Technische Universität Berlin

Fakultät II - Institut für Mathematik

Bachelorarbeit
im Studiengang Mathematik

Node Partitioning and Subtours Creation Problem (NPSC)

Angefertigt von: Gioni Mexi
Matrikelnummer: 365153

Betreut von:
Prof. Dr. Thorsten Koch

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin,

. . . . . .

Unterschrift: Gioni Mexi

Zusammenfassung

Die Überwachung von Bereichen wird zunehmend von unbemannten Luftfahrzeugen (UAVs) durchgeführt, da diese Informationen über ein Ziel aus großer Entfernung und Höhe sammeln können. UAVs können insbesondere Ziele überwachen, die auf andere Weise nicht zugänglich sind. Die Problemstellung dieser Arbeit basiert auf der Annahme, dass die Inspektion eines jeden Bereichs regelmäßig innerhalb einer sogenannten kritischen Zeit erfolgen muss. Die kritische Zeit eines Bereiches ist als obere Grenze für den Zeitraum zwischen zwei aufeinanderfolgenden Besuchen dieses Bereichs zu verstehen. Jeder Bereich darf nur von einem UAV besucht werden und die entstehenden Routen müssen Kreise sein. Ziel des Node Partitioning and Subtours Creation Problems (NPSC) ist es die minimale Anzahl von UAVs zu bestimmen, die nötig ist, um alle Bereiche innerhalb ihrer kritischen Zeit zu besuchen.

Im Rahmen dieser Arbeit definieren wir das NPSC Problem mathematisch und zeigen, dass es NP-schwer ist. Die Größe des Problems wird mithilfe von Preprocessing-Techniken reduziert und es wird eine Heuristik implementiert, die in kurzer Zeit zullässige Lösungen generiert. Anschließend formulieren wir insgesamt vier (nichtlineare) gemischt-ganzzahlige Programme (engl. Mixed Integer (Nonlinear) Program, MI(NL)P): Modell A, B, B+ und MTZ. Modell A ist ein MINLP und wird in Modell B linearisiert. Modell B+ enthält als Erweiterung von Modell B zusätzliche zulässige Ungleichungen. In allen drei Modellen A, B und B+ werden unzulässige Lösungen, in denen mindestens ein UAV mehreren zyklischen Routen zugeordnet ist, während des Lösungsprozesses durch Schnittebenen abgeschnitten. Modell MTZ weist hingegen eine polynomielle Anzahl von zusätzlichen Variablen und Ungleichungen auf, die diese unzulässigen Lösungen aus dem Lösungsraum entfernt. Die MI(NL)P Modelle werden unter Verwendung von modernen Optimierungssolvern unter Einbeziehung von Preprocessing-Techniken und der Heuristik gelöst. Im Anschluss werden die Modelle anhand von experimentellen Rechenergebnisse verglichen. Der Vergleich der Modelle A, B, B+ und MTZ zeigt, dass Modell B+ und MTZ in der Lage sind, größere Probleminstanzen in Bezug auf die Anzahl der zu besuchenden Bereiche zu lösen als Modell A und B. Zusammenfassend gelang uns die Etablierung von zwei Modellen B+ und MTZ, die als Grundlage unserer weiteren Arbeit dienen.

Abstract

Area guarding tasks are vastly executed by Unmanned Aerial Vehicles (UAVs), since they can gather information about areas from long distance or high altitude. Moreover, they are able to visit areas that are not accessible in other ways. We are concerned with patrolling a set of areas under critical time constraints. The critical time of an area is an upper bound on the time between two consecutive patrols of this area by an UAV. In addition, we assume that each area is visited by exactly one UAV and that each UAV is assigned a cyclic route, which means that it starts and ends its route at the same area and visits all other assigned areas exactly once. Goal of the Node Partitioning and Subtours Creation Problem (NPSC), is to find the minimum number of UAVs, in order to regularly visit all areas within their critical time.

In this work, we mathematically define NPSC and prove its NP-hardness. Further, we introduce preprocessing techniques to reduce the problem size and implement a heuristic, which generates feasible solutions for our test instances in short amounts of time. Next, we formulate four different Mixed Integer (Nonlinear) Programs (MI(NL)P): Model A, B, B+ and MTZ. Model A is a MINLP. The nonlinear constraints of Model A are linearized in Model B. Model B+ is an extension of Model B by including further valid inequalities. In all of the previous models infeasible solutions, where at least one UAV is assigned multiple cyclic routes, are cut off during the optimization process. Model MTZ however is compact, i.e., compared to B+ we include an additional polynomial number of variables and constraints in order to remove such infeasible solutions right from the start. Finally, all MI(NL)P models are solved for randomly generated test instances with state-of-the-art optimization solvers, and their performance is compared. The comparison of the models shows that Model B+ and Model MTZ are able to solve larger problem instances than models A and B.

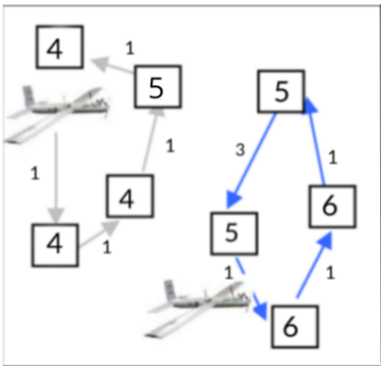
Contents

1. Introduction

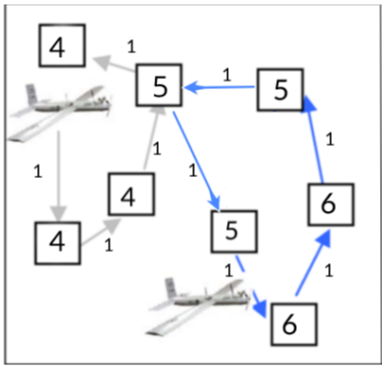
1.1. Motivation

Unmanned aerial vehicles (UAVs) are vastly used to execute surveillance tasks, since they can gather information about an area from long distance or high altitude. In particular, they are able to visit areas that are not accessible in other ways. The node partitioning and subtours creation problem (NPSC), which this thesis deals with, was established by Burdakov [6] and originates from the optimal scheduling of such surveillance UAVs. Given a set of areas $V = \{1, 2, ..., N\}$ to guard, our goal is to find the minimum number of UAVs and an individual flying route for each of them to successfully guard all areas, while meeting three side constraints. First, each area $i \in V$ has a critical time $T_i \in \mathbb{R}_{\geq 0}$, which is an upper bound on the duration it can remain unattended, and a scanning time $S_i \in \mathbb{R}_{\geq 0}$, which is the amount of time an UAV needs to scan area i. This means that after scanning an area i for S_i time units, the UAV must return to i within T_i time units and rescan the area. Second, the UAVs must fly in cyclic routes, which means that an UAV starts and ends its flying route at the same area and visits all other assigned areas exactly once. We assume that an UAV continues on the same flying route after finishing it without any delay. Third, exactly one UAV is assigned to guard each area, i.e., each area is contained in exactly one route, since the intersection of flying routes would result in complications related to the necessity of introducing additional flight schedules to avoid possible collisions.

Figure 1.1 illustrates two examples of invalid assignments. Here, each square represents an area and the cyclic route for each UAV is denoted by the arrows. The numbers in the squares denote the critical times of the areas and the numbers on the arrows the durations of the flights between them. In this example we assume that $S_i = 0,\ \forall i \in V$. In Figure 1.1a the flying route with blue arrows can be completed by an UAV within 6 time units. However, two of the areas must be revisited within 5 time units, thus this assignment is invalid. In Figure 1.1b, even though all areas can be revisited within their critical time, the assignment is invalid, since the flying routes share a common node.
Lastly, in Figure 1.2 the flying routes do not intersect and each area can be revisited within its critical time. Hence, the assignment is valid.



(a) Areas cannot be revisited within their critical time.



(b) Area contained in two flying routes.

Figure 1.1.: Examples of invalid assignments.

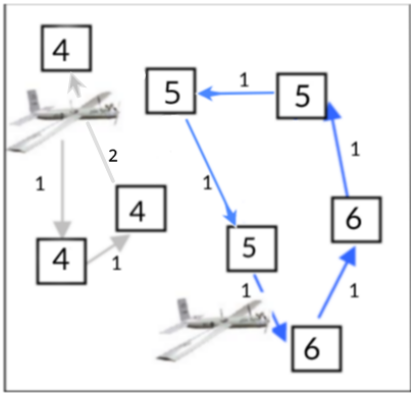


Figure 1.2.: Valid assignment.

1.2. Problem Formulation and Definitions

A general reference for basic graph theory is the book by Bondy and Murty [4], where the following definitions, which are important for the definition of NPSC, are adapted from.

Definition 1. *Let $G = (V, E)$ be an undirected graph, and $S \subseteq V$ non-empty. Then, the induced subgraph $G[S]$ is the graph whose node set is S and whose edge set consists of all edges that have both endpoints in S.*

Definition 2. *A cycle C that contains every node of an undirected graph $G = (V, E)$ exactly once is called a Hamiltonian cycle in G. We will refer to Hamiltonian cycles as tours.*
A subtour C' in G is a tour in a subgraph G' of G.
The total length $\tau_{C'}$ of a subtour C' is defined as:

$$\tau_{C'} = \sum_{\{i,j\} \in C'} t_{i,j} \,,$$

where $t_{i,j} \in \mathbb{R}$ is the weight of edge $\{i, j\} \in E$. We denote a cycle C as a vector of the visited nodes, i.e., $C = [v_0, ..., v_l, v_{l+1}, ...v_{l'}]$, which means that after node v_l node v_{l+1} is visited, and after node $v_{l'}$ we return to the start node v_0.

