have $x_{P_1} + x_{P_2} \leq 1$ for all $P_1 \in \mathcal{P}_{v_1}^j$ and $P_2 \in \mathcal{P}_{v_2}^k$, which would be the weakest possible form of overtaking constraints. A very simple strengthening would be $\sum_{P \in \mathcal{P}_{v_1}^j} x_P + \sum_{P \in \mathcal{P}_{v_2}^k} x_P \leq 1$. By proceeding further, taking also into account the arrival and departure constraints, one gets to the constraints (3.5), which are maximal in the sense that no other variable x_P for $P \in \mathcal{P}^j \cup \mathcal{P}^k$ could be added to the left-hand side maintaining validity. Specifically, nodes v_1 and v_2 represent the earliest departure times from i for trains j and k, respectively, involved in the constraints (3.5). Stated in words, constraints (3.5) forbid the simultaneous departure from station i of train j at a time instant between $\theta(v_1)$ and $\theta(v_2) + b_i^{kj} - 1$ and of train k at a time instant between $\theta(v_2)$ and $\theta(v_1) + b_i^{jk} - 1$. Since the departure of j at time $\theta(v_2) + b_i^{kj}$ is compatible with the departure of k at time $\theta(v_2)$, and the departure of k at time $\theta(v_1) + b_i^{jk}$ is compatible with the departure of j at time $\theta(v_1)$, the time windows in the constraint cannot be enlarged. Finally, note that, although ILP model would be valid also without constraints (3.3) and (3.4), as arrival/departures too close in time are forbidden also by constraints (3.5), the former lead to a stronger LP relaxation and are also much faster to separate in practice, so it is much

better to keep them in the formulation.

Note that ILP model above is of *set packing type*, and is associated with the stable set problem defined on a graph with one node for each path $P \in \mathcal{P}$, with associated profit p_P, and one edge joining each pair of nodes corresponding to paths that cannot be both selected in the solution, i.e., that have both coefficient 1 in at least one of the inequalities.

3.5 Solution of linear programming relaxation

As already mentioned, a number of variables in the new ILP formulation can be very large. Rough bounds on a number of paths associated with each train $j \in T$ are the following, letting $v_j^{\max} := \max\{v : \pi_j - \alpha_j v > 0\}$ and $\mu_j^{\max} := \max\{\mu : \pi_j - \gamma_j \mu > 0\}$ be the maximum shift and stretch, respectively, in a timetable for train j that has positive profit. It is not difficult to see that, once the shift for a path of train j is fixed, the number of paths that accumulate a total stretch not larger than $\mu_j^{\max}$ in the $m_j := l_j - f_j - 1$ intermediate stations for train j is equal to the number of vectors with m_j integer nonnegative components whose sum is at most $\mu_j^{\max}$, which is equal to $\binom{\mu_j^{\max} + m_j}{m_j}$. Accordingly, we have that $|\mathcal{P}^j|$ is at least equal to the number of paths with zero shift and stretch not larger than $\mu_j^{\max}$, i.e., $|\mathcal{P}^j| \geq \binom{\mu_j^{\max} + m_j}{m_j}$ and at most equal to the number of paths with shift not larger than $v_j^{\max}$ and stretch not larger than $\mu_j^{\max}$, i.e., $|\mathcal{P}^j| \leq (2v_j^{\max} + 1)\binom{\mu_j^{\max} + m_j}{m_j}$. In any case, $|\mathcal{P}^j|$ may be exponential in m_j, and for any bigger network this makes it absolutely impractical to consider explicitly all paths in $\mathcal{P}^j$.

The natural approach to tackle such a large number of variables is to use column generation technique for a solution of LP relaxation, working with an LP with only a subset of the variables (at the beginning, e.g., only the variables associated with the ideal path for each train), adding variables with positive reduced profit to the LP at each iteration. This is illustrated in detail in section 3.6.

