Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Algorithms for the Mixed Chinese Postman Problem

Final Report for an Interdisciplinary Project by Maximilian P.L. Haslbeck

Examiner:	Prof. Dr. Peter Gritzmann
Advisor:	Wolfgang Ferdinand Riedl, Dr. Michael Ritter
Submission Date:	June 12, 2015

I hereby confirm that this is my own work, and that I used only the cited sources and materials.

München, June 12, 2015

Maximilian P.L. Haslbeck

Abstract

This document reports on a project conducted on the Mixed Chinese Postman Problem (MCPP). The project's goal is to implement an educational game that familiarizes a player with the problem and enables her to try to solve problem instances hands-on. To measure her performance, the resulting custom tour is visualized side by side with an optimal tour. The latter is provided by a solver for the MCPP, which was developed in this project.

Zusammenfassung

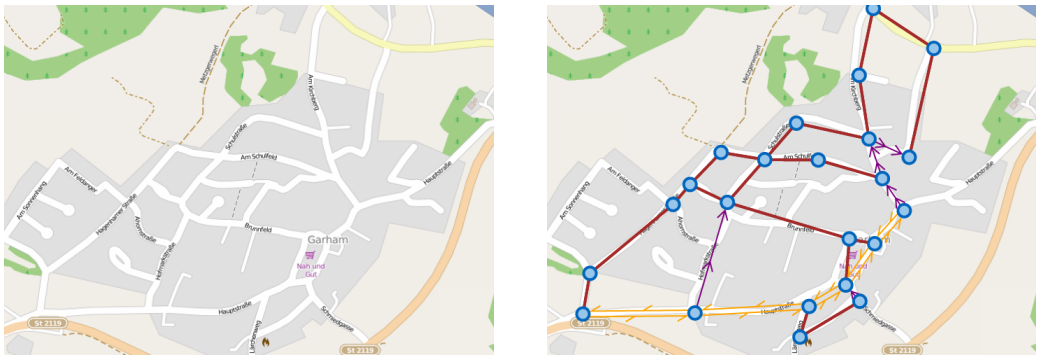
Dieses Dokument berichtet über ein Projekt zum Mixed Chinese Postman Problem (MCPP). Das Ziel des Projekts war die Erstellung eines Lernspiels, welches das Problem einer Spielerin näher bringt und ihr ermöglicht Probleminstanzen spielerisch zu lösen. Um ihren Erfolg zu messen, kann anschließend die eigene Tour mit einer optimale Lösung verglichen werden. Letztere wird von einem MCPP Löser bereitgestellt, welcher im Rahmen dieses Projektes entwickelt wurde.

Contents

Chapter 1

Introduction

Imagine you are working as a postman being responsible for the delivery of mail in your part of a town. You are given a map (e.g. 1.1a) of your town and have to design a tour through the town delivering all your letters, which makes it necessary to visit all streets at least once. As time is money, you want to find the shortest such tour. Note that depending on the town layout some streets will have to be visited more than once.



(a) Map of a small town **(b)** Model of town as a graph

Figure 1.1: Abstraction from map to graph; ©OpenStreetMap

Your town has some peculiarities: there are some one way streets that can only be traversed in one direction and there are streets that are so narrow that it is possible to deliver mail to both sides of the road when passing in either direction. The street layout can be modelled as a mixed graph (see map 1.1b), which contains directed and undirected links.

The problem specified above is known in literature as the Mixed Chinese Postman Problem (*MCPP*). It turns out that the problem is easy to solve if only narrow streets or no narrow streets are used in the town layout. In graph theory terms: for directed and undirected graphs, the problem is in P [EGL95a]. Whereas the mixed case, which we will consider, is even NP-complete [Pap76].

This document reports on a project about the MCPP. In the course of this project a method for solving MCPP instances was implemented and a game was designed that

enables a user to try and design a tour himself and then visualizes an optimal tour side by side with the custom tour.

The remainder of this report is structured as follows: first, necessary mathematical notation will be revised and the MCPP will be defined (Section 2). Then some techniques to solve a MCPP are summarized: in Section 3 we give an overview of the various methods for solving the MCPP. We review a selection of heuristics that where proposed in literature (Section 3.2). In Section 4 two formulations of the MCPP as an Integer Linear Program (*ILP*) are stated and techniques to solve these are presented. In Section 5 details of the implementation of the MCPP solver are explained: this includes a review of the LP solver used and the separation algorithm implemented. Section 6 presents the game that was designed. Finally, a conclusion is drawn and further enhancements are proposed.

Chapter 2

Notation and the Mixed Chinese Postman Problem

In this section the necessary notations will be introduced and the Mixed Chinese Postman Problem will be stated. This introduction is along the lines of [CMS06], but notice that they handle the more general mixed rural postman problem (MGRP). As the notion of mixed graphs is seldomly used, we give a definition here:

Definition 2.1
A **mixed graph** $G = (V, E, A)$ is a triple, consisting of a set of vertices V, a set of (undirected) *edges* E and a set of (directed) *arcs* A. When we talk about the *links* of a mixed graph, we refer to all elements $l \in E \cup A$. To each link l we assign a cost $c_l \in \mathbb{R}$.

In order to define strong connectedness and also the MCPP we need the following notion:

Definition 2.2
Given a mixed graph $G = (V, E, A)$, a **directed path** from v to u is a sequence of links $v = v_1 \xrightarrow{l_1} v_2 \xrightarrow{l_2} \dots \xrightarrow{l_{n-1}} v_n = u$ with either $l_i = (v_i, v_{i+1}) \in A$ or $l_i = \{v_i, v_{i+1}\} \in E$. We call G **strongly connected**, if there exists a directed path from any vertex to any other vertex. A directed path π starting and ending at the same vertex is called a **tour**, with its cost being the sum of the cost of all its links: $c_\pi = \sum_{1 \le i \le n} c_{l_i}$.

Note that a directed path may contain the same link several times. In the following we only consider strongly connected mixed graphs.

In contrast to the setting we consider, in the MGRP of [CMS06] some arcs may not be required, but could be used to short cut (A_{NR}). In the problem considered here, we assume $A_{NR} = \emptyset$. Now we define the Mixed Chinese Postman Problem as follows:

Definition 2.3
Given a strongly connected mixed graph $G = (V, E, A)$, the **MCPP** is the problem of finding a minimum cost tour passing through each link $l \in E \cup A$ at least once.

For a mixed graph $G = (V, E, A)$ and two disjoint sets of vertices $S_1, S_2 \subset V$,

- $(S_1 : S_2) = \{(i,j) \in E \cup A : (i \in S_1 \wedge j \in S_2) \text{ or } (i \in S_2 \wedge j \in S_1)\}$ denotes all links between the sets S_1 and S_2,
- $\delta(S) = (S : V \setminus S)$ is called link cut-set of G defined by S,
- $A^+(S) = \{(i,j) \in A : i \in S \wedge j \notin S\}$, all arcs going out of S,
- $A^-(S) = \{(i,j) \in A : i \notin S \wedge j \in S\}$, all arcs going into S,
- $E(S) = \{\{i,j\} \in E : i \in S \wedge j \notin S\}$, all edges leaving S.

Please note that the input for a MCPP consists of a strongly connected mixed graph G that does not contain multiedges. But tours on G that visit any link at least once might need to pass through a link more than once. To represent and characterize feasible tours in a concise way, we introduce a new mathematical object: an augmented mixed graph $G^{\mathcal{F}} = (V, E^{\mathcal{F}} \cup A^{\mathcal{F}})$, which is obtained by considering each copy of a link in a tour as a different element. This enables us to define properties such as *even*, *balanced* and *symmetric*; and thus characterize augmented mixed graphs which correspond to feasible MCPP tours.

In order to relate an augmented mixed graph $G^{\mathcal{F}}$ to the underlying mixed graph G, we denote with x_l how many copies of the link $l \in E \cup A$ exist in $G^{\mathcal{F}}$. We combine all those x_l into an *incidence vector* $x \in \mathbb{R}^{|E \cup A|}$ and given $T \subset E \cup A$ we write $x(T)$ for $\sum_{e \in T} x_e$. If it is necessary to make an explicit reference to the direction in which a required link is traversed, we will use x_{ij} instead of x_l. One may observe that $G^{\mathcal{F}}$ is specified by the underlying mixed graph G and the vector x, nevertheless it is instructive to have the augmented mixed graph in mind when thinking about finding a tour.

Now we are interested in which augmented mixed graphs $G^{\mathcal{F}}$ give rise to a valid tour in G. First we define some properties of such graphs: For an augmented mixed graph $(V, E^{\mathcal{F}} \cup A^{\mathcal{F}})$, a vertex $v \in V$ is called

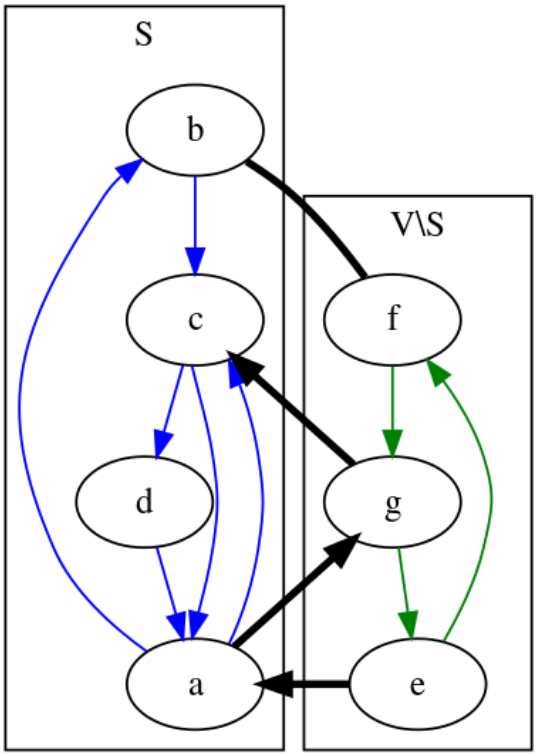
- *even*, if v is incident to an even number of copies of links,
- *odd*, otherwise;

and G is called

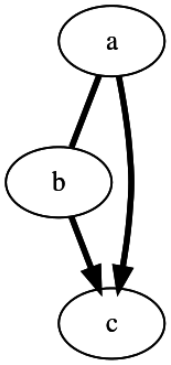
- *even*, if every vertex is even,
- *balanced*, if for every $S \subset V$, the difference between the number of copies of arcs in $A^+(S)$ (leaving S) and the number of copies of arcs in $A^-(S)$ (entering S) is less than or equal to the number of copies of edges in $E(S)$. This condition is called *balanced-set condition* for set S. In symbols:

$$x(A^+(S)) - x(A^-(S)) \leq x(E(S)) \ , \forall S \subset V$$

- *symmetric*, if for every vertex the number of copies of arcs entering it is equal to the number of copies of arcs leaving it.