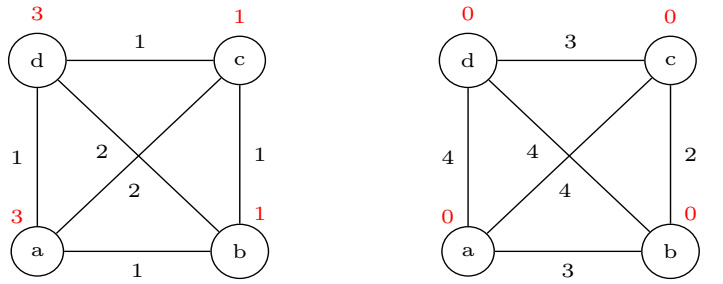
Next, we are going to give a mathematical formulation of the NPSC problem. For an instance $\mathcal{I}$ of NPSC we are given an undirected graph $G_{\mathcal{I}} = (V, E)$, where the set of nodes $V = \{1, 2, ..., N\}$ represents the areas and the set of edges $E \subseteq V \times V$ represents paths between them. Thereby, each node $i \in V$ is assigned a weight $T_i \in \mathbb{R}_{\geq 0}$, which is the critical time of the area it represents and a scanning time $S_i \in \mathbb{R}_{\geq 0}$. Further, each edge $\{i, j\} \in E$ has an edge weight $t_{i,j} \in \mathbb{R}_{\geq 0}$, which corresponds to the flying time between area i and area j. A cyclic route of an UAV is a cycle C in $G_{\mathcal{I}}$ that traverses $E' \subseteq E$ edges and visits $V' \subseteq V$ nodes. and is completed in total time τ_C equal to:

$$\tau_C = \sum_{\{i,j\} \in E'} t_{i,j} + \sum_{\{i\} \in V'} S_i \,.$$

W.l.o.g. we can assume that there are no scanning times, since these can be added to the edge weights as follows: For each edge $\{i, j\} \in E,\ i \neq j$ we can redefine the edge weights as $t'_{i,j} = t_{i,j} + (1/2) \cdot S_i + (1/2) \cdot S_j$. Then,

$$\sum_{\{i,j\} \in E'} t'_{i,j} = \sum_{\{i,j\} \in E'} t_{i,j} + \sum_{\{i\} \in V'} S_i \,.$$

Example. Figure 1.3a shows an instance with scanning times (red labels) and Figure 1.3b the same instance with redefined edge weights and no scanning times. It is easy to check that the total flying time to complete each cyclic route is in both cases the same.



(a) Example with scanning times.

(b) Equivalent example with redefined edge weights.

Figure 1.3.: Redefining edge weights.

Hence, in the following we denote an NPSC instance as a 4-tuple $\mathcal{I} = (V, E, T, t)$. We will refer to NPSC as metric-NPSC if $G_{\mathcal{I}}$ is complete and for all nodes $i, j, k \in V$, the triangle inequality $t_{i,j} \leq t_{i,k} + t_{k,j}$ holds.

A solution of an instance $\mathcal{I}$ of NPSC is a tuple $(\mathcal{N}, \mathcal{C})$, where $\mathcal{N} = \{N_1, N_2, ..., N_k\}$ is family of $k \in \mathbb{N}$ non-empty, pairwise disjoint subsets that cover V, i.e.,

$$N_l \cap N_m = \emptyset,\ \forall\, N_l, N_m \in \mathcal{N},\ l \neq m \,, \tag{1.1}$$

and

$$V = \bigcup_{N_m \in \mathcal{N}} N_m \tag{1.2}$$

hold, and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ is a family of cycles, where each $C_m \in \mathcal{C}$ is a tour in $G_{\mathcal{I}}[N_m]$.

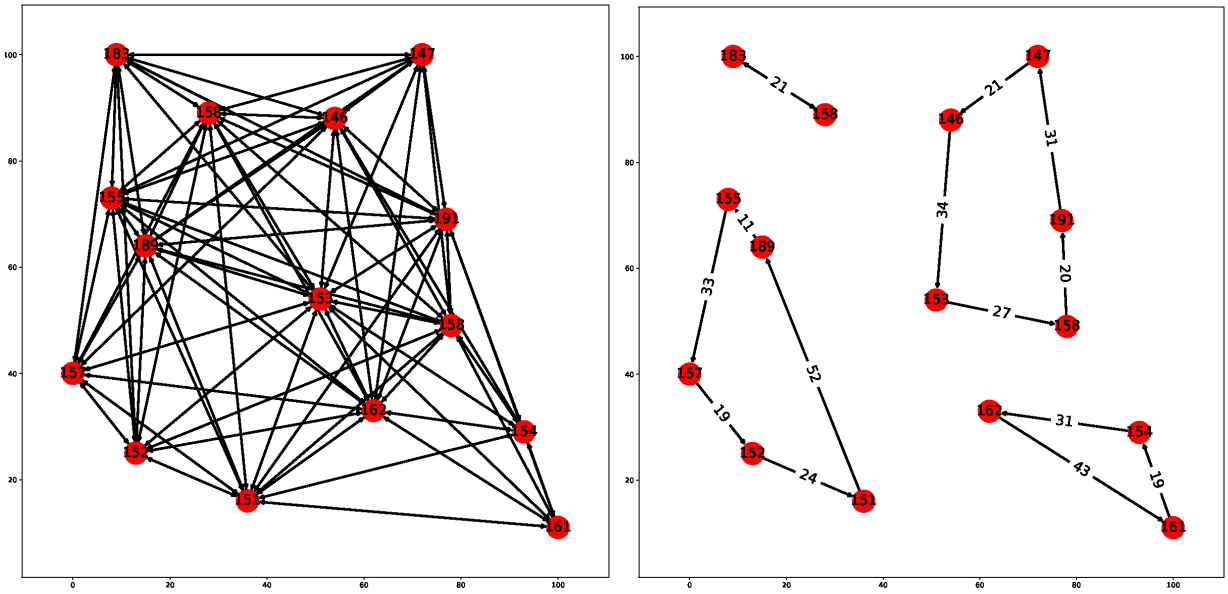
Moreover, if

$$\tau_{C_m} \leq \min_{i \in N_m} T_i, \forall\, m = 1, 2, ..., k \tag{1.3}$$

we call $(\mathcal{N}, \mathcal{C})$ a feasible solution of instance $\mathcal{I}$, and each subtour $C_m \in \mathcal{C}$ a feasible subtour in $G_{\mathcal{I}}$. Conditions (1.1) and (1.2) ensure that each node is contained in exactly one subtour and inequalities (1.3) ensure that each node is revisited within its critical time. Our goal is to find a feasible solution $(\mathcal{N}, \mathcal{C})$ with a minumum number of subsets, i.e., a tuple $(\mathcal{N}, \mathcal{C})$ fulfilling (1.1) - (1.3) and minimizing $|\mathcal{N}|$.

Remark. *For algorithmic and notational purposes we assume that there is a loop at each node $i \in V$ having length zero, i.e., $\{i, i\} \in E$ and $t_{i,i} = 0$ for all $i \in V$. Hence, there exists the so-called trivial solution for each instance $\mathcal{I}$ of NPSC, where every subset consists of one node, and each feasible subtour is a self-loop, i.e., $(\mathcal{N}, \mathcal{C}) = (\{\{1\}, ..., \{N\}\}, \{[1], ..., [N]\})$.*

Figure 1.4a illustrates an example of a metric-NPSC instance with 15 nodes and node weights denoted in each of them. In this example the edge weights are given by the euclidean distances of the nodes. Figure 1.4b shows an optimal solution for this instance.



(a) NPSC instance

(b) An optimal solution containing 4 subtours.

Figure 1.4.: NPSC example.

NPSC is a NP-hard problem. We can prove this by using a polynomial reduction from the decision variant of TSP, as shown in **Lemma 2**. Details on TSP can be found in the work of Applegate et al. [2]. For details on complexity theory and polynomial reductions we refer to Garey and Johnson [12].

Lemma 1. *NPSC is NP-hard.*

Proof. Claim 1: NPSC is in NP.
Let $\mathcal{I} = (V, E, T, t)$ be an NPSC instance and $(\mathcal{N}, \mathcal{C})$ be a candidate solution. Clearly we can check in polynomial time if conditions (1.1) - (1.3) are fulfilled. Hence, NPSC is in NP.

Claim 2: TSP $\leq_P$ NPSC, i.e., the decision variant of TSP is polynomially reducible to NPSC. We define TSP as in Garey and Johnson [12] (Problem [ND22]).
INSTANCE: Let $G = (V, E)$ be a complete undirected graph with nodes $V = \{1, 2, ..., N\}$, edges E, and edge weights $t_{i,j} \in \mathbb{R}_{>0}$ for each $\{i, j\} \in E$.
QUESTION: Is there a tour of V having length $\tilde{T} \in R_{>0}$ or less, i.e., does there exist a permutation $\pi(1), \pi(2), ..., \pi(N)$ of V such that

$$\left(\sum_{i=1}^{N-1} t_{\pi(i),\pi(i+1)}\right) + t_{\pi(N),\pi(1)} \leq \tilde{T} \ ? \tag{1.4}$$

By setting the node weights $T_i := \tilde{T}$ forall $i \in V$, we directly derive an instance $\mathcal{I} = (V, E, T, t)$ of NPSC. If (1.4) holds, then there exists a tour C with total weight at most $\tilde{T}$, containing all nodes of V. Hence, $(\mathcal{N}, \mathcal{C}) = (\{V\}, \{C\})$ is an optimal solution of instance $\mathcal{I}$. On the other hand, if there exists an optimal solution $(\mathcal{N}, \mathcal{C})$ of NPSC such that $|\mathcal{N}| = 1$, then by (1.3) there exists a tour C in $G_{\mathcal{I}}$ with length at most $\tilde{T}$, satisfying (1.4). Therefore, TSP $\leq_P$ NPSC.

Claim 3: NPSC is NP-hard.
We showed that NPSC $\in$ NP and TSP $\leq_P$ NPSC. Therefore NPSC is NP-hard. □

1.3. Related Work

Similar problems of visiting the nodes of a graph under constraints have been studied extensively. However, to the best of our knowledge, no previous work has been done to solve NPSC. Some of the most studied combinatorial optimization problems related to NPSC are the TSP variant with deadlines (DTSP), see for example Bockenhauer et al. [3], the TSP variant with time windows (TSPTW), see for example Dumas et al. [11], the Vehicle Routing Problem (VRP), see for example Laporte [18] and the Vehicle Routing Problem with Time Windows (VRPTW), see for example Desrochers et al. [9].

DTSP, is defined on a complete undirected weighted graph $G = (V, E)$, with a starting node $s \in V$ and a deadline function $d : V \mapsto \mathbb{Z}_{\geq 0}$. The objective is to find a minimum weight Hamiltonian path satisfying all deadlines. In TSPTW we replace the deadline of each node $i \in V$ by a time window interval $[a_i, b_i]$ and we modify the objective into finding a minimum weight Hamiltonian path such that all nodes are visited within their time windows. In both problems, DTSP and TSPTW, the problem is infeasible if there exists no Hamiltonian path satisfying all deadlines or time window constraints, respectively. In NPSC we do not search for a Hamiltonian path through all nodes, but for a set of subtours, such that each node is contained in exactly one subtour. Another difference is that in NPSC the critical time is an upper bound between two consecutive visits of a node, but in DTSP and TSPTW, the nodes have to be visited only once within their deadline and time window, respectively.