Although not exponential, also a number of constraints in a new ILP formulation is fairly large. Specifically, we have $\mathcal{O}(|N|)$ arrival and departure constraints (3.3) and (3.4), and $\mathcal{O}(t^2 |N| b^{\max})$ overtaking constraints (3.5) where $b^{\max}$ is the maximum value of b_i^{jk} over all stations i and train pair j, k. Accordingly, rather than imposing all these constraints since the beginning, it is much better to handle them by using separation techniques, as illustrated in section 3.7.

Note that the column generation problem simply amounts to the computation of an optimal path in an acyclic graph, and the effect of adding new constraints with nonnegative dual variables simply corresponds to changing the "penalties" of some nodes in this path computation. Accordingly, the combination of separation and column generation poses no serious problem in our case (i.e., the addition of new constraints does not destroy the structure of the column generation problem, as opposed to what is frequently the case).

The general structure of our method to solve LP relaxation is the following:

1. We initialize a reduced LP with only t variables associated with the ideal paths for each train, and constraints (3.2) and (3.6);

2. We solve the reduced LP, obtaining primal solution x^* and dual solution y^*;

3. We apply column generation: if variables with positive reduced profit with respect to y^* are found, we add them to the reduced LP and go to Step 2;

4. We apply separation for constraints (3.3) and (3.4): if constraints violated by x^* are found, we add them to the reduced LP and go to Step 2;

5. We apply separation for constraints (3.5): if constraints violated by x^* are found, we add them to the reduced LP and go to Step 2;

6. We terminate since x^*, y^* is an optimal primal-dual pair for the whole LP. It satisfy all the constraints, (3.1) to (3.6).

The reason for separating constraints (3.3) and (3.4) before constraints (3.5) is that the former are stronger as well as faster to check for violation. On the other hand, the choice to perform column generation before separation was motivated by experimental evidence (one can check the results in [3]), showing that convergence is faster if we start separation only when the current primal solution is optimal with respect to the constraints in the reduced LP. Of course, in the first iteration there are no variables with positive reduced profit, and separation amounts to checking feasibility of the solution defined by the ideal timetables.

3.6 Column generation for linear programming relaxation

The column generation problem is solved by considering in turn each train $j \in T$ and checking if there exists a path $P \in \mathcal{P}^j$ with positive reduced profit.
Let the dual variables associated with constraints (3.2), (3.3), (3.4) and (3.5) be γ_j ($j \in t$), α_u ($i \in S \setminus \{1\}$, $u \in U^i$), β_v ($i \in S \setminus \{s\}$, $v \in W^i$), and δ_{j,k,v_1,v_2} ($i \in S \setminus \{s\}$, $j, k \in T, v_1, v_2 \in W^i$) satisfying the requirements in (3.5). Moreover, for $i \in S \setminus \{1\}$ and $w \in U^i$, let $\zeta_w := \sum_{u \in U^i : u \leq w, \Delta(u,w) < a_i} \alpha_u$, and, similarly, for $i \in S \setminus \{s\}$ and $w \in W^i$, let $eta_w := \sum_{v \in W^i : v \leq w, \Delta(v,w) < d_i} \beta_v$. Finally, for $l \in T$, $i \in S \setminus \{l_l, \ldots, s\}$ and $w \in W^i$, let ξ_{lw} be the sum of variables δ_{j,k,v_1,v_2} over all constraints for which l is either j or k and node w is one of the nodes in the summations for train l in constraints (3.5) (i.e., the first two summations if

$l = j$ and the third one if $l = k$).