(a) The subset S is a balanced set. The graph itself is not balanced!



(b) An even graph that is neither symmetric nor balanced and clearly has no Euler tour.

Figure 2.1: Examples for the graph properties.

The balancedness property might not be obvious: if the number of incoming and outgoing arcs is not equal there must be a sufficient number of edges to balance, in order to permit a valid tour; this is captured by the balancedness property. For example, in figure 2.1a the subset S is balanced, but the graph is not. Note that any symmetric graph is balanced. Figure 2.1b shows an example of an even (augmented) mixed graph.

In contrast to the necessary and sufficient conditions for directed and undirected graphs having an Euler tour, it is not that obvious for the mixed case. To see the argmuent more clearly, we will sketch the proof by Ford and Fulkerson [FF62]. Note that they talk about mixed graphs and not augmented graphs, but – as augmented graphs could be transformed into normal mixed graphs by splitting multiple links with the help of some extra vertices – the result carries over.

Theorem 2.4
The mixed graph $G = (V, E, A)$ has an Euler tour iff the following three conditions hold
(a) G is connected;
(b) G is even, i.e. every vertex of G is even;
(c) G is balanced, i.e. for every $X \subset V$, the difference between the number of incoming and outgoing arcs is less than or equal to the number of incident edges.

Proof. Note that this theorem subsumes the directed and undirected cases. The necessity is easy to see. To prove sufficiency one can proceed by establishing a circulation (i.e. a flow without source and sink, but lower and upper bounds on the flow of every link, c.f. [FF62]) in G, then directing some of the originally undirected arcs according to this circulation in order to construct the Euler tour.

Showing existence of such a circulation requires some results about circulations, which will not be presented here (but can be found in [FF62]). The result, after directing the undirected arcs according to the circulation, is a mixed graph G_2 which satisfies (a), (b) of the theorem and also (c'): *for every vertex the number of incoming directed arcs equals the number of outgoing directed arcs.*

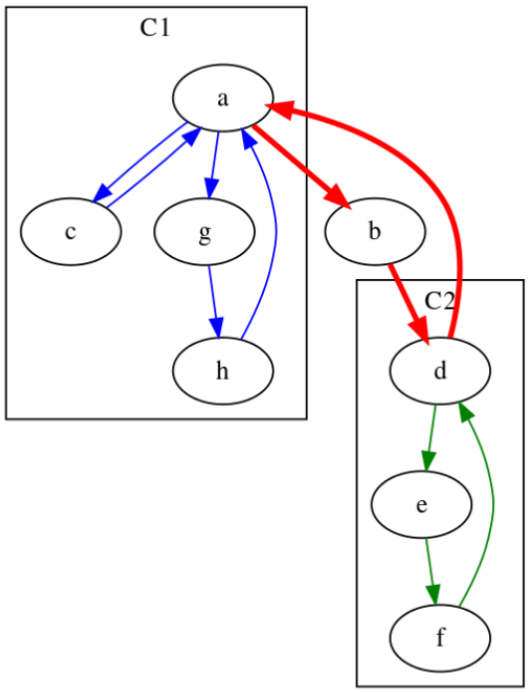


Figure 2.2: Visualization of the induction step: when deleting the red tour, two connected components remain, which by induction hypothesis possess Euler tours themselves

With the properties of a circulaton, it now suffices to show that there exists an Euler

tour in G_2. This can be done by induction on the number of arcs of such a graph. If G_2 has no arcs and is connected, it consists of a single vertex and thus contains an Euler tour of length 0. Assume that a graph with fewer than $m > 0$ arcs fulfilling the properties (a), (b) and (c') has an Euler tour. Then consider such a graph G_2 with exactly m arcs. By (b) and (c') the graph contains a closed tour that visits more than one vertex. Delete the arcs of this tour and obtain a graph G_2' satisfying (b) and (c') (see Figure 2.2; now consider the connected components of G_2': they fullfill (a), (b) and (c') and have less than m arcs, hence possess an Euler tour by the inductive assumption. It follows that G_2 has an Euler tour. □

Applying the necessary and sufficient conditions for a connected mixed graph to be Eulerian, we can state that $\mathcal{F}$ is a tour for the MCPP when the following conditions are satisfied:

- $\mathcal{F}$ contains all the required links,
- graph $(V, E^{\mathcal{F}} \cup A^{\mathcal{F}})$ is connected,
- graph $(V, E^{\mathcal{F}} \cup A^{\mathcal{F}})$ is even
- graph $(V, E^{\mathcal{F}} \cup A^{\mathcal{F}})$ is balanced

Section 3 summarizes literature about the MCPP and presents some established heuristics.

Chapter 3

Solving the MCPP

In the following we first give an overview of theoretical results and the various methods that were proposed in literature in order to solve the MCPP. We go into more detail about heuristics in section 3.2 and present a method for solving the MCPP exactly with Integer Linear Programming in section 4.

3.1 Survey of methods

A lot of research has been conducted related to the Chinese Postman Problem and a variety of derived problems were examined (e.g. the Mixed Chinese Postman Problem, the Rural Postman Problem, the Windy Postman Problem, the Hierarchical Postman Problem, etc.). Several surveys condensing this research exist ([EGL95a], [EGL95b], [CP10]).

As already indicated above, the MCPP was shown to be NP-complete by Papadimitriou in 1976 [Pap76] by a reduction to 3SAT. Furthermore this result was even extended to special cases, such as planar graphs and uniform edge costs. Other special cases, e.g. the problem on even graphs, can be solved in polynomial time ([CP10, Section 2.1.1], [EJ73]). The latter observation is the basis of heuristics due to Frederikson [Fre79] and its enhancements, which will be explained in section 3.2.

Apart from polynomial algorithms for special cases and heuristics for the general case – to the knowledge of the author – there exists only one further method for solving the MCPP: Integer Linear Programming. In literature two formulations F1 and F2 are described. In section 4 these will be presented and compared. Although the polyhedron associated with the MCPP was not studied directly, results for a generalization [CRS03] can be applied: for example the odd-cut inequalities and balanced-set inequalities – which will be introduced in section 4 – are face-inducing and new families of valid inequalities have been found (e.g. zig-zag inequalities [CPS06]). With this method instances of upto 3000 nodes and 9000 arcs could be solved to optimality [CP10, Section 2.1.1].

3.2 Heuristics

As the MCPP is NP-complete [Pap76] it is very unlikely that efficient algorithms for its exact solution exist, therefore it is natural to seek for approximation algorithms. Pearn and Chou describe several recent heuristics in their paper [PC99]: There are two families of heuristics (called Mixed-1 and Mixed-2) originally due to Edmonds and Johnson [EJ73] and Frederickson [Fre79], respectively. Both heuristics were *modified* first by Pearn and Liu [PL95], such that artificial cycles were removed from the solution. Then they were further *improved* by Pearn and Chou [PC99], such that the best solution of several graphs with some edge directed is chosen. For sake of simplicity only the two original algorithms are stated here:

- Mixed-1 first converts the given network into an even network by unorienting the arcs and solves it by standard flow methods. Then the solution is transformed into a symmetric network which may not be even anymore. Finally, the symmetric network will be recovered back to even.
- Mixed-2 first converts the given network into a symmetric network by adding some new arcs and solves the minimal-cost flow problem on the converted flow network. On this symmetric network CPP on undirected graphs can be applied in order to obtain an Euler tour.

The two heuristics apply the same two steps in opposite order. Frederikson showed that both algorithms lead to a worst case ratio of 2. Applying both heuristics and then choosing the best has a worst case ratio of $\frac{5}{3}$. This idea of combining the two algorithms can also be applied to the modified and improved Mixed heuristics.

Computational experiments in [PC99] show that the heuristics of each family perform better in historical order; but neither of the families outperforms the other.

Chapter 4

Formulating the MCPP as an ILP

This section summarizes the literature on tackling the MCPP via integer linear programming.

Formulations of the MCPP as an ILP usually take two different forms: Their main difference is that the first formulation F1 has one variable assigned for each edge, whereas the second formulation F2 assignes a variable for each orientation of an edge. First we present these two versions (according to [CMS06]) and in a second step we get into detail how instances of these ILP can be solved and finally summarize the comparison of the two.

4.1 Two formulations

4.1.1 F1

The first formulation denoted by $F1$ follows the idea of assigning one variable to each edge. In order to construct a tour from a solution for an instance of $F1$, one has to orientate the edges afterwards. But this can be done in polynomial time [EJ73].

So the first formulation follows by stating the characterization of a MCPP tour, as defined in section 2, in terms of linear programming:

Given a strongly connected graph $G = (V, E, A)$ we already introduced an incidence vector (for F1 we use y) denoting the multiplicity of every link in the MCPP tour. With that in mind, the set of valid MCPP tours is equivalent to the set of vectors $y = (y_e : e \in E \cup A) \in \mathbb{R}^{|E \cup A|}$ satisfying:

$$y_{ij} \geq 1 \text{ and integer}, \qquad \forall (i,j) \in A \tag{4.1}$$

$$y_e \geq 1 \text{ and integer}, \qquad \forall e \in E \tag{4.2}$$

$$y(\delta(\{i\}) \equiv 0 \mod 2, \qquad \forall i \in V : i \text{ is even} \tag{4.3}$$

$$y(\delta(\{i\}) \equiv 0 \mod 2, \qquad \forall i \in V : i \text{ is odd} \tag{4.4}$$

$$y(A^+(S)) - y(A^-(S)) \leq y(E(S)), \qquad \forall S \subset V \tag{4.5}$$

The characteristic properties can be checked easily: (4.1) and (4.2) imply that the required links are in the solution, (4.3) and (4.4) assure that the resulting graph is even and (4.5) that the balanced-set conditions will be satisfied.