The classical VRP can be described as the problem of finding an optimal collection of routes for a fleet of vehicles from a depot to a number of customers. VRP can be defined on a complete weighted undirected graph $G = (V, E)$, where the node set V represent the depot and the customers, and the edge set E represents direct connections between the nodes. The edge weight of an edge $\{i, j\} \in E$ is the traveling time $t_{i,j} \in \mathbb{R}_{\geq 0}$ between i and j. In VRP each vehicle begins and ends it's route at the depot and each customer is visited exactly once by exactly one vehicle. In many extensions of VRP constraints on the maximum traveling time of each vehicle are included. A variant of VRP is the VRPTW, where each customer $i \in C$ has a time window, which is an interval $[a_i, b_i]$, and a service time $s_i \in \mathbb{R}_{\geq 0}$. This means that the service of each customer must start within the time window, and the vehicle must stop at the customers location for exactly s_i time units. If a vehicle arrives at a customer too early, it has to wait. The goal of VRP and VRPTW is to find a set of vehicle routes with minimum total traveling time. The major differences between VRP(TW) and NPSC are the following: In VRP(TW) the number of vehicles is fixed and all of them start and end their tour at the depot. In NPSC however, the subtours do not share any common nodes and there are no explicit start nodes given. Further, in NPSC the number of subtours is not fixed, since our objective is to minimize their number.

Another difference is that in NPSC the critical time is an upper bound between two consecutive visits of a node, but in the VRPTW the time window is an interval in which each customer should be served only once.

Lastly, a research similar to our work is the *Cyclic routing of unmanned aerial vehicles* (CR-UAV) done by Drucker et al. [10]. Similarly, the objective of CR-UAV is to minimize the number of vehicles patrolling a set of areas unter critical time constraints. The only difference between CR-UAV and NPSC is that in CR-UAV the areas can be visited by more than one vehicle. In CR-UAV a lower- and upper-bound on the number of required UAVs were proposed and a reduction of the problem to a Boolean combination of "difference constraints" (constraints of the form $x - y \geq c$, $x, y \in \mathbb{R}$, c constant) is suggested. The problem is solved with Satisfiability Modulo Theories (SMT) solvers. Ho and Ouaknine [16] were able to show that CR-UAV is PSPACE-complete even in the case of a single UAV.

1.4. Outline

In this section we give a brief outline of this work. **Chapter 2** proposes preprocessing steps, in order to reduce the size of the graph $G_{\mathcal{I}}$, which delete edges, which cannot be part of any feasible solution. Furthermore, concepts of finding infeasible subtours with more than two nodes are discussed, and a lower bound for NPSC is given. **Chapter 3** presents the IaR-heuristic, inspired by the insertion heuristic for TSP, which generates a feasible solution for NPSC problem instances. This heuristic solution is considered as an upper bound for the number of subtours in an optimal solution. In **Chapter 4** we derive compact mathematical programming formulations for NPSC by describing four different programming models. Further, we discuss ways to detect infeasible solutions during the optimization process and how to cut them off by adding further constraints. In **Chapter 5** we generate random metric-NPSC test instances, which are then solved with state-of-the-art mathematical programming solvers. We discuss the quality of the heuristic and we compare the performance of the programming models. **Chapter 6** summarizes the results of this thesis and suggests possible future work.

2. Preprocessing

In the following, let $\mathcal{I} = (V, E, T, t)$ be an NPSC instance. The aim of our preprocessing techniques is to reduce the size of $G_{\mathcal{I}}$ by deleting edges, which cannot be contained in any feasible tour. Furthermore, we show that a decomposition of the problem in subproblems is possible if $G_{\mathcal{I}}$ consists of more than one connected components. Lastly, we introduce a conflict graph for NPSC, where two nodes share an edge if and only if they cannot be contained in the same subtour in any feasible solution.

2.1. Graph Preprocessing

Let $s_{i,j} \in \mathbb{R}_{\geq 0}$ be the length of a shortest path between $i, j \in V$. If no path between i and j exists, we define $s_{i,j} = +\infty$. Commonly used algorithms for finding all shortest paths in a graph in polynomial time are the ones of Dijkstra and Bellman-Ford, which can be found for in the book of Cormen et al. [7].

Lemma 1. *Let $\{i, j\} \in E$. If*

$$t_{i,j} + s_{i,j} > \min\{T_i, T_j\}, \tag{2.1}$$

then $\{i, j\}$ cannot be contained in any feasible subtour in $G_{\mathcal{I}}$.

Proof. Let C_m be a feasible subtour in $G_{\mathcal{I}}$. Assume that C_m contains an edge $\{i, j\} \in E$ such that $t_{i,j} + s_{i,j} > \min\{T_i, T_j\}$. Since (1.3) holds, we have that

$$\begin{aligned}
\min_{i \in N_m} T_i &\geq \tau_{C_m} \\
&= \sum_{\{i',j'\} \in C_m} t_{i',j'} \\
&= t_{i,j} + \sum_{\{i',j'\} \in C_m \setminus \{i,j\}} t_{i',j'} \\
&\geq t_{i,j} + s_{i,j} \\
&> \min\{T_i, T_j\} \\
&\geq \min_{i \in N_m} T_i \,,
\end{aligned}$$

which is a contradiction. Hence $\{i, j\} \notin C_m$. □

Remark. *In the metric-NPSC case, since $t_{i,j} = s_{i,j}$ for all $\{i, j\} \in E$, (2.1) can be written as $2 \cdot t_{i,j} > \min\{T_i, T_j\}$.*

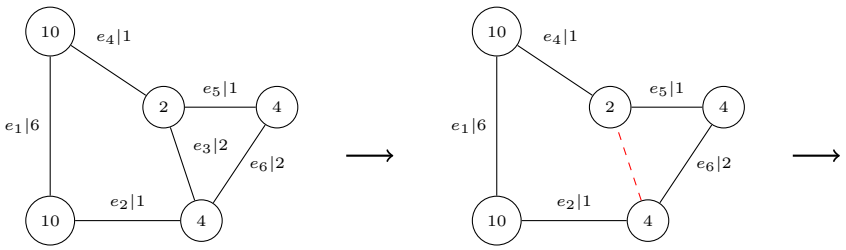
In the following, we call an edge $e \in E$ satisfying (2.1) a bad edge. Therefore, since no feasible subtour in $G_{\mathcal{I}}$ can contain a bad edge, we can delete it from $G_{\mathcal{I}}$.

Algorithm 1 Removal of Bad Edges

```
1: repeat ← True
2: while repeat do
3:     repeat ← False
4:     for all {i, j} in E do
5:         compute s_{i,j}
6:         if t_{i,j} + s_{i,j} > min{T_i, T_j} then
7:             delete edge {i, j} from E
8:             repeat ← True
```

In **Algorithm 1** the procedure of identifying bad edges is repeated, since after removing a bad edge e, the length of a shortest path between two other nodes containing e may have changed. We terminate when no more edges are deleted, as demonstrated in the following example.

Example. Consider the NPSC instance in Figure 2.1a. The node weights are denoted in the nodes and the edge weights next to the edge name. W.l.o.g assume that the edges are picked in line 4 of **Algorithm 1** in the order $e_1, e_2, e_3, e_4, e_5, e_6$. After the first iteration of the while-loop edge e_3 is removed (Figure 2.1b), which changes the total shortest path length of the endpoints of e_1 from 4 to 5. Hence, edge e_1 becomes a bad edge and is removed in the second iteration of the while-loop (Figure 2.1c). Since no other edges are deleted **Algorithm 1** terminates with the graph shown in (Figure 2.1d)



(a) NPSC instance.

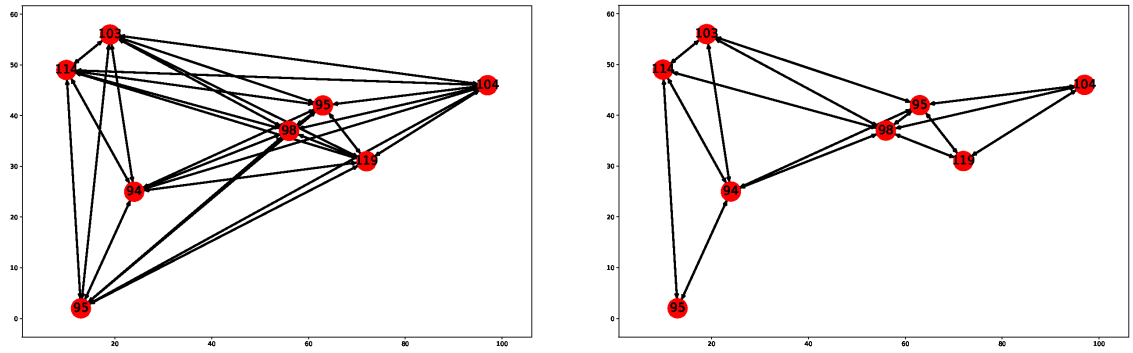
(b) First while-loop deletes e_3.

(c) Second while-loop deletes e_1.

(d) Algorithm terminates, since no more edges are deleted.

Figure 2.1.

Figure 2.2a illustrates an example of a metric-NPSC instance before the removal of bad edges and Figure 2.2b the same instance after the removal. In this example, 12 out of 28 edges were deleted. Figure 2.3 shows an optimal solution for this instance.



(a) Graph before preprocessing.

(b) Graph after preprocessing.

Figure 2.2.: Preprocessing example.