For a path $P \in \mathcal{P}^j$, let U_P and W_P be, respectively, the set of arrival and departure nodes visited by path P. The reduced profit of path P is given by:

$$p_P - \gamma_j - \sum_{u \in U_P} \zeta_u - \sum_{v \in W_P} (\eta_v + \xi_{jv})$$

Accordingly, one may find path $P \in P^j$ with the maximum reduced profit by finding, in $\mathcal{O}(|A^j|)$ time, the path of the maximum profit from σ to τ in G that uses only arcs in A^j, proceeding in a completely analogous way as in the solution of Lagrangian relaxation in [4], where the profit of a path $P \in P^j$ is now given by

$$\sum_{a \in P} p_a - \sum_{u \in U_P} \zeta_u - \sum_{v \in W_P} (\eta_v + \xi_{jv})$$

Remark 3.6.1. *Note that both in [4] and here profits are associated both with nodes and arcs of G, and they can be handled in a way completely analogous to the case of arc profits only.*

If the maximum profit is not larger than γ_j, then all variables x_P, $P \in \mathcal{P}^j$, have nonpositive reduced profit. Otherwise, the path found is associated with the variable with the most positive reduced profit.

3.7   Separation

Separation of constraints (3.3), (3.4) and (3.5) is done by trivial enumeration, as it is the case within the relax-and-cut procedure associated with Lagrangian relaxation, with the difference that we have to deal with fractional variable values in this case. Specifically, for constraints (3.3) and (3.4), for each node $v \in N$ we initialize a value y_w^* to 0. Then, we consider in turn each positive variable x_P^* in the reduced LP solution and increase the value y_w^* of all nodes $w \in U_P \cup W_P$ by x_P^*. Finally, for each station $i \in S \setminus \{1\}$ and node $u \in U^i$, we test if $\sum_{w \in U^i: w \geq u, \Delta(u,w) < a_i} y_w^* > 1$, in which case constraint (3.3) associated with node u is violated. The computation of the node values can be carried out in $\mathcal{O}(sn)$ time, where n is the number of positive variables in the reduced LP solution, and the subsequent check is easily done in $\mathcal{O}(|N|)$ time by using accumulators to store, for each node $u \in U^i$, the sum of the values of consecutive nodes (i.e., of nodes associated with consecutive time instants) in U^i, starting from an (arbitrarily chosen) initial node and ending in u. The check for constraints (3.4) is perfectly analogous.

For constraints (3.5), for each train $j \in T$ and for each node $v \in N^j$ we first initialize a value z_{jv}^* to 0, and consider in turn each positive variable x_P^* in the reduced LP solution for

$P \in \mathcal{P}^j$, increasing the value of all nodes in W_P by x_P^*. Then, we enumerate all pairs of trains $j, k \in T$ whose paths may be in conflict (by determining, for each train $j \in T$ and once for all, the set of trains whose paths may be in conflict with a path of j), and all stations $i \in \{f_j, \ldots, l_j-1\} \cap \{f_k, \ldots, l_k-1\}$. For each pair of nodes $v_1, v_2 \in W^i$, $v_1 \leq v_2$, $\Delta(v_1, v_2) < b_i^{jk}$ we have that constraint (3.5) corresponding to j, k, v_1, v_2 is violated if and only if

$$\sum_{w \in W^i \cap N^j : v_1 \leq w < v_2} z_{jw}^* + \sum_{w \in W^i \cap N^j : w \geq v_2, \Delta(v_2, w) < b_i^{kj}} z_{jw}^* + \sum_{w \in W^i \cap N^k : w \geq v_2, \Delta(v_1, w) < b_i^{jk}} z_{jw}^* > 1.$$

The time complexity of the above procedure to separate constraints (3.5) is again $\mathcal{O}(sn)$ for the computation of the values, and then $\mathcal{O}(t^2|N|b^{\max})$ for the subsequent check, since, by using accumulators similar to those used for the separation of constraints (3.3) and (3.4), each constraint can be checked in constant time.