Note that (4.3) and (4.4) can be stated linearly by $y(\delta(\{i\})) - 2z_i = 0$ with additional variables z_i.

4.1.2 F2

As the symmetry property of a mixed graph implies the balanced property, we can obtain another characterization of a MCPP tour (all required links, connected, symmetric). To state the symmetry property, all edges have to be orientated. Thus the second formulation $F2$ assignes two variables x_{ij} and x_{ji} to each edge e denoting how many times the edge is traversed in the respective orientation.

Lets take a minute to think about whether we loose some feasible solutions here: it is easy to see that every symmetric and connected augmented mixed graph is also balanced and even. Conversely, given a solution to F1: it induces a connected, even and balanced augmented mixed graph. "Edmonds and Johnson proposed a method of assigning a direction to the edges of an even and balanced mixed graph in order to obtain, with the same cost, a symmetric directed graph. Thus, from any feasible solution to F1 we can obtain a feasible solution to F2 with the same cost" [CMS06].

Lets state the ILP formulation F2:

$$x_{ij} \geq 1 \text{ and integer}, \qquad \forall (i,j) \in A \tag{4.6}$$

$$x_{ij}, x_{ji} \geq 0 \text{ and integer}, \qquad \forall e = \{i,j\} \in E \tag{4.7}$$

$$x_{ij} + x_{ji} \geq 1, \qquad \forall e = \{i,j\} \in E \tag{4.8}$$

$$x(A^+(i)) + \sum_{j \in V} x_{ij} = x(A^-(i)) + \sum_{j \in V} x_{ji}, \qquad \forall i \in V \tag{4.9}$$

It is again easy to see that the solutions of this ILP correspond to the MCPP tours on G: (4.6) and (4.8) assure that every required link is in the solution and (4.9) implies that the symmetry condition is satisfied. Together with integrality, the symmetry condition (4.9) implies that the resulting graph is even.

The two formulations can be shown to be equivalent in the sense that they give rise to the same set of MCPP tours. [CMS06] give a detailed proof, spelling out the proof sketched above.

As the ILP can not be solved directly, but there exists a vast number of methods to solve Linear Programs, we now are interested in the corresponding LP relaxations of the two formulations.

4.2 LP relaxations

As we plan to solve the ILPs with linear programming we have to consider the corresponding LP relaxations of the two formulations.

Which conditions have to be removed? Surely the integrality constraints in $F1$ and $F2$, and consequently constraints (4.3) and (4.4) in $F1$ are obsolete, as they cannot be stated linearly without integrality.

The problem is that the evenness conditions in $F1$ cannot be stated linearly without integrality. Also for $F2$, evenness is only assured via the symmetry condition plus integrality of the solution. However it is possible to express some weaker condition than evenness linearly:

For every odd vertex i, it must hold $y(\delta(\{i\})) \geq |\delta(\{i\})| + 1$, as the vertex has to be of even degree in the solution. This can be lifted to odd cut sets in order to obtain the *odd cut inequalities*:

$$y(\delta(S)) \geq |\delta(S)| + 1, \forall S \text{ with } |\delta(S)| \text{ odd}$$

Every feasible solution to F1 and F2 has to fulfill these inequalities.

As we can see, there will be exponentially many balanced-set inequalities 4.5, one for each subset S of V. If we consider such S with $E(S) = \emptyset$ the corresponding balanced-set conditions for S and $\bar{S}$ imply the so called *system equation* $y(A^+(S)) = y(A^-(S))$. Considering the set of such equations for all S with $E(S) = \emptyset$ one can see that most of these equations will be linearly dependent. But we can efficently find a linearly independent subset of those: We look at the connected components $K_1, \dots K_q$ of the Graph $G(V, E)$ only containing edges. Each of these K_i gives rise to a system equation. It can be shown that any $q - 1$ of those are linearly independent [CMS06]. Now we call the set of equations

$$y(A^+(K_i)) = y(A^-(K_i)), i = 1, ..., q \tag{4.10}$$

the *system equations*.

Now we consider the LP relaxation LP_{F1} of the first formulation, as we have seen the integrality constraints have to be removed, thus the inequalities enforcing evenness get obsolete and must be replaced by the odd-cut inequalities 4.14.

$$y_{ij} \geq 1, \qquad \forall (i,j) \in A \tag{4.11}$$
$$y_e \geq 1, \qquad \forall e \in E \tag{4.12}$$
$$y(A^+(S)) - y(A^-(S)) \leq y(E(S)), \qquad \forall S \subset V \tag{4.13}$$
$$y(\delta(S)) \geq |\delta(S)| + 1, \qquad \forall S \text{ with } |\delta(S)| \text{ odd} \tag{4.14}$$

For the LP relaxation LP_{F2} of F2, also integrality constraints have to be removed. In principle once an integral solution is found, the symmetry condition 4.18 implies evenness of the graph. As the odd-cut inequalities 4.19 have to be satisfied by any solution to F2, they can be added to the LP relaxation.

$$x_{ij} \geq 1, \qquad \forall (i,j) \in A \tag{4.15}$$

$$x_{ij}, x_{ji} \geq 0, \qquad \forall e = \{i,j\} \in E \tag{4.16}$$

$$x_{ij} + x_{ji} \geq 1, \qquad \forall e = \{i,j\} \in E \tag{4.17}$$

$$x(A^+(i)) + \sum_{j \in V} x_{ij} = x(A^-(i)) + \sum_{j \in V} x_{ji}, \qquad \forall i \in V \tag{4.18}$$

$$x(\delta(S)) \geq |\delta(S)| + 1, \qquad \forall S \text{ with } |\delta(S)| \text{ odd} \tag{4.19}$$

Note that there are exponentially many odd cut inequalities (4.14) and (4.19). Also, in $F1$ the balanced set inequalities (4.13) have exponentially many instances!

Thus only a subset of those can be explicitly added to the LPs to be solved. Corberán et al. [CMS06] define the following initial LP relaxations:

$LP0_{F1}$ containing:

- system equations (4.10)
- constraints (4.11) and (4.12)
- one balanced-set inequality (4.13) for each unbalanced vertex
- one odd cut inequality (4.14) for each odd balanced vertex

where an unbalanced vertex i is a vertex for which $|E(i)| < |A^+(i)| - |A^-(i)|$ or $|E(i)| < |A^-(i)| - |A^+(i)|$.

$LP0_{F2}$ containing:

- constraints (4.15) and (4.17)
- one symmetry inequality (4.18) for each vertex
- one odd cut inequality (4.19) for each odd balanced vertex

4.3 Solving the ILP

Now we have formulated the problem as an ILP and stated the corresponding LP relaxation. For solving the ILP, one strategy is to use a *cutting-plane algorithm*. First a solution x^* for an initial LP, containing only a small subset of all the valid inequalities, is computed. The algorithm then looks for valid inequalities, that are violated by x^*. Those violated inequalities are added to the LP and a new solution is determined with LP techniques. This process is iterated as long as new violated inequalities are found.

As a subroutine this algorithm has to solve the *separation problem*: violated inequalities have to be identified efficiently. [CMS06] states that for each of the inequalities in $F1$ and $F2$ this can be done in polynomial time. Of special interest is the separation problem associated with the odd cut inequalities, which will be addressed in the following subsection.

Once the cutting plane algorithm terminates, a solution x^* for the LP relaxation of the ILP formulations is found. If this is integral, a solution for the MCPP is found, if not, Corberán et al. [CMS06] propose to use a branch-and-bound algorithm to obtain an integral solution. If the integer solution then is feasible it is optimal, if not violated inequalities can be identified with the help of the separation algorithms and a new iteration can be started.

4.3.1 Separation problem for odd cut inequalities

Violated odd cut inequalities can be found in polynomial time using the Padberg and Rao algorithm [PR82]. In this presentation, we assume we are solving the ILP $F2$ with a cutting plane algorithm. Thus we obtained a solution x^* to some LP containing at least the constraints in $LP0_{F2}$, i.e. it holds $x_{ij} \geq 1$ and $x_{ij} + x_{ji} \geq 1$ for arcs and edges, respectively.

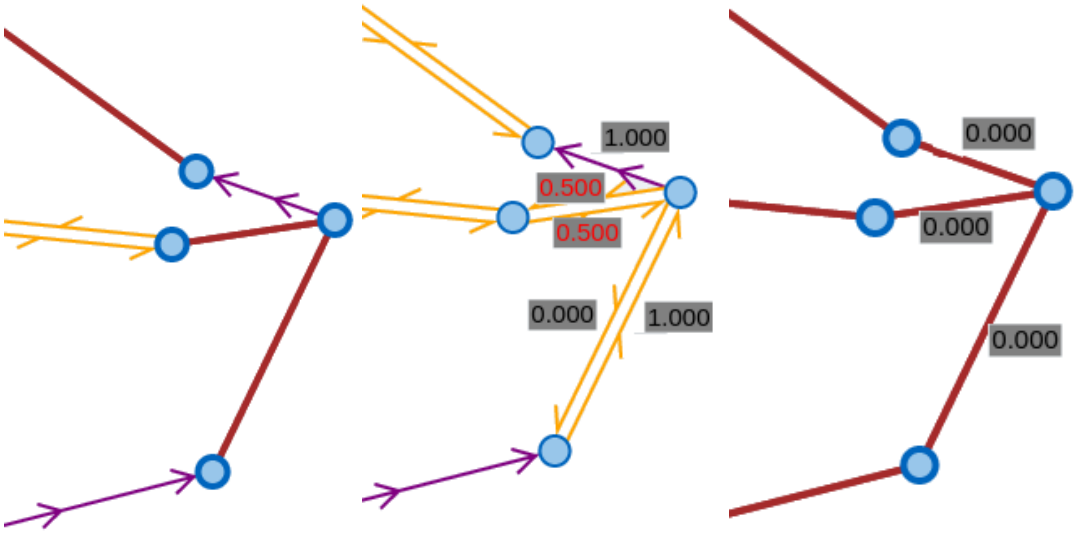
Now we look for an odd cut S that violates the odd cut inequality, which can be restated as:

$$\sum_{e \in \delta(S)} x_e^* - 1 = x^*(\delta(S)) - |\delta(S)| \geq 1 \tag{4.20}$$

The separation problem can be formally defined as:

Definition 4.1
Let $G = (V, E, A)$ be a strongly connected graph, $LP0_{F2}$ the corresponding initial relaxation, and x^* a solution in the solving process satisfying the constraints in $LP0_{F2}$, then the problem of finding a set $S \subset V$ that violates equation 4.20 is called the **separation problem for odd cut inequalities**.