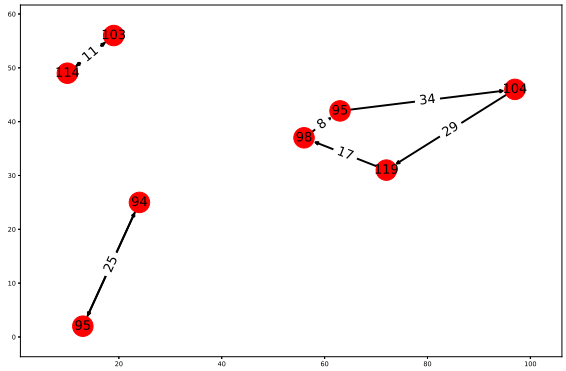


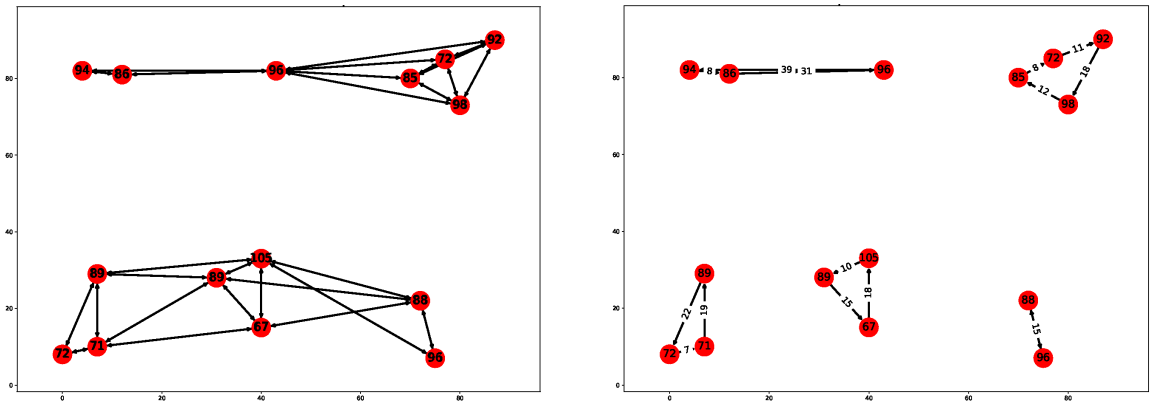
Figure 2.3.: Optimal solution.

2.2. Connectivity-based problem decomposition

Next, we describe how we can identify subproblems, whose combined optimal solutions results in an optimal solution of $\mathcal{I}$. Suppose that $G_{\mathcal{I}}$ contains multiple connected components $G_{\mathcal{I}_1}, ..., G_{\mathcal{I}_n}$. Connected components in an undirected graph can be computed in linear time using either breadth-first search (BFS) or depth-first search (DFS), see Cormen et al. [7]. Let $\mathcal{I}_1, ..., \mathcal{I}_n$ be the NPSC instances defined on the subgraphs $G_{\mathcal{I}_1}, ..., G_{\mathcal{I}_n}$, respectively. The following lemma shows that the combination of optimal solutions of $\mathcal{I}_1, ..., \mathcal{I}_n$ is an optimal solution of $\mathcal{I}$. An example is illustrated in Figure 2.4.

Lemma 2. *Let* $(\mathcal{N}_1, \mathcal{C}_1), ..., (\mathcal{N}_n, \mathcal{C}_n)$ *be optimal solutions for the instances* $\mathcal{I}_1, ..., \mathcal{I}_n$ *respectively. Then,* $(\mathcal{N}, \mathcal{C}) := (\{\mathcal{N}_1, ..., \mathcal{N}_n\}, \{\mathcal{C}_1, ..., \mathcal{C}_n\})$ *is an optimal solution of instance* $\mathcal{I}$*.*

Proof. Since $G_{\mathcal{I}_1} \cup ... \cup G_{\mathcal{I}_n} = G_{\mathcal{I}}$, and every feasible subtour of $G_{\mathcal{I}}$ contains only nodes from one connected component, $(\mathcal{N}, \mathcal{C})$ clearly is a feasible solution $\mathcal{I}$. Further, it is optimal, since otherwise at least one of the solutions $(\mathcal{N}_1, \mathcal{C}_1), ..., (\mathcal{N}_n, \mathcal{C}_n)$ would be not optimal for its corresponding subproblem, which contradicts our assumption. □



(a) Graph consisting of two connected components.

(b) Optimal solution can be found by solving the two subproblems, defined on each connected component independently.

Figure 2.4.: Example of connectivity-based problem decomposition.

2.3. A Conflict Graph for NPSC

Further preprocessing of $G_{\mathcal{I}}$ can be done by finding subsets of V, subsets of V, whose nodes cannot be contained in the same feasible subtour in any feasible solution. Recall that $s_{i,j}$ denotes the length of a shortest path between $i, j \in V$.

Definition 3 (Conflict Graph)**.** *The graph* $G_{\mathcal{I}_c} = (V, E_c)$*, where the set of edges* $E_{\mathcal{I}_c} := \{\{i, j\} \in V \times V : \ 2 \cdot s_{i,j} > \min\{T_i, T_j\}\}$*, is called the conflict graph of instance* $\mathcal{I}$*.*

Definition 4 (Bondy and Murty [4])**.** *A clique* $U \subseteq V$ *of* $G_{\mathcal{I}_c}$ *is a subset of the node set, such that every two distinct nodes are adjacent, i.e.,*

$$U \text{ is a clique of } G_{\mathcal{I}_c} \Leftrightarrow (\forall i, j \in U, \ i \neq j \implies \{i, j\} \in E_{\mathcal{I}_c})$$

A clique U *is called maximal, if it is not a subset of a larger clique.*

Lemma 3. *Let* $U \subseteq V$ *be a clique of* $G_{\mathcal{I}_c}$ *and* $(\mathcal{N}, \mathcal{C})$ *be a feasible solution of NPSC. Then the following inequalities hold,*

$$|U \cap N_k| \leq 1, \ \forall \, N_k \in \mathcal{N} \, .$$

Proof. Assume there exist nodes $i, j \in U, i \neq j$, which are both contained in the same set $N_k \in \mathcal{N}$ and therefore $|U \cap N_k| > 1$. Since $i, j \in U$, this implies that $\{i, j\} \in E_{\mathcal{I}_c}$, and thus $2 \cdot s_{i,j} > \min\{T_i, T_j\}$. Further, since $t_{i,j} + s_{i,j} \geq 2 \cdot s_{i,j} > \min\{T_i, T_j\}$, by Lemma 1 nodes i and j cannot be in the same subtour in any feasible solution, which is a contradiction. □

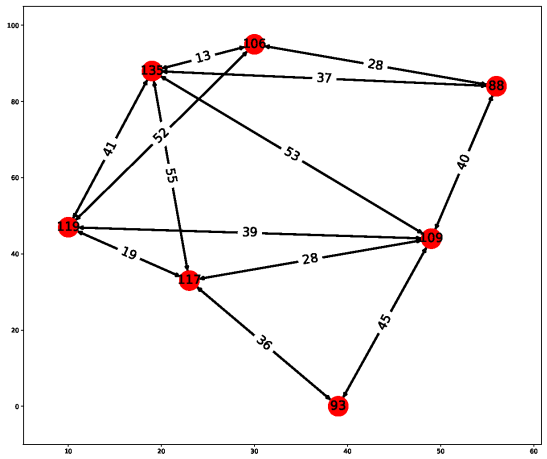
Corollary 1 (Lower bound for NPSC)**.** *Let $\mathcal{U}$ be the set of all cliques of the conflict graph $G_{\mathcal{I}_c}$ of an NPSC instance $\mathcal{I}$. A lower bound for the number of subsets in every feasible solution $(\mathcal{N}, \mathcal{C})$ is the cardinality of a maximum clique in the conflict graph, i.e.,*

$$|\mathcal{N}| \geq \max_{U \in \mathcal{U}} |U| .$$

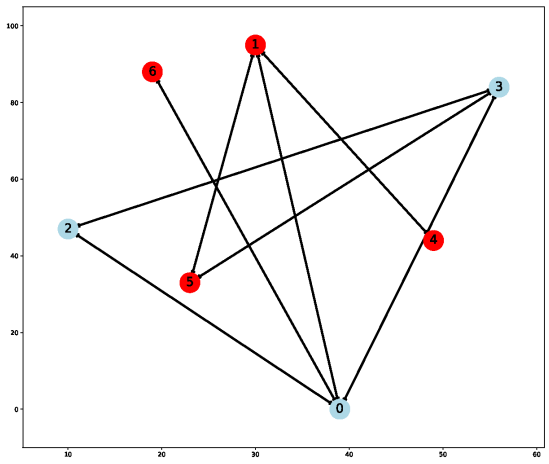
Proof. Let $U_{max} \in \mathcal{U}$ be a maximum clique in $G_{\mathcal{I}_c}$, i.e., $|U_{max}| = \max_{U \in \mathcal{U}} |U|$. From Lemma 3 it follows, that the nodes in U_{max} must be contained in disjoint sets in $\mathcal{N}$. Therefore,

$$|\mathcal{N}| \geq |U_{max}| .$$ □

An example of an NPSC instance after preprocessing, its corresponding conflict graph, and an optimal solution are shown in Figures 2.5a, 2.5b and 2.6, respectively.



(a) Graph after preprocessing.



(b) Conflict graph containing 6 maximal cliques. The unique maximum clique (light-blue nodes) has cardinality 3.

Figure 2.5.

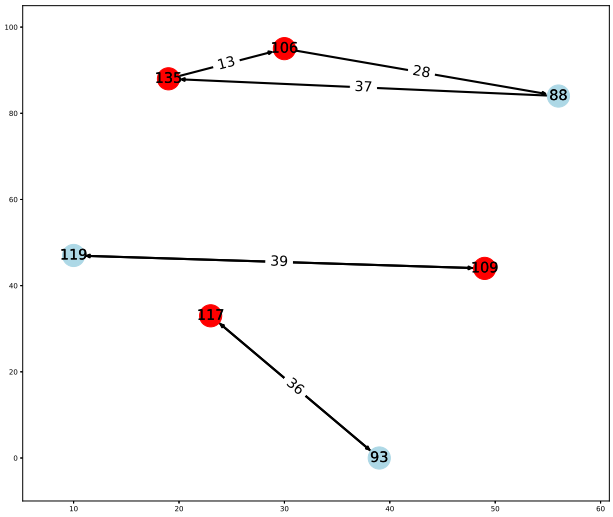


Figure 2.6.: Optimal solution consists of three subtours. Each subtour contains exactly one of the nodes of the maximum clique.

3. Insert and Reorder (IaR)-Heuristic

In the following, let $\mathcal{I} = (V, E, T, t)$ be an instance of NPSC. The maximum number of subtours in a feasible solution $(\mathcal{N}, \mathcal{C})$ is equal to $|V|$, i.e., the trivial solution contains $|V|$ subtours. Our goal is to find a non-trivial feasible solution and hence a tighter upper bound. Therefore, we will introduce a heuristic proposed by Burdakov [6]. Figure 3.1 shows the solution found by the heuristic for a large, practically intractable problem.