3.8 Conclusion

We have finally reached the goal of our work. After lot of preliminary theoretical results, we introduced a model for representing *train timetable problem*. This model, as opposed to the usual practice, has decision variables associated with paths instead of nodes/arcs. In her Ph.D. thesis [3], Valentina Cacchiani together with her team, gave an algorithm for solving it. Because of our results in sections 1 and 2 it is easy to understand each of the 6 steps in the algorithm. Furthermore, since the train timetable problem is "broken" into a lot smaller ones, today's state of technology allows us to solve them efficiently. Key role in our algorithm has column generation technique (see section 2.4) which can be solved quite cheaply using revised simplex method. One can, understanding this work, very quickly implement the algorithm for a single line network to get a periodic timetable. It is even possible to use this model for a non-periodic case but there exist better models which deal with it so we do not recommend it.

To conclude, even though there existed few models with decision variables associated with paths, this new model represents a turnaround at the way we look at the train timetabling problem. Especially because using *column generation technique* we can drastically reduce the important information for solving it. Therefore, the problem is solved in more elegant, understandable and efficient way.

Bibliography

[1] Ravindra K Ahuja, Claudio B Cunha, and Güvenç Sahin, *Network models in railroad planning and scheduling*, TutORials in operations research **1** (2005).

[2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin, *Network flows: theory, algorithms, and applications*, Prentice hall, 1993.

[3] Valentina Cacchiani, *Models and algorithms for combinatorial optimization problems arising in railway applications*, (2009).

[4] Alberto Caprara, Matteo Fischetti, and Paolo Toth, *Modeling and solving the train timetabling problem*, Operations research **50** (2002), no. 5.

[5] Alberto Caprara, Michele Monaci, Paolo Toth, and Pier Luigi Guida, *A lagrangian heuristic algorithm for a real-world train timetabling problem*, Discrete applied mathematics **154** (2006), no. 5.

[6] Institute for Operations Research, the Management Sciences. National Meeting, and J Cole Smith, *Tutorials in operations research: Emerging theory, methods, and applications*, INFORMS, 2005.

Summary

This work is dealing with *train timetabling problem.* In the first chapter, one can find an introduction to *network flows* which is needed for understanding deeper concepts later on. Namely, basic graph theory definitions are stated as well as core problems like the *minimum cost flow* and *shortest path problem.* Furthermore, two equivalent representations of network flows are described, including some useful properties connected to each of them. At the end of the chapter, *linear programming* and *simplex method* are introduced into some detail.

In the second chapter more complex theory is introduced. At the beginning, *multi-commodity flow problem* is stated and few solutions approaches are briefly described. Once we settled for one of them, the rest of the chapter is dealing with *Lagrangian relaxation* and *column generation* techniques. Since column generation is the main result needed for solving our problem, some finer results, like determining lower and upper bounds, are stated.

In the last, third chapter, one can find a model for representing *train timetabling problem* for a single line network. That model was introduced by Valentina Cacchiani in her Ph. D. thesis. In this work, periodicity of timetable is assumed because it makes computations way quicker, as well as it has some other benefits. At the end, one can find an algorithm based on column generation technique for solving introduced model. That algorithm is based on 6 steps, and after reading this work, one should be able to fully understand each of them.

Sažetak

Ovaj rad bavi se *problemom rasporeda vožnje u željezničkom prometu*. U prvom poglavlju nalazi se uvod u *mrežne tokove* koji je potreban za razumijevanje naprednijih koncepata. Konkretno, iskazane su osnovne definicije teorije grafova kao i neki temeljni problemi poput *problema najjeftinijeg toka* i *problema najkraćeg puta*. Nadalje, opisana su dva ekvivalenta prikaza *mrežnih tokova*, uključujući neka korisna svojsta za svaki od njih. Na kraju poglavlja, *linearno programiranje* i *simpleks metoda*, objašnjeni su na razini razumijevanja.

U drugom poglavlju nalazi se naprednija teorija koja se nadovezuje na prvo poglavlje. Na početku poglavlja prikazan je *problem više dobara*, kao i nekoliko pristupa rješavanju navedenog problema. Nakon što smo se odlučili za jedan od pristupa, ostatak poglavlja bavi se *Lagrangeovom relaksacijom* i *metodom generacije stupaca*. Kako je upravo metoda generacije stupaca najvažniji rezultat za rješavanje našega problema, napredniji rezultati vezani uz određivanje donjih i gornjih granica su detaljno objašnjeni.