This can be done by looking for a minimum weighted odd cut in a special Graph G^+. This graph is simply obtained by undirecting the arcs and adding them to the edge set, formally:



(a) Part of a mixed graph G **(b)** with a fractional solution x^* **(c)** and its corresponding graph G^+

Figure 4.1

Definition 4.2
Given a strongly connected mixed graph $G = (V, E, A)$ and a partial solution x^*, we define the undirected weighted graph $G^+ = (V, E^+)$ with $E^+ = E \cup \{\{v, u\} | (v, u) \in A\}$ being the set of links unoriented. We define the weights for former edges $\{i, j\} \in E$ by $w_{ij} = w_{ji} = x^*_{ij} + x^*_{ji} - 1$ and for former arcs $(i, j) \in A$ as $w_{ij} = w_{ji} = x^*_{ij} + x^*_{ji} - |\{(i, j), (j, i)\} \cap A|$.

In figure 4.1a a part of a mixed graph G is displayed, a fractional solution can be seen in figure 4.1b and the corresponding graph G^+ is illustrated in figure 4.1c.

The Padberg-Rao algorithm [PR82] now proceeds in constructing a cut-tree of G^+ via the standard Gomory Hu algorithm and then looks for the minimum odd cut represented by one of the $V - 1$ edges of this cut-tree. One can see that there is an odd cut set violating the odd cut inequality if and only if the cut value obtained by the algorithm is smaller than 1. Note that the Padberg-Rao algorithm runs in $\mathcal{O}(|V|^2|E^+| \log(|V|^2/|E^+|))$.

Here, an optimization is possible. If more than one odd cut set is found that violates the odd cut inequality, a subset of those can be added to the LP [CPS05, Section 4.2].

4.3.2 Improving the bounds

The process of finding a solution could be accelerated by considering new families of inequalities. In Section 3.3 of [CMS06], some families are described for the MGRP. It is not clear to the author whether these could also be applied to the MCPP, because further on Corberán et al. state that "the additional valid inequalities [...] do not apply when all the links are required" [CMS06, p. 3398]. This could be further examined.

A new family of inequalities that could be used for the MCPP are the zigzag inequalities, presented in [CPS06].

4.4 Comparison

As already stated above, the formulations $F1$ and $F2$ are equivalent in the integer case. Corberán et al. investigated on the difference of the two formulations in the general case of MGRP. They come to the conclusion that although the initial LP $LP0_{F2}$ dominates $LP0_{F1}$, the bounds found by the cutting-plane algorithm are equivalent for F1 and F2 (the former can be observed from computational experiments, whereas the latter is proven) [CMS06] . Even additional families of inequalities do not have an effect here.

Nevertheless, differences occur with respect to the run time of the algorithms providing the bounds. As the algorithm based on $F1$ has only one variable per edge, the resulting LPs have less variables and thus one could expect shorter runtime. However, the algorithm based on $F2$ only has to identify violated inequalities of one family. In conclusion there is no common trend with respect to the running time needed; however the algorithm based on $F2$ appears to be easier to code [CMS06].

Chapter 5

Implementation of the backend

In the previous sections, necessary notions for the MCPP were revised and the problem statement defined; two standard formulations of the MCPP as an ILP were shown and an algorithm to solve them exactly was sketched. Additionally, several different heuristics for the MCPP were mentioned.

Based on the reviewed literature, we carried on planning a MCPP solver. An exact solver was desired, because running time is not crucial for the small problem instances considered, but one should be able to learn from the method used. For the implementation three major design choices had to be taken:

ILP formulation
: As we have seen, the MCPP can be solved with means of Integer Linear Programming. There are two possible formulations of the MCPP as an ILP. While none of the two outperforms the other, the algorithm based on $F2$ is easier to implement, because only the separation algorithm for odd cut inequalities has to be provided. Thus we use formulation $F2$.

LP Solver
: In order to execute the cutting plane algorithm we need to solve linear programs. Therefore we employ Gurobi [Gur15], a state-of-the-art mathematical programming solver which can be used freely in academia and comes with a well documented API for Java. Some notes on Gurobi will be stated in Section 5.1.

Graph library
: As the separation algorithm is based on graph algorithms we do not reinvent the wheel but reuse the well designed open source library JGraphT[1]. How JGraphT is used to model mixed graphs and implement the necessary algorithms to perfom the separation is described in Section 5.2.3.

The central part of the implementation is the cutting plane algorithm, which was already outlined in the discussion in Section 3.3, and will be specified further in Section 5.1. In Section 5.2 the separation algorithm is presented once more and details

[1] http://www.jgrapht.org

on the implementation of necessary graph algorithms are provided. Generation of random problem instances is presented in Section 5.3. Finally we refer to where these components can be found in the Java project (Section 5.4).

5.1 Cutting plane Algorithm and Gurobi

As we have seen in Section 3.3, solving an instance of MCPP as an ILP comes in two stages:

First the problem is formulated as the LP relaxation $LP0_{F2}$ which is then passed to the LP solver Gurobi in order to solve it.

While solving the instance, Gurobi finds some partial solutions that may be fractional. The LP solution at that stage may not be feasible, because it violates some odd cut inequalities. To prohibit the found solution, one of the violated odd cut inequalities has to be identified. The separation algorithm provides such an inequality and adds it to the instance. Based on this enhanced LP instance Gurobi carries on solving the problem. This is iterated as long as no further violated odd cut inequalities are found. This means the found LP solution is feasible. If this solution still is fractional, Gurobi goes on looking for an integer solution employing branch and bound.

This mechanism is standard in linear programming and is also implemented in Gurobi. First given an mixed graph G, the LP relaxation $LP0_{F2}$ is generated and the optimization is started. Via a Callback in Gurobi's stages *MIPNode* and *MIPSol* the separation algorithm is used to find violated odd cut inequalities, which are added as lazy cuts to the Gurobi model. As we only allow strongly connected graphs, an optimal solution always exist; when Gurobi terminates with an integer solution this is a feasible solution for $F2$.

Gurobi can be obtained under free license for academic use and provides a well documented API for Java [Gur15]. The installation of Gorubi and usage with Java turned out to be quite smooth.

One important ingredient for the cutting plane algorithm is the separation algorithm. This was the heart of the algorithmic challenge of this project and is described in the following section:

5.2 Separation algorithm and JGraphT

In section 4.3.1 we already described the mathematical basis with which the separation problem can be solved. In this section we will describe in detail the way we implemented this.

First we give a detailed description of the Padberg Rao algorithm, which includes computing a cut-tree for the underlying graph. We implemented the Gormory Hu algorithm to solve that problem, which is in turn based on $|V| - 1$ minimal cut

computations. As this is a standard network algorithm we used the JGraphT library for that purpose.

5.2.1 Padberg Rao

Recall that search for an odd cut S satisfying equation 4.20, can be reduced to finding a minimum odd cut in the derived graph G^+. In the following, we describe the Padberg Rao algorithm, which finds a minimum odd cut in a given undirected graph.

Algorithm 1 Padberg Rao [PR82]

Input: Graph $G^+ = (V, E^+)$ and weights $c \in \mathbb{Q}_+^{E^+}$.
Output: a minimum odd cut.

1: Compute a cut-tree of the graph G^+ with weights c. ▷ with Gomory Hu
2: **for** each of the $n-1$ edges of the cut-tree **do**
3: Let $\delta(U)$ denote the cut induced by the cut-tree edge.
4: Compute the parity $|U| \mod 2$ and the weight $c(U)$ of the cut.
5: If adequate, store U.
6: **return** the best cut U.

To solve this problem, first a cut-tree of the graph G^+ is determined. A cut-tree T is a compact representation of any cut in G^+. For a formal definiton see [LRT08]. Given two vertices u and v, the minimum cut value λ_{uv} between u and v is the minimum edge weight on the path from u to v in T. Note that thus one property of a cut-tree is, that any possible minimum cut value is represented by an edge in T. Consequently, a minimum odd cut can be found by going through any of the $n-1$ edges of T and testing whether the cut it induces is odd.

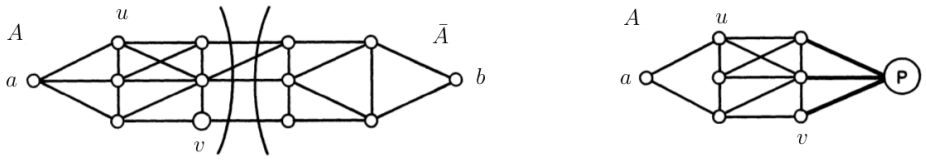
It is easy to see that if a odd cut with weight smaller than 1 is found, we have a violated odd cut inequality (c.f. equation 4.20).

5.2.2 Gomory Hu

Gomory and Hu first mentioned the term cut-tree. They showed that for every network there exists a cut-tree. As this tree only contains $n-1$ edges there are only $n-1$ numerically different cuts possible, which made them suspect that all $n(n-1)/2$ cuts can be determined with only $n-1$ minimal cut computations.