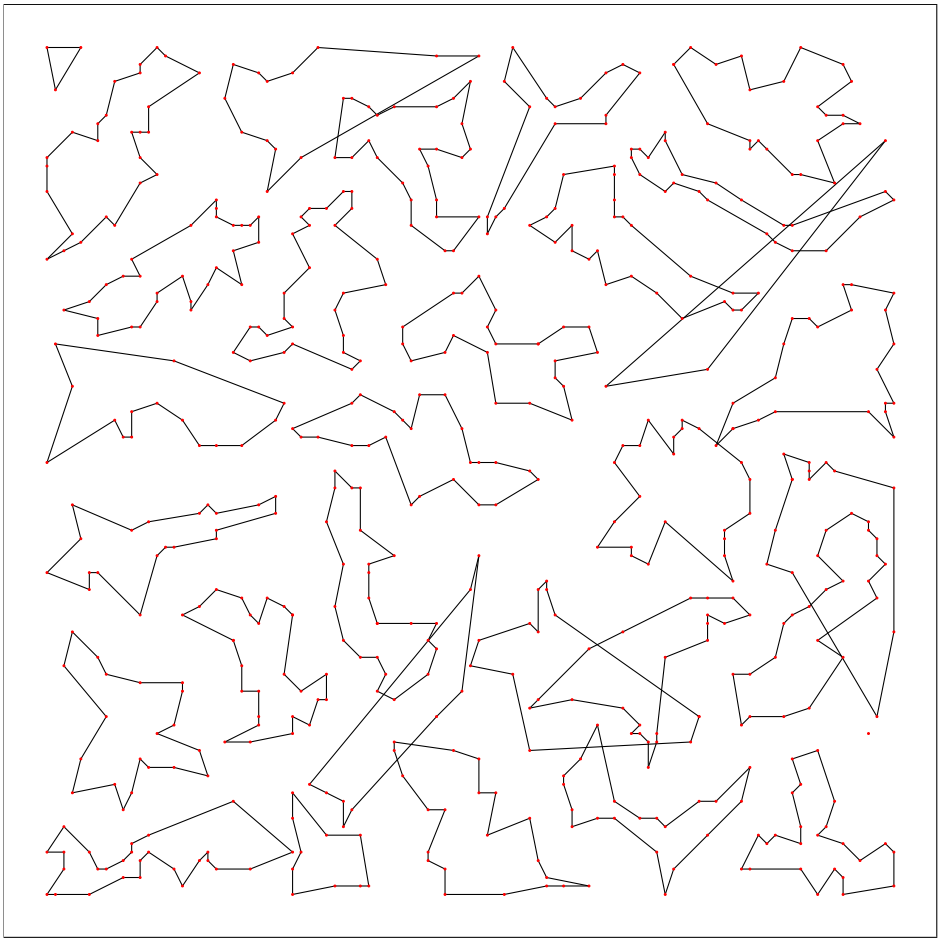


Figure 3.1.: Feasible solution found by the IaR-heuristic for a 600-node metric-NPSC test instance. This solution contains 31 subtours.

Remark. *The intersection of cycles as seen in Figure 3.1 is not an indication for a suboptimal solution. This degeneration is merely due to large critical times of the involved nodes.*

Consider for example the NPSC instance of Figure 3.2a, where all nodes have critical time equal to 6. Since no tour visiting all nodes has length less than 6, both feasible solutions shown in 3.2b and 3.2c are optimal.

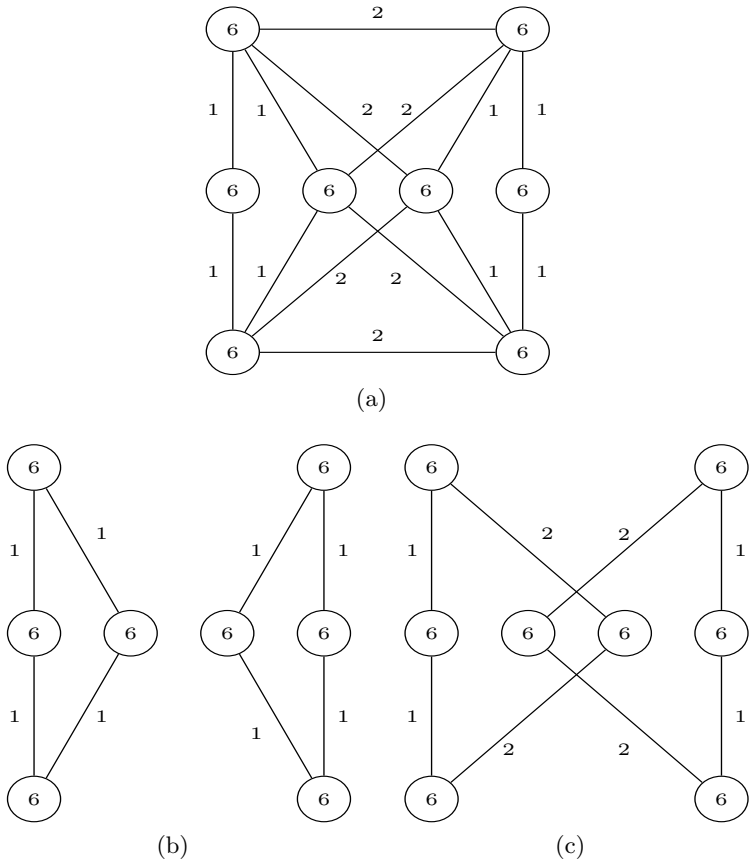


Figure 3.2.: Example to show the possibility of degenerated optimal solutions.

3.1. The IaR-Heuristic

The IaR-heuristic is inspired by the cheapest insertion heuristic (CI) for TSP, which is described by Gutin and Punnen [14]. In CI we start with a partial tour t consisting of a starting node and its nearest neighbor. Then, we repeatedly choose triples (a, b, c), where a, b are adjacent nodes in t and $c \notin t$ and we pick the triple that minimizes the increase in tour length that would occur if c was inserted between a and b. Then, for this triple (a, b, c) we insert c between a and b. This procedure continues until all nodes are inserted into t.

Our insertion heuristic aims at finding a set of feasible subtours $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ in $G_{\mathcal{I}}$ visiting all nodes. The main idea is the following:

Initially all nodes are not contained in any subtour. We denote the set of these nodes with $\tilde{V}$. We start with an empty subtour (empty list) C_k, initially with $k = 1$, and insert to C_k a node $m \in \tilde{V}$, having the smallest critical time T_m of all nodes in $\tilde{V}$. Then, we repeatedly

choose adjacent nodes $a, b \in C_k$ (at the first iteration $a = b = m$), and $c \in \tilde{V} \setminus \bigcup_{i=1}^{k} C_k$ and analogously to (CI) we pick a triple (a, b, c) that minimizes the increase in total tour length that would occur if c was inserted between a and b. The c causing this minimal increase is inserted between a and b if the new total tour length is less than T_a. After no more nodes can be inserted in C_k, we repeat the same procedure to build the next subtour C_{k+1}, and terminate when all nodes are contained in a subtour.

Optionally, after no more nodes can be inserted into a subtour C_k and if $|C_k| > 3$, we can run any exact or approximation algorithm for solving the TSP problem on the induced subgraph $G_{\mathcal{I}}[C_k]$. If a new order of the nodes results in a decrease of τ_{C_k}, we reorder the nodes in C_k and check if we can insert more nodes into this subtour. In the IaR-heuristic we denote the length of the tour found by TSP on the induced subgraph $G_{\mathcal{I}}[C_k]$ with TSP(C_k).length. Note, that solving TSP instances with exact algorithms results in a non-polynomial running time for the heuristic.

3.1.1. Excursion: An Integer Programming Model for TSP

TSP can be solved with the well known Integer Programming (IP) formulation introduced by Dantzig, Fulkerson and Johnson [8]. Let $G = (V, E)$ be an undirected graph, with edge weights $t_{ij} \geq 0,\ \forall \{i, j\} \in E$. We define for each edge $\{i, j\} \in E$ a decision variable x_{ij}, which is 1 if the edge is contained in the tour and 0 otherwise. Then TSP can be formulated as:

$$\text{minimize} \quad \sum_{(i,j)\in E} t_{ij} x_{ij}$$

subject to

$$\sum_{j \in V : \{i,j\} \in E} x_{ij} = 2 \qquad \forall i \in V \tag{2.2.1}$$

$$\sum_{i,j \in S,\ i \neq j : \{i,j\} \in E} x_{ij} \leq |S| - 1 \qquad \forall S \subset V,\ S \neq \emptyset \tag{2.2.2}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E \tag{2.2.3}$$

How this model is solved, and particularly how constraints (2.2.2) are added to the model is explained in Appendix A. Many other algorithms for TSP can be also found in Gutin and Punnen [14].

3.1.2. Pseudocode of the IaR-Heuristic

Algorithm 2 IaR Heuristic

```
1:  function IaR-Heuristic(V, E, t):
2:      Ṽ ← V                                   ▷ Nodes not visited, initially V
3:      k ← 0                                   ▷ Number of subtours, initially 0
4:      for {i, j} ∉ E do
5:          t_ij ← +∞                           ▷ set weight of non-existing edges to infinity
6:      while Ṽ ≠ ∅ do
7:          k ← k + 1                           ▷ Increase number of subtours by 1
8:          C_k, Ṽ ← SUBTOUR(k, Ṽ, E, t)
9:      return C_1, ..., C_k                    ▷ return feasible subtours C_1, ..., C_k
10:
11: function SUBTOUR(k, Ṽ, E, t):
12:     m ← argmin_{j∈Ṽ} T_j                    ▷ Find node with smallest critical time in Ṽ
13:     C_k ← [m], Ṽ ← Ṽ\{m}                    ▷ add node to C_k
14:     τ_{C_k} ← 0, expand ← TRUE
15:     while expand do
16:         Δτ ← +∞
17:         for each c ∈ Ṽ do
18:             for each a ∈ C_k do
19:                 b ← node after a in C_k
20:                 δ ← −t_ab + t_ac + t_cb
21:                 if δ < Δτ then
22:                     Δτ ← δ, c* ← c, a* ← a   ▷ c* candidate to insert in C_k
23:         if τ_{C_k} + Δτ ≤ T_m then          ▷ insert c* in C_k if the subtour is feasible
24:             insert c* in C_k after a*
25:             Ṽ ← Ṽ\{c*}, τ_k ← τ_k + Δτ
26:         else                                ▷ Check for better subtours
27:             if TSP(C_k).length < τ_{C_k} then
28:                 Reorder nodes in C_k as in the TSP tour
29:                 τ_{C_k} ← TSP(C_k).length
30:             else          ▷ if no better feasible subtour is found, end while-loop
31:                 expand = FALSE
32:     return C_k, Ṽ                           ▷ returns subtour C_k and not visited nodes Ṽ
```

Lemma 2. ***Algorithm 2*** *terminates and returns a set $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ of feasible subtours in $G_{\mathcal{I}}$, which are the subtours of a feasible solution.*

Proof. The algorithm terminates when every node $i \in \tilde{V}$ is contained in a subtour (line 6). It always terminates, since in the procedure SUBTOUR every node will enter exactly one subtour C_k at some point (line 13 and 24). Let $N_1, N_2, ..., N_k$ be the sets of nodes of the subtours $C_1, C_2, ..., C_k$, respectively. Then, condition (1.1) and (1.2) always hold for $\mathcal{N} := \{N_1, N_2, ..., N_k\}$. We still have to show that condition (1.3) holds. Nodes are inserted into a subtour C_k, only when $\tau_{C_k} + \Delta\tau \leq T_m$ (line 23, 24), where m is the node with the

smallest critical time in C_k. Hence, the constraints on the critical times (1.3) are also satisfied and therefore the IaR heuristic produces a feasible solution $(\mathcal{N}, \mathcal{C})$.