U posljednjem, trećem poglavlju, nalazi se model za prikazivanje problema rasporeda vožnje za mreže s jednom tračnicom. Navedni model prvi puta je predstavljen u doktorskom radu Valentine Cacchiani. U ovom radu također pretpostavljamo periodičnost rasporeda vožnje kako bismo, između ostalih, ostvarili prednost poput bržeg vremena računanja. Na kraju rada nalazi se algoritam, temeljen na metodi generacije stupaca, za rješavanje predstavljenog modela. Navedeni algoritam sastoji se od 6 koraka, od kojih je svaki detaljno opisan u ovome radu.

Biography

I was born in Našice on 15th of January, 1992. I finished primary school "OŠ Ivane Brlić-Mažuranić", in my hometown Orahovica. After that, I went to mathematical gymnasium "SŠ Izidora Kršnjavoga" in Našice. My education proceeded in 2010. when I became student at University of Zagreb, Faculty of science, Department of mathematics, where I achieved my bachelor's degree in 2013. Somewhere during that period, in 2011., I became *runner up chess champion* of Croatia under 20 years old. Immediately after having my degree in 2013., I became student of masters of mathematical statistics at the same university. In the winter semester of academic year 2013./14., prof. dr. sc. Mirko Polonijo gave me an honour and pleasure to be *officially appointed tutor* for Euclidean spaces course. From 2013. until 2016., I was an active member of a student association eSTUDENT. In 2015., I was an intern in PBZ Croatia Osiguranje, one of four Croatian compulsory pension funds. I spent there three months in *Research department*. In a meantime, I also spent academic year 2015./16. as an Erasmus exchange student at University of Bielefeld in Germany. After that I became part of Morgan Stanley technology analyst program in New York and London. I finished it successfully, and am currently working as *associate software developer* in Morgan Stanley Budapest office.

Životopis

Rođen sam u Našicama 15-og siječnja 1992. godine. Odrastao sam u Orahovici, gdje sam završio sam osnovnu školu "OŠ Ivane Brlić-Mažuranić". Nakon toga sam pohađao matematičku gimnaziju u srednjoj školi "SŠ Izidora Kršnjavoga" u Našicama. Obrazovanje sam nastavio 2010. godine kada sam postao student Sveučilišta u Zagrebu, na Odjelu za matematiku, Prirodoslovno-matematičkog fakulteta gdje sam 2013. godine postao prvostupnik preddiplomskog studija matematike. U međuvremenu, 2011. godine, postao sam *vice-prvak Hrvatske u šahu* u konkurenciji do 20 godina. Odmah nakon prvostupničke diplome 2013. godine, upisao sam diplomski studij matematičke statistike na istom fakultetu. U zimskom semestru akademske godine 2013./14., prof. dr. sc. Mirko Polonijo dodijelio mi je čast i zadovoljstvo da budem *demonstrator* na kolegiju Euklidski prostori. Od 2013. do 2016. godine, bio sam aktivni član studentske udruge eSTUDENT. Potom sam 2015. godine pohađao praksu u PBZ Croatia Osiguranju, jednom od četiri obavezna mirovinska fonda u Hrvatskoj. Praksu sam proveo u *Istraživačkom odjelu* u trajanju od tri mjeseca. U međuvremenu sam akademsku godinu 2015./16. proveo na Erasmusovoj razmjeni studenata na Sveučilištu u Bielefeldu u Njemačkoj. Nakon toga sam pristupio technology analyst programu međunarodne kompanije Morgan Stanley u New Yorku i Londonu. Navedeni program sam uspješno završio, i trenutno sam zaposlen kao *associate software developer* u Morgan Stanleyevom uredu u Budimpešti.