To further optimize such an algorithm, first consider a network G with a minimal cut $(A, \bar{A})$ separating vertices $a \in A$ and $b \in \bar{A}$. We are now interested in a minimal cut separating two other vertices u and v (see figure 5.1a. We could just calculate the minimal cut in G, but we can do better: If we contract all vertices in $\bar{A}$ into a super node P and collapse all edges going from some vertex in A to $\bar{A}$ we obtain the condensed graph G' (figure 5.1a). Note that G' is smaller or equal to G. The following

lemma makes sure that calculating the minimal cut on G' obtains the same result as the calculation on G:



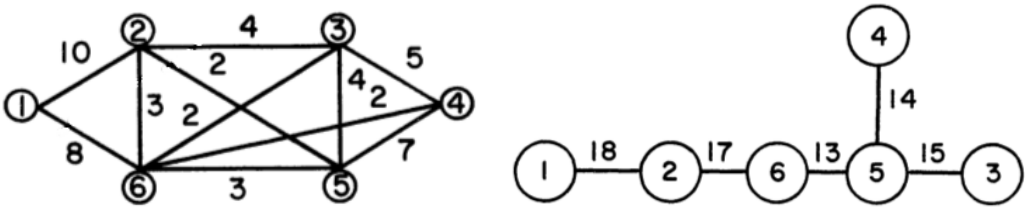
(a) A graph G **(b)** and its condensed graph G'.

Figure 5.1

Lemma 5.1
Let $G = (V, E, c)$ be a network and $(A, \bar{A})$ a minimal cut, then the cut between two vertices $u, v \in A$ in the condensed network G' w.r.t. A is numerically equal to the cut λ_{uv} in the original network G.

The proof can be found in the paper by Gomory and Hu (Lemma 1 in [GH61]). This lemma enables us to speed up the minimal cut computations, as the condensed graph G' is smaller. It would also be possible to calculate the remaining cuts in A and $\bar{A}$ in parallel.

The Gomory Hu algorithm (2) repeatedly chooses a set of vertices X (line 2) that is not already partitioned, then chooses two vertices s, t from it and calculates the minimum cut between them (line 5). As we have seen, this can be speeded up by condensing G on basis of the cuts already known. This is done by algorithm 3. After that, this cut will be added to the data structure storing the known cuts (a "partial" cut-tree): first the old partition X is deleted and the new partitions $A \cap X$ and $B \cap X$ are added (line 6), the tree edges are redirected (lines 7-9) and the new tree edge between $A \cap X$ and $B \cap X$ is inserted (line 10). The process ends when no set of vertices has more than one vertex, thus after $n - 1$ steps.



(a) A network G **(b)** and one of its cut-trees T.

Figure 5.2: The resulting cut-tree T after the Gomory Hu algorithm [GH61].

Algorithm 2 Gomory Hu [GH61]

Input: A weighted undirected graph $G = ((V_G, E_G), c)$.
Output: A cut-tree $T = (V_T, E_T)$.

1: Set $V_T = \{V_G\}$ and $E_T = \emptyset$.
2: Choose some $X \in V_T$ with $|X| \geq 2$ if such X exists. Otherwise, go to step 11.
3: For each connected component $C = (V_C, E_C)$ in $T \setminus X$, let $S_C = \bigcup_{v_T \in V_C} v_T$. Let $S = \{S_C | C \text{ is a connected component in } T \setminus X\}$.
4: Condense the graph to form G' ▷ with ContractComponents
5: Choose two vertices $s, t \in X$ and find a minimum s-t cut (A', B') in G'.
Set $A = (\bigcup_{S_C \in A' \cap S} S_C) \cup (A' \cap X)$ and $B = (\bigcup_{S_C \in B' \cap S} S_C) \cup (B' \cap X)$.
6: Set $V_T = (V_T \setminus X) \cup \{A \cap X, B \cap X\}$.
7: **for** each $e = (X, Y) \in E_T$ **do**
8: If $Y \subset A$, set $e' = (A \cap X, Y)$, else set $e' = (B \cap X, Y)$.
9: Set $E_T = (E_T \setminus \{e\}) \cup \{e'\}$ and $w(e') = w(e)$.
10: Set $E_T = E_T \cup \{(A \cap X, B \cap X)\}$.
Set $w((A \cap X, B \cap X)) = c'(A', B')$.
Go to step 2.
11: Replace each $\{v\} \in V_T$ by v and each $(\{u\}, \{v\}) \in E_T$ by (u, v).
12: **return** cut-tree T.

As we do not only know about one cut, as in the example above, but several, we can condense several partitions to super nodes. We consider the partial cut-tree T which contains all the known cuts. This is a tree with nodes being sets of vertices of the original graph G. One of those nodes is X. Now deleting X from T results in a possibly disconnected graph, where every connected component S_C can be collapsed into a super node. Essentially, this is done by Algorithm 3.

5.2.3 Minimal cut and JGraphT

The Gomory Hu algorithm relies on the calculation of minimal cuts, thus an algorithm for that problem is necessary. As this is a classical graph algorithm we could use the Java graph library JGraphT for that purpose. It comes with data structures for different sorts of graphs, including simple undirected graphs and weighted directed graphs, and provides a minimal cut algorithm for undirected graphs.

As the *Mixed* CPP is considered, a data structure for mixed graphs is needed. JGraphT provides also functions to test directed graphs for strong connectivity. Before starting the calculation of a solution it has to be made sure that the underlying graph is strongly connected, thus it is beneficial to reuse this method. Thus the mixed graph was designed as a directed graph, with every link having a flag whether it is an edge or an arc. This enables us to use several algorithms provided by the library and

Algorithm 3 ContractComponents

Input: A weighted undirected graph $G = ((V_G, E_G), c)$, its partial cut-tree T, X, u and v.

Output: condensed graph $G' = ((V'_G, E'_G), c')$

1: For each connected component $C = (V_C, E_C)$ in $T \setminus X$. Let $S_C = \bigcup_{v_T \in V_C} v_T$. Let $S = \{S_C | C \text{ is a connected component in } T \setminus X\}$.

2: $V'_G = X \cup S$.
$E'_G = E_G|_{X \times X} \cup \{(u, S_C) \in X \times S | (u, v) \in E_G \text{ for some } v \in S_C\}$.
$c' : V_{G'} \times V_{G'} \to R^+$ is the capacity function defined as,

$$\begin{cases} c'(u, S_C) = \sum_{v \in S_C : (u,v) \in E_G} c(u, v) & \text{if } (u, S_C) \in E_G|_{X \times S} \\ c'(u, v) = c(u, v) & \text{otherwise} \end{cases}$$

representing the graph in a compact way.

JGraphT is freely available and dual-licensed under LGPL[2] and EPL[3]. It provides data structures for directed and undirected graphs and a variety of efficient graph algorithms.

5.3 Random Planar Graphs

To provide the frontend with problem instances that can be presented nicely and resemble the structure typical for the street layout of a town we generate random planar graphs. This process can be controlled with four parameters:

n the number of nodes in the town

p the probability that a link connects two nodes

q the probability that a link is an edge

seed a random seed

The process of generating random planar graphs works as follows: iteratively n points $x, y \in [0, 999]$ are drawn uniformly at random and added to the vertex set if the distance to any other point is bigger than a threshold. This should prevent junctions from being to close and thus displayed overlain. After that, for every combination of points P, Q first condition 1 is tested: if a link from P to Q crosses any other link already in the graph do not add the link. If it can be added without generating a

[2] http://www.gnu.org/licenses/lgpl-2.1.html
[3] http://www.eclipse.org/org/documents/epl-v10.php

crossing do so with probability p; add it as an edge with probability q and as an arc otherwise. If, after that process, the graph is not strongly connected, the strongly connected components $C_1, \ldots, C_c$ are calculated and are connected circularly.

Note that the experiments, which will be described later, were executed for random graphs. These were generated without embedding in the plane, thus condition 1 was dropped and links were added only depending on the coin flips with probability p and q.

5.4 Overview of the Classes in the Implementation

In this section, we describe in which files the implementation can be found[4]. The entry points for MCPP.jar and – what we will later see – AvailableSettings.jar are located in the package **main**. Here, parameter and I/O are handled. The package **maps** provides a class *map* which is constituted by *City* (i.e. junctions) and *Link* objects. Package **util** contains some util datastructures for handling maps and game instances. **metrics** specifies the interface *Metrics* and implements three instances. The main logic for solving MCPP instances can be found in the package **mcppSolver**:

MixedWeightedGraph

The class **MixedWeightedGraph** is the datastructure to represent mixed graphs. It wraps a weighted directed graph *delegate* and stores for every link a flag, denoting whether it is an edge or an arc, and the corresponding variable of the Gurobi model. It also provides several important methods, including the calculation of an Euler tour from a Gurobi solution, the graph G^+ for the separation algorithm based on a partial solution of Gurobi, methods for initializing the LP relaxation and exporting the graph in different formats.

GraphGenerator

This class implements the ideas of Section 5.5: it offers methods to generate random (planar) graphs provided the parameters n, p, q and *seed*.

GraphAlgorithms

In this file, the main algorithms Padberg Rao and Gomory Hu are implemented.

Callback

The class **Callback** implements the GRBCallback interface and specifies the behaviour for the callbacks MIPNode and MIPSol. For both the separation algorithm will be triggered: this involves calculating G^+ of *MixedWeightedGraph* based on the current LP solution and then executing the Padberg Rao algorithm of *GraphAlgorithms*.

[4]The sources can be found on the attached CD in folder /src

CuttingPlane

The method *solveWithCallback* is defined here: given a *MixedWeightedGraph* G, it calculates the optimal MCPP tour. To this end, first the LP relaxation is initialized, a Callback object is created and registered with Gurobi and then the Gurobi optimization is started.

5.5 Experiments

Several experiments were planned in order to measure the running time of the solving process. The idea was to find out how the running time of the algorithm grows with the size of the input graph. This was complicated by the fact that running time of the solving process varies even for the same problem instance. It turned out that the execution time grows for subsequent Gurobi calls. This strange effect could not be explained fully by the Garbage Collection of Java, neither could the problem be located in the separation algorithm. Timing grows in the Gurobi part. Nevertheless a conversation with the Gurobi support could not solve that problem.

Fortunately, this effect doesn't occur when restarting the Java application. Thus, measuring time of the execution of the complete JAR-File results in meaningful data. Also it has to be noted that in the educational game, we designed, for every solving process the JAR-File is executed independently. Hence the experiments talk about the actual running time of the solving process.