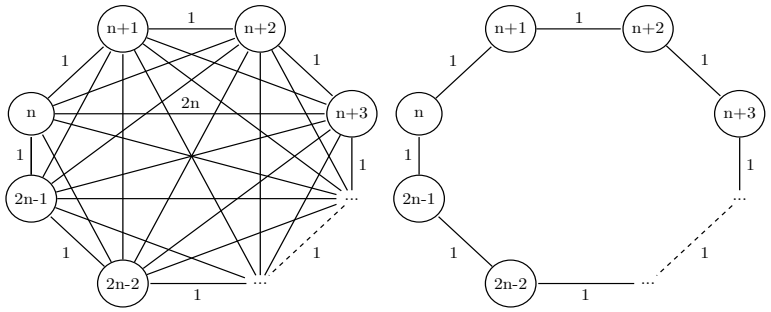
□

3.2. Approximation Error

Next, we show that the relative error of the IaR-heuristic is unbounded.

Lemma 3. ***Algorithm 2*** *has no constant approximation ratio.*

Proof. Let G be a complete undirected graph with an even number of nodes $V = \{1, ..., n\}$. We set the critical time of each node $i \in V$ to $T_i = n + i - 1$. Consider a Hamiltonian cycle $C_H = [1, 2, ..., n]$ in G, and let the edge weight be 1, for each edge $e \in C_H$. For all other edges we set a weight of $2n$ (Figure 3.3a). Obviously, $(\{V\}, \{C_H\})$ is the unique optimal solution of NPSC. (Figure 3.3b) However, the heuristic produces a solution consisting of $n/2$ subsets, each containing a pair of nodes, which are adjacent in C_H, that build a subtour (Figure 3.3c, 3.3d). Hence, the relative error of the heuristic is unbounded. □



(a) Complete graph (b) Optimal solution

(c) First possible heuristic solution (d) Second possible heuristic solution

Figure 3.3.

4. Programming Models for NPSC

In this chapter, we present programming models for solving NPSC instances. Our first model, Model A, is a Mixed Integer Nonlinear Program (MINLP). The next model, Model B, is a Mixed Integer Program (MIP), in which we linearize all nonlinear constraints of Model A. In the next MIP, Model B+, we include additional constraints such as "symmetry breaking" constraints, constraints to eliminate infeasible combinations of nodes in subsets, and valid inequalities derived from the maximal cliques in the conflict graph. Since for every subset of nodes $N_k \in \mathcal{N}$ in a solution $(\mathcal{N}, \mathcal{C})$ of an instance $\mathcal{I}$, $C_k \in \mathcal{C}$ has to be a tour in $G_{\mathcal{I}}[N_k]$, we present in Section 4.2 three types of subtour elimination constraints. In contrast to these previous models, we present another compact model, called Model MTZ, where infeasible subtours are eliminated in a similar way as in the TSP formulation proposed by Miller, Tucker and Zemlin [20].
For basic definitions and concepts of Mixed Integer Programming we refer to Achterberg [1]. A general reference on Mixed Integer Nonlinear Programming is the work of Lee and Leyffer [19]. Model building in mathematical programming is explained in the work of Williams [21].

4.1. Formulation of Models

Let $\mathcal{I} = (V, E, T, t)$ be an instance of NPSC. In the following, we consider the induced directed equivalent graph $D_{\mathcal{I}} = (V, A)$ of $G_{\mathcal{I}}$, where the set of arcs A contains for each edge $\{i, j\} \in E$, the directed arcs (i, j) and (j, i) with symmetric arc weights $t_{i,j} = t_{j,i}$.
Recall that an optimal solution has at most $|V|$ subtours, since the trivial solution has $|V|$ subtours.
A better upper bound can be obtained by the solution generated by the IaR-heuristic (Chapter 3). Set $\mathcal{K} = \{1, 2, ..., K\}$, where K is the number of subtours returned by the IaR-heuristic. Then, for each $k \in \mathcal{K}$ we define the decision variable u_k,

$$u_k = \begin{cases} 1, & \textit{if } N_k \neq \emptyset \\ 0, & \textit{otherwise.} \end{cases}$$

We call a set N_k with $u_k = 1$ in a solution active.
For each node $i \in V$ and $k \in \mathcal{K}$ we define the decision variable v_i^k,

$$v_i^k = \begin{cases} 1, & \textit{if } i \in N_k \\ 0, & \textit{otherwise,} \end{cases}$$

and for each arc $(i, j) \in A$ and each $k \in \mathcal{K}$ we define the decision variable $x_{i,j}^k$,

$$x_{i,j}^k = \begin{cases} 1, & \textit{if } C_k \textit{ contains the arc (i,j)} \\ 0, & \textit{otherwise.} \end{cases}$$

The purpose of u_k and v_i^k is to construct sets of nodes $N_k \in \mathcal{N}$ in a feasible solution, and the purpose of $x_{i,j}^k$ is to create tours $C_k \in \mathcal{C}$ in each subgraph $D_{\mathcal{I}}[N_k]$. Now we can model NPSC as the following nonlinear programming model, based on Burdakov [6]:

4.1.1. Model A

Model A

$$\min \sum_{k \in \mathcal{K}} u_k$$

s.t.

$$v_i^k \leq u_k \qquad \forall\, k \in \mathcal{K}\,, \forall\, i \in V \tag{C.1}$$

$$\sum_{k \in \mathcal{K}} v_i^k = 1 \qquad \forall\, i \in V \tag{C.2}$$

$$x_{i,j}^k \leq v_i^k \qquad \forall\, (i,j) \in A, \forall\, k \in \mathcal{K} \tag{C.3}$$

$$x_{i,j}^k \leq v_j^k \qquad \forall\, (i,j) \in A, \forall\, k \in \mathcal{K} \tag{C.4}$$

$$\sum_{(i,j) \in A} t_{i,j} x_{i,j}^k = \tau_k \qquad \forall\, k \in \mathcal{K} \tag{C.5}$$

$$T_i \geq v_i^k \tau_k \qquad \forall\, i \in V, \forall\, k \in \mathcal{K} \tag{C.6}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in V:(i,j) \in A} x_{i,j}^k = 1 \qquad \forall\, i \in V \tag{C.7}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in V:(j,i) \in A} x_{j,i}^k = 1 \qquad \forall\, i \in V \tag{C.8}$$

$$\sum_{i,j \in S_1:(i,j) \in A} x_{i,j}^k + \sum_{i,j \in S_2:(i,j) \in A} x_{i,j}^k \leq |S_1| + |S_2| - 1 \qquad \forall\, S_1, S_2 \subset V,\ S_1, S_2 \neq \emptyset,\ S_1 \cap S_2 = \emptyset,\ \forall\, k \in \mathcal{K} \tag{C.9}$$

$$u_k \in \{0,1\},\ \tau_k \in \mathbb{R}_{\geq 0} \qquad \forall\, k \in \mathcal{K} \tag{C.10}$$

$$v_i^k \in \{0,1\} \qquad \forall\, i \in V, \forall\, k \in \mathcal{K} \tag{C.11}$$

$$x_{i,j}^k \in \{0,1\} \qquad \forall\, i,j \in V, \forall\, k \in \mathcal{K} \tag{C.12}$$

The objective function minimizes the number of subsets of V in a feasible solution. Constraints (C.1) ensure that a set is active, if a node is assigned to it and constraints (C.2) guarantee that each node is contained in exactly one set $N_k \in \mathcal{N}$. Constraints (C.3) and (C.4) ensure that an arc (i,j) can only be contained in a tour C_k if and only if $i,j \in N_k$. In (C.5) we introduce a new variable τ_k that denotes the total tour length of C_k. The non-linear constraints (C.6) assure that the critical time of each node is respected. For example, let $i \in N_k$ or equivalently $v_i^k = 1$. Then (C.6) implies that $\tau_k \leq T_i$, which assures that the critical time constraint of i is satisfied. If $v_i^k = 0$ no further implication is obtained from this constraint. Equations (C.7) and (C.8) ensure that each node in a subtour has one ingoing and one outgoing arc, as it must have in a cycle. Note that for every $i \in V$, the variable $x_{i,i}^k$ is included in both sums in (C.7), (C.8), hence the arc (i,i) can also build a feasible subtour. Lastly, constraints (C.9) commonly known as subtour elimination constraints ensure that in each $D_{\mathcal{I}}[N_k]$ no multiple subtours are allowed as shown in the following lemma.