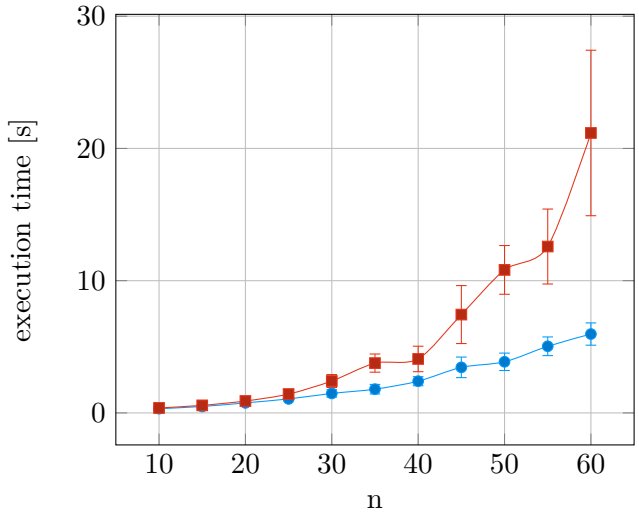
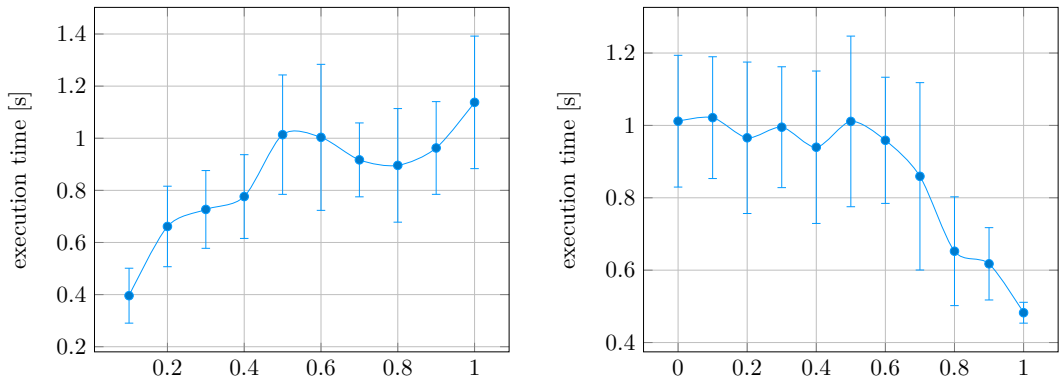


Figure 5.3: Running times for solving MCPP instances with $p = 0.8$, $q = 0.5$ and Euclidean Cost (red)/Uniform Cost (blue)

Figure 5.3 shows that the running time for solving a MCPP grows with the number of

vertices n. This growth seems to be exponential. Furthermore instances with uniform cost can be solved faster than instances with euclidean distance as link costs. Problem instances with upto 40 nodes can be solved within 5 seconds.

The parameter p has an effect on the number of links in the random graph, one expects running time to grow with p. This is indeed the case, see 5.4a. q determines the probability for a link to be an edge, for every edge two variables will be added to the MCPP LP. The probability for each direction of the link being added to the graph is $1 - q$. Thus with increasing q the number of variables decreases and consequently the running time, see 5.4b.



(a) $q = 0.5$, $p \in [0.1, 1.0]$ **(b)** $p = 0.5$, $q \in [0.0, 1.0]$

Figure 5.4: Running times for solving MCPP instances with $n = 20$ nodes and varying p, q for Euclidean Cost

Chapter 6

Frontend: The MCPP Game

The MCPP game implemented in this project is part of a series of visualizations[1] of different algorithmic problems and their corresponding algorithms. To provide a coherent interface, this game is based on the website of the TSP game[2] by Sebastian Lotz, which in turn is based on the design of Richard Stotz' Bellman Ford game[3].

In this section, we present this game that, realized as an online application, can be reached under: https://www-m9.ma.tum.de/material/en/mCPP-game/. As the game is rather intuitive, the reader is encouraged to try the game on his own and then may skip the first subsection which describes the functionality of the website. After that, we present the architecture of the website including the communication with the server that runs the solver application. Finally, we give some remarks on how to deploy the client and server in order to obtain a working system.

6.1 Functionality

We will present the functionality of the frontend of the game, which was realized as an online application. The website is organized in several tabs, which will be described in the chronological order of a normal usage.

When first visiting the webpage, one will see the introduction tab (6.1). It gives a short problem description of the MCPP and offers the user two choices how to proceed: Create a new game or read a more detailed problem and algorithm description.

If the user chooses to create a new game via the button on the introduction tab or choosing the Create Game Tab in the menu, the Create Game Tab (figure 6.2) is displayed. It offers a list of several predefined maps that can be used and the choice RANDOM. If this map is selected the parameters already described in section 5.3 can be specified in order to generate random graphs: one can choose the number of junctions in the map, the parameter $p \in [0, 1]$ and $q \in [0, 1]$. In the domain of the game p determines whether two junctions are connected by any street and q controls the probability of it being a narrow street or a oneway street.

[1] http://www-m9.ma.tum.de/Allgemeines/GraphAlgorithmenEn
[2] http://www-m9.ma.tum.de/material/en/tsp-game/
[3] http://www-m9.ma.tum.de/material/en/spp-bellman-ford/

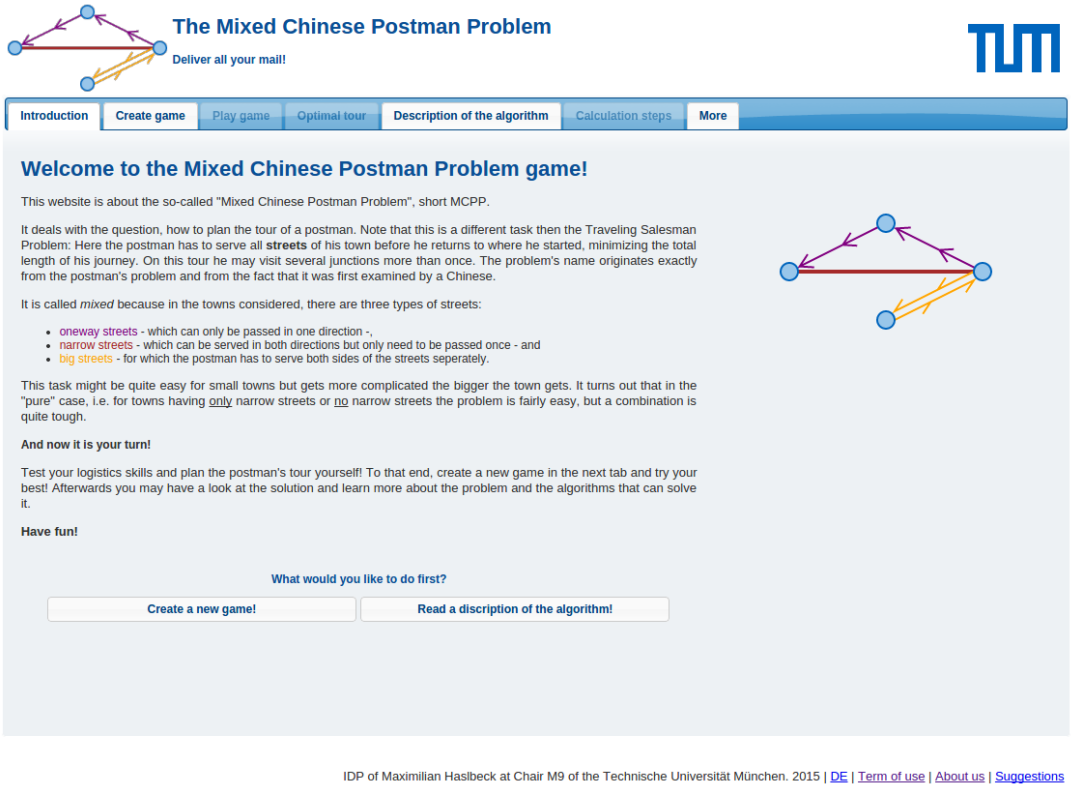


Figure 6.1: The introduction tab

Another option, besides generating a new problem instance, is to load an already existing problem instance by its GameCode. This enables the user to replay a certain problem instance.

When the user has generated or loaded a game instance, she will enter into the playing tab (figure 6.3). Here the map will be displayed and the player has the possibility to select a starting node by clicking on one of the nodes in the canvas. Then by clicking on other nodes, the user can move around in the map and design a tour through the map, that fulfills the conditions of an Euler tour. The description below the canvas explains the different colors of the links. The tip of the black arc always follows the users mouse movements so that it is clear where the current traversal of a street starts and ends. In this process, it is possible to undo the last move or start over again. Once the tour specified by the user served all the links and returned to the starting node (marked in light green), the game is over and the user is pointed to a visualization of the optimal solution: a button is enabled that leads to the solution tab.

In order to repeat the current game instance the user is able to view the corresponding

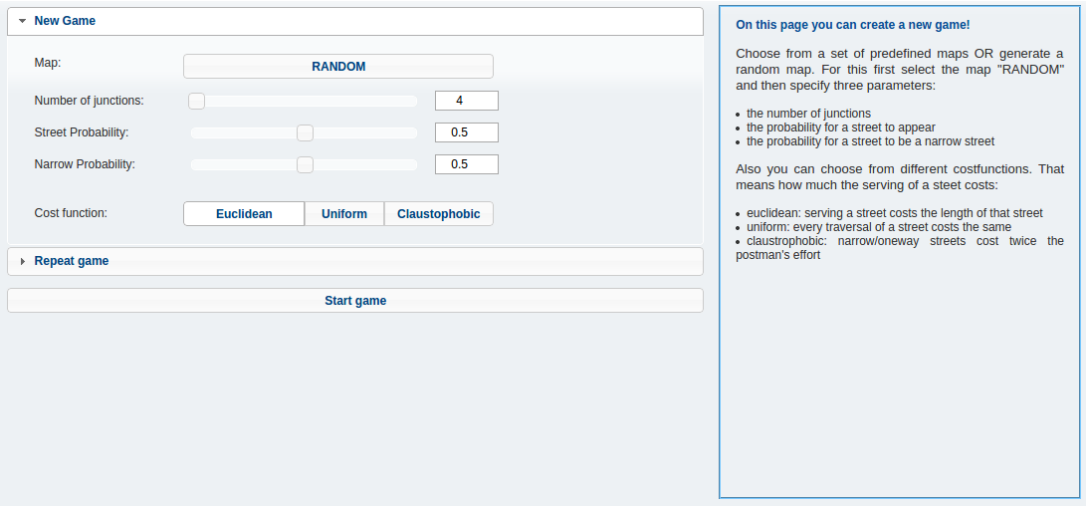


Figure 6.2: Creating a random map

GameCode.

Once the user ended the playing process, the optimal solution determined by the solver can be viewed side by side with the custom tour (figure 6.4). To that end, the map will be displayed again in the canvas with all edges replaced by two one-way streets. This makes it possible to display how often each direction of an edge was taken. As the map then only contains directed arcs, each of these is labeled by two numbers: the first denoting how often the custom tour traverses this arc, and the second how often the optimal solution uses this arc. On the right hand status window one can get more information about the optimal and custom tour: the concrete tours can be displayed and an animation can be startet that traverses this tour on the canvas.

Then the user can continue displaying some steps of the calculation of the optimal solution or view more information about the MCPP and the way it can be solved.

If the user decides to open the Calculation Steps Tab (figure 6.5), the map will again be displayed in the canvas. On the right hand side the user can choose different stages of the optimization process. The objective value of the current configuration is displayed in the list and on the map the value assigned to each link is displayed. In a feasible solution these values are all integers, but in the solution of LP relaxations some values are fractionals (these are marked in red). In the cutting plane algorithm, these infeasible solutions have to be cut off. This is done by adding Odd Cut Inequalities. The inequality used to cut off the given infeasible solution is visualized by showing the underlying partition of the vertices in the map (green vs. blue nodes).

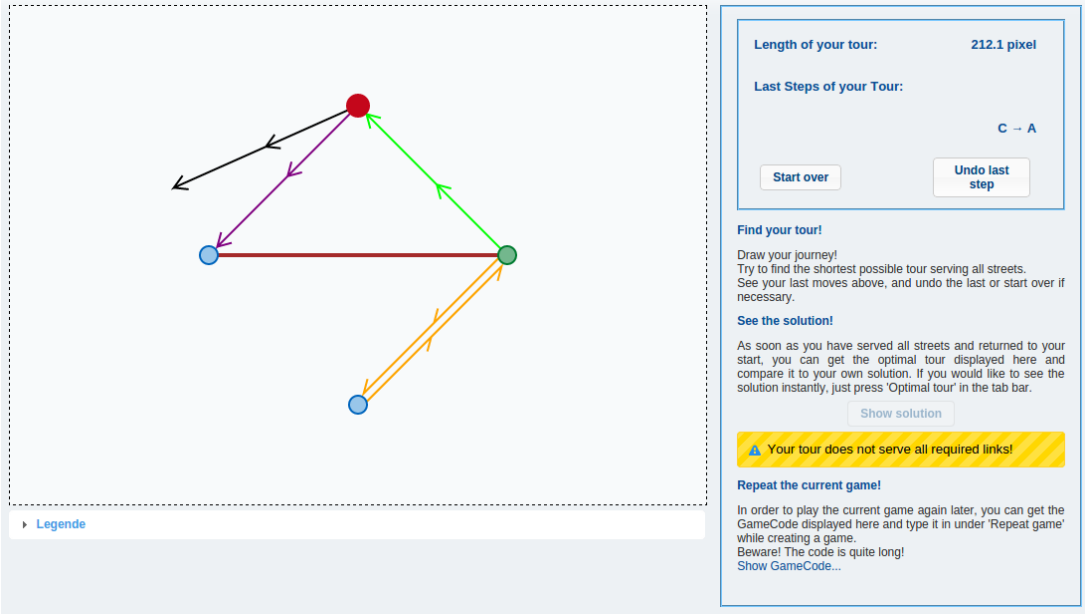


Figure 6.3: Playing the Game

6.2 Architecture

The application's architecture is separated into two parts: on the one hand the frontend, which is a website, handles the graphical user interface, including interactions of the user, display and animation of the maps. On the other hand, the backend, that serves requests of the client and kicks off the solving process.

First, we describe the implementation of the frontend by explaining the funcionality of all the important JavaScript files. After that, we illustrate a sample sequence of the communcication between frontend and backend.

6.2.1 The frontend

The client is implemented as a HTML website that uses JavaScript with the jQuery library to realize.

- A jQuery-JavaScript-library is included via jquery-1.10.2.js. It enables the manipulation of the Document Object Model (DOM). The jQuery User Interface defined in jquery-ui-1.10.4.custom.min.js and jquery-ui-1.10.4.custom.min.css specifies a new design to some HTML elements: buttons, tabs and the accordeon effect are customized there. The User Interface was configured by the so called "ThemeRoller" of the jQuery-UI- project. Customizations can be easily done via

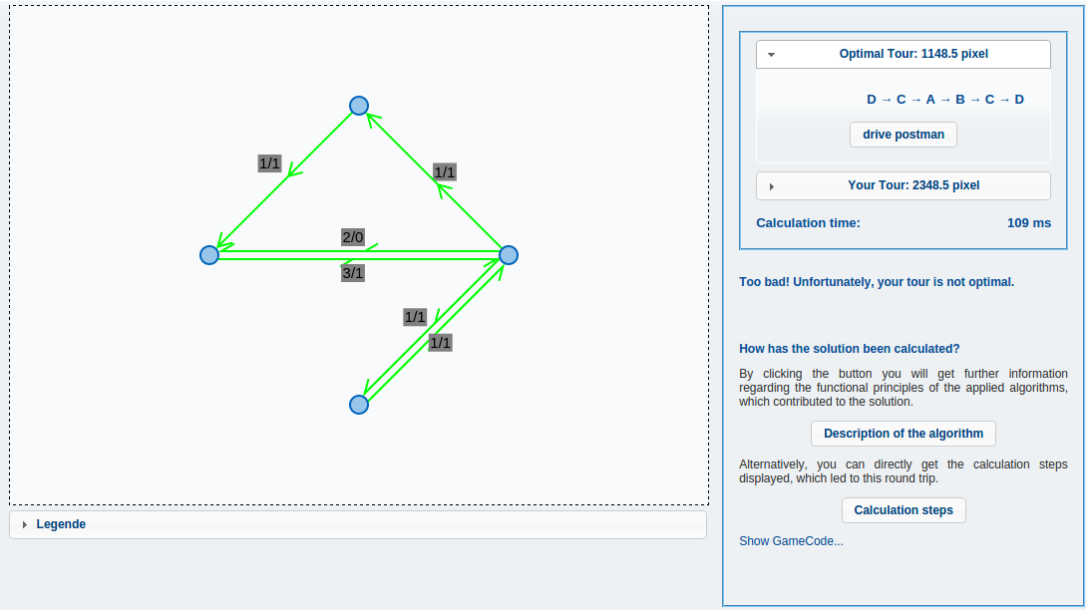


Figure 6.4: showing the solution

the path in the comment in the jquery-ui-1.10.4.custom.min.css. All three files are available under the "MIT-License", a copy resides in the folder /tum-theme2.

- The cascading Stylesheet **style.css** specifies the layout of the website.
- The file **config.js** contains additional parameters like the URL and port of the backend. Also the maximum number of junctions that can be used in random maps is determined there. Furthermore the standard language can be set via the variable *language*. At the moment only English (**EN**) is available, but more languages can easily be added.
- The file **siteAnimation.js** provides central methods for the animation of the website. The method *initializeSiteLayout* initializes the animations of the website by transforming HTML elements of buttons, tabs and accordions into jQuery-UI objects. It also adds all important handlers to define behaviour when buttons are clicked or tabs are changed. Also, in this stage the available maps are requested via the respective methods from *loadUtilities.js* and filled into the dropdown menu in the Create Game Tab.
- The class *CanvasDrawer* is defined in **canvasDrawer.js**: It contains all the vital methods that are needed by tabs that use a canvas in order to display a map. Basically when generating and initializising a new CanvasDrawer it adds the

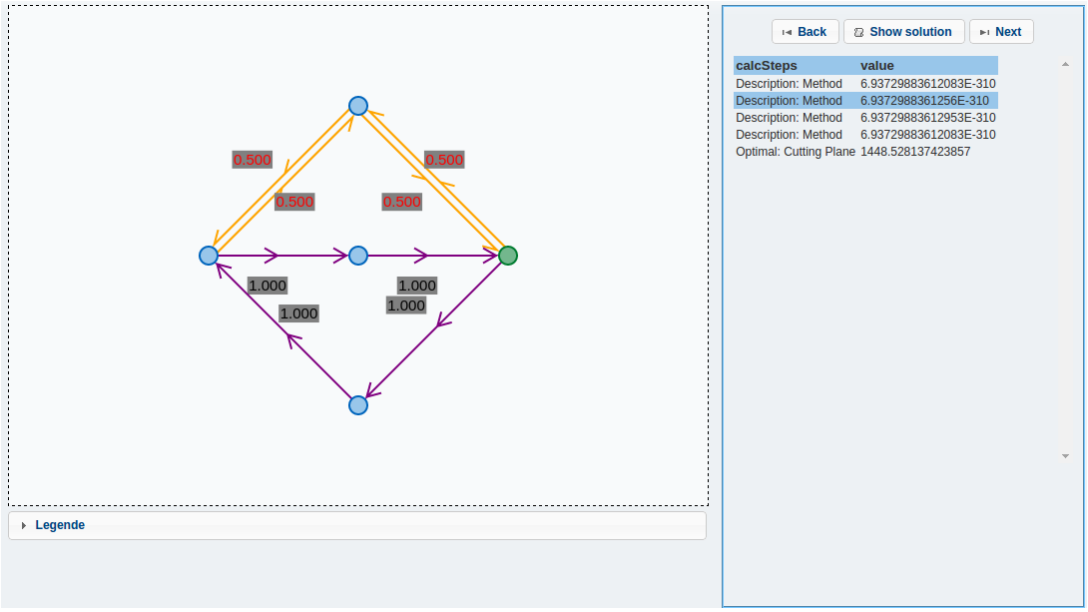


Figure 6.5: Showing calculation Steps

background image to the canvas and draws the graph it gets. If the user starts a new edge in the tour the graph will be painted every 20 milliseconds, in order to give the impression as if the user draws the edge. The display of city names when hovering over a node and dialogs when exiting certain tabs are implemented here as well. For example, when exiting the Play Game Tab a warning dialog is displayed, stating that after exiting the game will be ended.