Lemma 4. *Constraints (C.9) cut off all solutions for an instance $\mathcal{I}$, that contain multiple subtours and are therefore infeasible. Further, (C.9) does not cut off any feasible solutions.*

Proof. Let $(\mathcal{N}, \mathcal{C})$ be a solution for an instance $\mathcal{I}$, that contains infeasible subtours, i.e., for some $N_k \in \mathcal{N}$, $D_{\mathcal{I}}[N_k]$ contains multiple subtours $C_{k_1}, ..., C_{k_l}$. Denote the visited nodes in each subtour by $S_{k_1}, ..., S_{k_l}$, respectively. Obviously, $S_{k_1}, ..., S_{k_l}$ are non-empty and pairwise disjoint. Hence, for all $r, s \in \{1, ..., l\}$, $r \neq s$, (C.9) implies that the following should hold,

$$\begin{aligned} |S_{k_r}| + |S_{k_s}| - 1 &\geq \sum_{i,j \in S_{k_r}:(i,j)\in A} x_{i,j}^k + \sum_{i,j \in S_{k_s}:(i,j)\in A} x_{i,j}^k \\ &\geq \sum_{(i,j)\in C_{k_r}} x_{i,j}^k + \sum_{(i,j)\in C_{k_s}} x_{i,j}^k \\ &= |C_{k_r}| + |C_{k_s}| \\ &= |S_{k_r}| + |S_{k_s}|\,, \end{aligned}$$

which is a contradiction. Hence, (C.9) cuts off the infeasible solution $(\mathcal{N}, \mathcal{C})$.
It remains to prove that (C.9) cuts off no feasible solutions. Suppose this assumption is false. Then, for a feasible solution $(\mathcal{N}, \mathcal{C})$ there exist $k \in \mathcal{K}$ and non-empty, disjoint subsets $S_1, S_2 \subset V$ such that:

$$\begin{aligned} \sum_{i,j \in S_1:(i,j)\in A} x_{i,j}^k + \sum_{i,j \in S_2:(i,j)\in A} x_{i,j}^k \quad &\geq \quad |S_1| + |S_2| \\ \Leftrightarrow \sum_{i,j \in S_1:(i,j)\in C_k} x_{i,j}^k + \sum_{i,j \in S_2:(i,j)\in C_k} x_{i,j}^k \quad &\geq \quad |S_1| + |S_2| \end{aligned}$$

From this inequality and (C.7), (C.8) it follows immediately that there exist at least two subtours in $D_{\mathcal{I}}[N_k]$, one containing only edges from $\{(i,j) \in A : i, j \in S_1\}$ and the other one containing only edges from $\{(i,j) \in A : i, j \in S_2\}$, which contradicts the feasibility of $(\mathcal{N}, \mathcal{C})$. □

The number of constraints (C.9) is of exponential size w.r.t. the number of nodes. Hence, explicitly including them into the model may lead to computational intractability. In section 4.2 we discuss a separation procedure for constraints cutting off infeasible solutions with multiple subtours.

4.1.2. Model B

Next, we derive a MIP formulation by replacing (C.6) with the following linear constraints:

$$\tau_k \leq T_i + (M - T_i)(1 - v_i^k),\ \forall\, i \in V,\ \forall\, k \in \mathcal{K}$$

where $M := \max_{i \in V} T_i$. For example, let $i \in N_k$ or equivalently $v_i^k = 1$, then it follows that $\tau_k \leq T_i$, which assures that the critical time constraint of i is satisfied. Otherwise, if $v_i^k = 0$, then $\tau_k \leq M$, which must always hold, due to the choice of M. Model B is then defined by the following MIP:

Model B

$$\min \sum_{k \in \mathcal{K}} u_k$$

s.t. (C.1) - (C.5)

(C.7) - (C.12)

$$\tau_k \leq T_i + (M - T_i)(1 - v_i^k) \quad \forall\, i \in V, \forall\, k \in \mathcal{K} \qquad \text{(C.6}')$$

4.1.3. Model B+

Symmetry Breaking Inequalities:

The solution space of NPSC is highly symmetric. Assume that an optimal solution consists of $k' < |\mathcal{K}|$ sets. Then, for each optimal solution there are $\binom{|K|}{k'}$ possibilities of choosing the indices for the active sets. By including the following "symmetry breaking" valid inequalities,

$$u_{k+1} \leq u_k \ , \forall\, k \in \{1, \ldots, K-1\}$$

we ensure that a set N_{k+1} can only be active, if the previous set N_k is also active.

Conflict Graph Inequalities:

In order to further reduce the size of the search space, we make use of the cliques in the conflict graph $G_{\mathcal{I}_c}$. Finding a maximum clique is itself an NP-hard problem, as shown by Karp [17]. In order to compute all maximal cliques of $G_{\mathcal{I}_c}$ we use the non-polynomial algorithm of Bron and Kerbosch [5] as implemented in Pythons library Networkx [15]. Let $\mathcal{U}$ be the set of all maximal cliques in $G_{\mathcal{I}_c}$. We showed in Lemma 3, that all nodes of a maximal clique must be contained in different sets in a feasible solution. This can be rephrased to the following set of valid inequalities:

$$\sum_{i \in U} v_i^k \leq 1 \ , \forall\, U \in \mathcal{U}, \forall\, k \in \mathcal{K}\,.$$

In addition, the cardinality of the largest maximal clique is a lower bound on the objective value (Corollary 1). Hence,

$$\sum_{k \in \mathcal{K}} u_k \geq \max_{U \in \mathcal{U}} |U|\,.$$

Infeasible 3-Node Subset Inequalities:

Let $s_{i,j} \in \mathbb{R}_{\geq 0}$ be the length of a shortest path between $i, j \in V$. Next, we want to detect infeasible subsets of V with three nodes that cannot be contained in any set in a feasible solution. Let $S \subseteq V$ with $|S| = 3$. Every cycle C containing the nodes of a subset $S = \{i, j, l\}$ has at least total length $\tau_C := s_{i,j} + s_{j,l} + s_{l,i}$. If $\tau_C > \min\{T_i, T_j, T_l\}$, then no subset in a feasible solution could contain all nodes of S. Hence, the following valid inequalities can be added to our model:

$$v_i^k + v_j^k + v_l^k \leq 2 \ , \forall\, i, j, l \in V \ , \forall\, k \in \mathcal{K}, \text{ if } s_{i,j} + s_{j,l} + s_{l,i} > \min\{T_i, T_j, T_l\}\,.$$

Model B+

$$\begin{aligned}
&\min \sum_{k \in \mathcal{K}} u_k \\
&\text{s.t.} \quad \text{(C.1) - (C.5)}, \\
&\qquad\qquad \text{(C.6}'\text{)}, \\
&\qquad\quad \text{(C.7) - (C.12)}, \\
&\qquad\qquad u_{k+1} \leq u_k \qquad \forall\, k \in \{1, \ldots, K-1\}, \forall\, i \in V \qquad \text{(C.13)} \\
&\qquad\qquad \sum_{k \in \mathcal{K}} u_k \geq \max_{U \in \mathcal{U}} |U| \qquad \text{(C.14)} \\
&\qquad\qquad \sum_{i \in U} v_i^k \leq 1 \qquad \forall\, U \in \mathcal{U}, \forall\, k \in \mathcal{K} \qquad \text{(C.15)} \\
&\qquad\quad v_i^k + v_j^k + v_l^k \leq 2 \qquad \forall\, i, j, l \in V, \forall\, k \in \mathcal{K}: \qquad \text{(C.16)} \\
&\qquad\qquad\qquad\qquad\quad s_{i,j} + s_{j,l} + s_{l,i} > \min\{T_i, T_j, T_l\}
\end{aligned}$$

4.2. Adding Subtour Elimination Constraints

Let $\mathcal{I} = (V, E, T, t)$ be an NPSC instance. Including constraints (C.9) to Models A, B and B+ could make them computationally intractable, because they are of exponential size w.r.t. $|V|$. Therefore, we use a sepration approach, i.e., after a solution has been found, we check whether or not constraints (C.9) are satisfied. If the solution violates (C.9) it contains multiple subtours, which are dynamically cut off. This procedure is continued until the optimization process is stopped or an optimal feasible solution is found. In the following we will present three sets of lazy constraints that can be added during optimization, whenever an infeasible solution is found. Denote

$$C_k = \{(i, j) \in A \,|\, x_{i,j}^k = 1\}\,.$$

If a cycle with arcs in C_k has less than $|N_k|$ nodes, then $D_{\mathcal{I}}[N_k]$ contains multiple subtours. The intuitive way of eliminating such infeasible solutions is by adding the following constraint to our model:

$$\sum_{(i,j)\in C_k} x^k_{i,j} \leq |C_k| - 1, \quad \forall k \in \mathcal{K}\,, \tag{SEC.1}$$

which states that for at least one arc $(i,j) \in C_k$, it should hold that $x^k_{i,j} = 0$. This constraints cuts only the current infeasible solution from the solution space. In the following we present two further sets of subtour elimination constraints.
Let $C_{k_1}, ..., C_{k_r}$ be the subtours in $D_{\mathcal{I}}[N_k]$, and $N_{k_1}, ..., N_{k_r}$ the visited nodes in each subtour, respectively. We denote $\tilde{C}_k = \{C_{k_1}, ..., C_{k_r}\}$ and $\tilde{N}_k = \{N_{k_1}, ..., N_{k_r}\}$. Then, we can add the following constraints to our model:

$$\sum_{i,j\in N_{k_l}:(i,j)\in A} x^k_{i,j} + \sum_{i,j\in N_{k_m}:(i,j)\in A} x^k_{i,j} \leq |N_{k_l}| + |N_{k_m}| - 1 \quad \begin{array}{l}\forall N_{k_l}, N_{k_m} \in \tilde{N}_k, \\ l \neq m,\ \forall k \in \mathcal{K}\end{array} \tag{SEC.2}$$

Constraint (SEC.2), are a subset of (C.9) and ensures that for each pair $N_{k_l}, N_{k_m} \in \tilde{N}_k,\ l \neq m$, no two subtours containing all nodes of N_{k_l} and N_{k_m}, respectively, are feasible, since if this happens the left hand side of (SEC.2) is equal to $|N_{k_l}| + |N_{k_m}|$.
Next, denote $A_{k_l} = \{(i,j) : i,j \in N_k, (i,j) \notin C_{k_l}\}$. Another type of subtour elimination constraints is the following:

$$\sum_{(i,j)\in C_{k_l}} x^k_{i,j} + x^k_a \leq |C_{k_l}|, \forall\, C_{k_l} \in \tilde{C}_k\ , \forall\, a \in A_{k_l}, \quad \forall k \in \mathcal{K}\,. \tag{SEC.3}$$

Constraints (SEC.3) assure that if a solution contains a subtour $C_{k_l} \in \tilde{C}_k$, i.e., if $\sum_{(i,j)\in C_{k_l}} x^k_{i,j} = |C_{k_l}|$, then $x^k_a = 0$ for all other arcs $a \in A_{k_l}$.