- The class *GraphDrawer* located in **graphDrawer.js** inherits from *CanvasDrawer*. This class provides all important methods for the interaction of the user when in the Playing Game Tab: this includes all mouse handler (e.g. for detecting if the mouse is located over a junction, starting and drawing of the tour), button handlers and tests whether the tour is valid.
- In **solutionGraphDrawer.js** and **calculationGraphDrawer.js** other classes inheriting from *CanvasDrawer* are defined. They handle the display of the map in the Optimal Tour tab and Calculation Steps tab respectively.
- All methods that actually draw on the canvas are defined in **canvasUtilities.js**: one method for nodes, one for links.
- The file **canvasElements.js** defines the classes Node, Edge and Graph and thus is in between *CanvasDrawer* and *CanvasUtilities* because it uses the latter to draw the elements when it is requested by the first to do so.

- The communication to the backend is encapsulated in **loadUtilites.js**: *checkCalculationFinished* checks the status of the current calculation, *loadSolution* requests subsequently the solution from the backend and saves a solution object, *loadAvailableSettings* loads available parameters and maps and creates the mapConfig object, *loadNewGame* hands the specified parameters from the Create Game tab to the backend and generates a gameConfig object from the answer.

- The last file **metrics.js** contains an interface for the metrics and the implementation of the different instances euclidean, uniform and claustrophobic. The Euclidean metrics assigns every edge the euclidean distance between its endpoints as weight, whereas uniform assignes all weights to 1 and claustrophobic assigns 1 to all edges and 2 to all arcs.

6.2.2 Communication to the backend

In Figure 6.6, the sequence of a typical communication between the frontend and the backend is illustrated.

It starts with the user calling the website, while initializing the site layout the frontend requests all available settings including the maps by a GET request to the backend.

The backend calls AvailableSettings.jar which reads the local settings and the existing maps and returns the result to the frontend. This enables the website to generate the Create Game tab. The user then may create a new map by specifying a metric ME and either a map name MAP or choosing map "RANDOM" and some values for n, p and q. In the former case a GET request to /mcpp/initiateCalculation/MAP/ME is issued, in the latter the frontend requests /mcpp/initiateRandomCalculation/n/p/q/ME. In any case, the backend passes on the information to MCPP.jar which loads or generates a map and both returns the map details and starts the solving process. Meanwhile the calculation starts the backend already passes the map information back to the frontend which is then able to draw the map and let the user play around trying to find the optimal tour.

Once the user wants to display the optimal solution, the frontend asks whether the calculation is already done. The backend maintains a list of gameCodes which are unknown, still calculating or already solved. This flag is returned upon request to the frontend. Once the calculation ended, MCPP.jar writes the result into a file named by its GameCodeadds it to the solution folder and signals the backend server process that this GameCode is done. On the next request of the frontend the backend can answer positively, whereupon the frontend requests the solution of this GameCode and the backend provides it. From this information the website can display the optimal tour as well as the calculation steps.

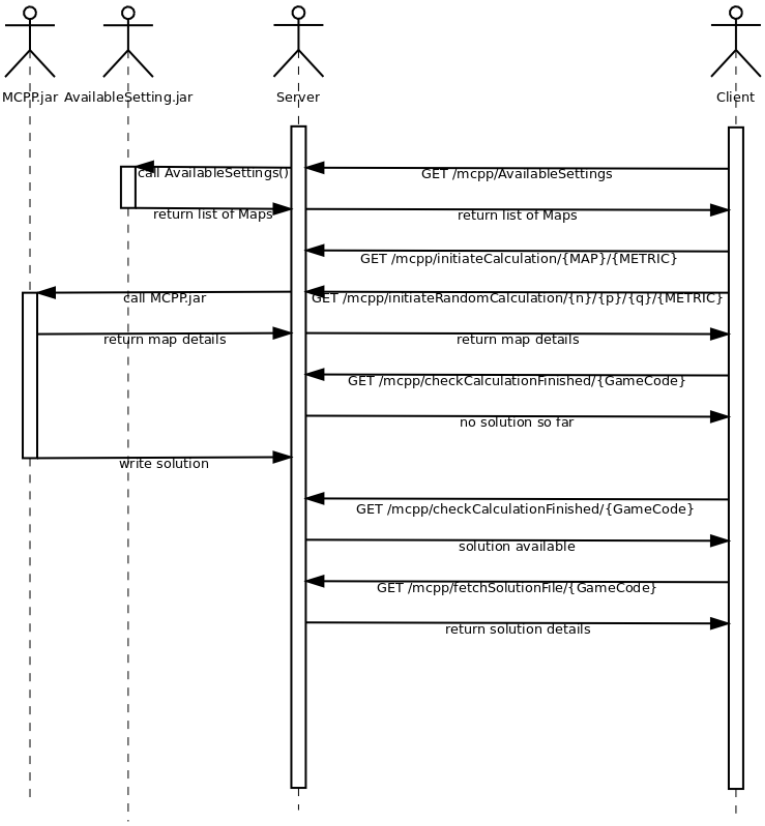


Figure 6.6: Sequence Diagram of a typical usecase

6.3 Deployment

For deploying the client, just copy the folder /Client to some webserver. All images are available and the JavaScript is run on the user side. Only the address and port of the backend has to be specified in *config.js*.

Setting up the backend is slightly more involved, as the MCPP solver requires Gurobi and JGraphT. Gurobi has to be installed and – for linux systems – the environment variables have to be set (following the Gurobi quickstart manual [4]). The required libraries for Gurobi and JGraphT are included in /Server/MCPP_lib. The backend is implemented as a CherryPy Server. Thus, Python and CherryPy have to be installed. Finally the server can be configured with the file *MCPP_Config.conf* and the server application *MCPP_CherryPy.py* can be started.

[4] http://www.gurobi.com/documentation/6.0/quickstart_linux.pdf

Chapter 7

Conclusion

This project first collected the reviewed literature about the Mixed Chinese Postman Problem. After that, a solver was implemented that uses a representation of the problem instances as ILPs and solves them with a cutting plane algorithm using the LP solver Gurobi. Experimental results show that problem instances upto 40 vertices can be solved in reasonable time.

Finally an educational game was designed and set up for visualizing the problem and providing some information about the MCPP.

Future work:

Further enhancements of the development could possibly be:

- come up with some additional families of inequalities and associated separation algorithms
- implement the separation algorithm also for *balanced-set inequalities* by Norbert and Picard algorithm and thus come up with a cutting plane algorithm using formulation F1
- redo the experiments of Corberán et al. (for MCPPs) in order to relate F1 and F2.[1]
- generalize to MGRP as in [CMS06]

[1] The used networks can be found under http://www.uv.es/corberan/instancias.htm

Bibliography

[CMS06] A. Corberán, E. Mota, and J. M. Sanchis. "A comparison of two different formulations for arc routing problems on mixed graphs". In: *Computers & OR* 33.12 (2006), pp. 3384–3402.

[CP10] A. Corberán and C. Prins. "Recent results on Arc Routing Problems: An annotated bibliography". In: *Networks* 56.1 (2010), pp. 50–69. ISSN: 1097-0037.

[CPS05] A. Corberán, I. Plana, and J. M. Sanchis. *A Branch & Cut Algorithm for the Windy General Routing Problem*. Tech. rep. Technical Report TR04-2005. Department of Statistics and OR, University of Valencia (Spain). Submitted to Networks, 2005.

[CPS06] A. Corberán, I. Plana, and J. M. Sanchis. "Zigzag inequalities: a new class of facet-inducing inequalities for Arc Routing Problems". English. In: *Mathematical Programming* 108.1 (2006), pp. 79–96. ISSN: 0025-5610.

[CRS03] A. Corberán, A. Romero, and J. M. Sanchis. "The mixed general routing polyhedron". In: *Mathematical Programming* 96.1 (2003), pp. 103–137.

[EGL95a] H. A. Eiselt, M. Gendreau, and G. Laporte. "Arc Routing Problems, Part I: The Chinese Postman Problem". In: *Operations Research* 43.2 (1995), pp. 231–242. eprint: http://dx.doi.org/10.1287/opre.43.2.231.

[EGL95b] H. A. Eiselt, M. Gendreau, and G. Laporte. "Arc Routing Problems, Part II: The Rural Postman Problem". In: *Operations Research* 43.3 (1995), pp. 399–414. eprint: http://dx.doi.org/10.1287/opre.43.3.399.

[EJ73] J. Edmonds and E. L. Johnson. "Matching, Euler tours and the Chinese postman". English. In: *Mathematical Programming* 5.1 (1973), pp. 88–124. ISSN: 0025-5610.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Vol. 1962. Princeton Princeton University Press, 1962.

[Fre79] G. N. Frederickson. "Approximation algorithms for some postman problems". In: *Journal of the ACM (JACM)* 26.3 (1979), pp. 538–554.

[GH61] R. E. Gomory and T. C. Hu. "Multi-Terminal Network Flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570. eprint: http://dx.doi.org/10.1137/0109047.

[Gur15] I. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2015.

[LRT08] A. N. Letchford, G. Reinelt, and D. O. Theis. "Odd Minimum Cut Sets and b-Matchings Revisited". In: *SIAM Journal on Discrete Mathematics* 22.4 (2008), pp. 1480–1487. eprint: http://dx.doi.org/10.1137/060664793.

[Pap76] C. H. Papadimitriou. "On the complexity of edge traversing". In: *Journal of the ACM (JACM)* 23.3 (1976), pp. 544–554.

[PC99] W. L. Pearn and J. B. Chou. "Improved solutions for the Chinese postman problem on mixed networks". In: *Computers & Operations Research* 26.8 (1999), pp. 819–827.

[PL95] W. L. Pearn and C. M. Liu. "Algorithms for the Chinese postman problem on mixed networks". In: *Computers & operations research* 22.5 (1995), pp. 479–489.

[PR82] M. W. Padberg and M. R. Rao. "Odd minimum cut-sets and b-matchings". In: *Mathematics of Operations Research* 7.1 (1982), pp. 67–80.