Next, we will present an example for the previous constraints. Assume that during the optimization of an instance $\mathcal{I}$ with Model A, B or B+, a solution $(\mathcal{N}, \mathcal{C})$ contains a set $N_k = [a,b,c,d,e,f] \in \mathcal{N}$ and the three subtours $C_{k_1} = [a,b,c]$, $C_{k_2} = [d,e]$ and $C_{k_3} = [f]$ (Figure 4.1a).

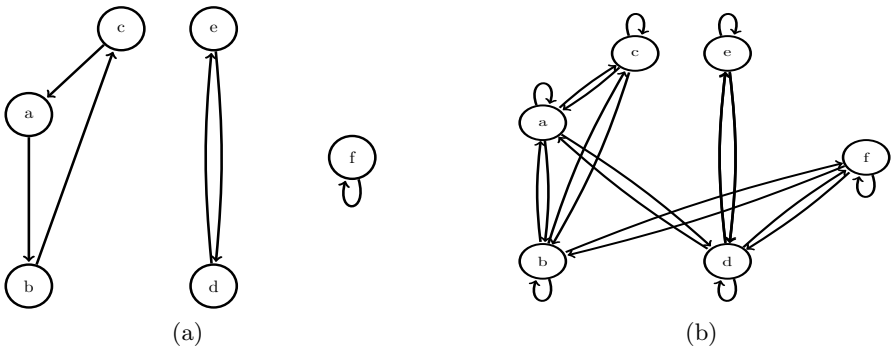


Figure 4.1.: (a) depicts the subtours in a subgraph $D_{\mathcal{I}}[N_k]$ in a found solution during the optimization process. (b) depicts the subgraph $D_{\mathcal{I}}[N_k]$ of $D_{\mathcal{I}}$.

By using (SEC.1) the following constraint is added to the model:

$$x^k_{a,b} + x^k_{b,c} + x^k_{c,a} + x^k_{d,e} + x^k_{e,d} + x^k_{f,f} \leq 5,\ \forall\, k \in \mathcal{K}\,.$$

By using (SEC.2):

$$x^k_{a,b} + x^k_{b,a} + x^k_{b,c} + x^k_{c,b} + x^k_{c,a} + x^k_{a,c} + x^k_{a,a} + x^k_{b,b} + x^k_{c,c} + x^k_{d,e} + x^k_{e,d} + x^k_{d,d} + x^k_{e,e} \leq 4, \forall\, k \in \mathcal{K}$$

$$x^k_{a,b} + x^k_{b,a} + x^k_{b,c} + x^k_{c,b} + x^k_{c,a} + x^k_{a,c} + x^k_{a,a} + x^k_{b,b} + x^k_{c,c} + x^k_{f,f} \leq 3, \forall\, k \in \mathcal{K}$$

$$x^k_{d,e} + x^k_{e,d} + x^k_{d,d} + x^k_{e,e} + x^k_{f,f} \leq 2, \forall\, k \in \mathcal{K}\,.$$

And by using (SEC.3):

$$x^k_{a,b} + x^k_{b,c} + x^k_{c,a} + x^k_{\tilde{a}} \leq 3, \forall\, \tilde{a} \in A_{k_1}, \forall\, k \in \mathcal{K}$$

$$x^k_{d,e} + x^k_{e,f} + x^k_{\tilde{a}} \leq 2, \forall\, \tilde{a} \in A_{k_2}, \forall\, k \in \mathcal{K}$$

$$x^k_{f,f} + x^k_{\tilde{a}} \leq 1, \forall\, \tilde{a} \in A_{k_3}, \forall\, k \in \mathcal{K}$$

The following algorithm summarizes the subtour elimination procedure:

Algorithm 3

function CALLBACK
  $x \leftarrow$ current MIP solution
  for $k \in \mathcal{K}$ **do**
    $C \leftarrow \{(i,j) \in A \,|\, x^k_{i,j} = 1\}$ ▷ get all arcs of subtours in $D_{\mathcal{I}}[N_k]$
    if C contains multiple subtours **then**
      model $\leftarrow$ addLazyConstraints (SEC.1, SEC.2 or SEC.3)

4.3. Model MTZ

In the following, we present a different model with a polynomial number of additional variables and constraints, which does not rely on subtour elimination constraints, i.e., we derive a compact model. Our method is based on the MTZ formulation for TSP [20]. The idea behind this formulation is to give an ordering to all nodes excluding the starting node. A straightforward use of this idea for NPSC is not possible, since different feasible solutions can contain different subsets of nodes. By "labeling" in each active set N_k a node as starting node of the subtour C_k, we are able to use a modified version of the MTZ constraints. For each node $i \in V$ and each $k \in \mathcal{K}$, we define the following binary variable:

$$s^k_i = \begin{cases} 1, & \textit{if } i \textit{ is the starting node in } N_k \\ 0, & \textit{otherwise}\,. \end{cases}$$

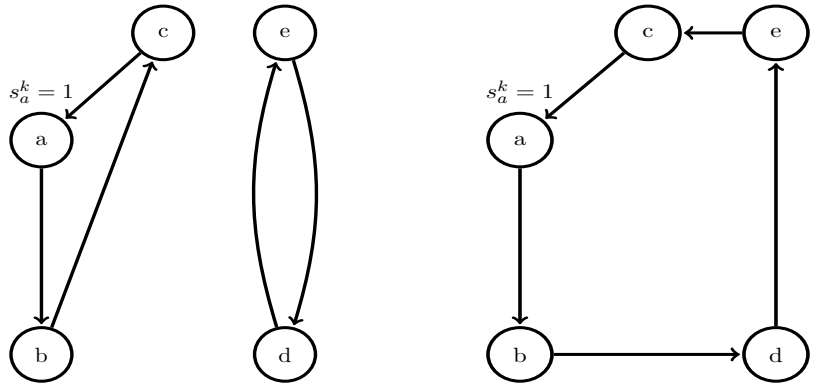
The following constraints assure that in every active set N_k exactly one node is labeled as starting node:

$$\sum_{i \in V} s^k_i = u_k, \forall\, k \in \mathcal{K}\,.$$

To exclude subtours we introduce for every $i \in V$ an additional variable $w_i \in \mathbb{R}_{\geq 0}$ and the following constraints:

$$\begin{aligned} &0 \leq w_i \leq |V| - 1 && \forall\, i \in V, \\ &w_i - w_j + |V| \cdot (x^k_{i,j} - s^k_j) \leq |V| - 1 && \forall\, i,j \in V,\ \forall\, k \in \mathcal{K}\,. \end{aligned} \tag{MTZ}$$

The (MTZ) constraints allow only one tour C_k in each $D_{\mathcal{I}}[N_k]$. Their correctness follows from the MTZ formulation of TSP in [20]. For example, consider a set $N_k = \{a, b, c, d, e\}$. In Figure 4.2a the subgraph $D_{\mathcal{I}}[N_k]$ contains two subtours $C_{k_1} = [a, b, c]$ and $C_{k_2} = [d, e]$, hence is an infeasible solution and in Figure 4.2b one feasible subtour $C_k = [a, b, d, e, c]$.



(a) Infeasible solution (b) feasible solution

Figure 4.2.

Let node a be the starting node in N_k. For the example in Figure 4.2a the MTZ constraints return for C_{k_1} the expressions $w_a + 1 \leq w_b$, $w_b + 1 \leq w_c$, $w_c - w_a \leq |V| - 1$, which can be satisfied, for example by $w_a = 0, w_b = 1, w_c = 2$. On the other hand, for the subtour $C_{k_2} = [d, e]$ we get the expressions $w_d \leq w_e + 1$ and $w_e \leq w_d + 1$, which cannot be satisfied for any $w_d, w_e \in [0, |V| - 1]$. Hence, this solution is infeasible w.r.t. MTZ constraints.
In Figure 4.2b, the MTZ constraints return for the subtour $C_k = [a, b, d, e, c]$ the expression $w_a + 1 \leq w_b$, $w_b + 1 \leq w_d$, $w_d + 1 \leq w_e$, $w_e + 1 \leq w_c$, $w_c - w_a \leq |V| - 1$, which can be satisfied, for example by $w_a = 0, w_b = 1, w_d = 2, w_e = 3, w_c = 4$.

Hence, Model MTZ is the following:

Model MTZ

$$\min \sum_{k \in \mathcal{K}} u_k$$

s.t.

$$\begin{aligned}
&\text{(C.1) - (C.5)}, \\
&\text{(C.6}'\text{)}, \\
&\text{(C.7) - (C.8)}, \\
&\text{(C.10) - (C.16)}, \\
&\sum_{i \in V} s_i^k = u_k && \forall\ k \in \mathcal{K} && \text{(C.17)} \\
&0 \leq w_i \leq |V| - 1 && \forall\ i \in V && \text{(C.18)} \\
&w_i - w_j + |V| \cdot (x_{i,j}^k - s_j^k) \leq |V| - 1 && \forall\ i, j \in V, \forall\, k \in \mathcal{K} && \text{(C.19)} \\
&w_i \in \mathbb{R}_{\geq 0} && \forall\ i \in V && \text{(C.20)} \\
&s_i^k \in \{0, 1\} && \forall\ i \in V,\ \forall\ k \in \mathcal{K} && \text{(C.21)}
\end{aligned}$$

5. Computational Experiments

In this chapter we present our computational results of testing the IaR-heuristic and our programming models on randomly generated test instances. The following software were used for our implementation and computations:

- Gurobi Optimizer 8.0 [13]
- Python libraries Numpy, Networkx.

The tests were run on a cluster of Intel Xeon 6E5-2690 2.6 GHz machines with 128 GB of RAM; a time limit of 3600 seconds was set.

5.1. Generation of Test Instances

For our computation, we create a test set of metric-NPSC test instances, by using pseudo-randomly generated numbers with python's random.seed method. The test instances are created as follows: The nodes V are pseudo-random points with coordinates $x, y \in (0, 100)$. The number of nodes n for each test instance is in $\{20, 25, 30, 35, 40\}$. The edge weights are given by the euclidean distances of the nodes. Next, in order to create appropriate node weights T_i for each node $i \in V$, the critical time vector $T = [T_1, T_2, ..., T_n] \in \mathbb{R}^n_{\geq 0}$ is randomly generated with values $T_i \in [mean\_dist \cdot low, mean\_dist \cdot high]$, where $mean\_dist$ is the average distance between two random points in V, and $low, high \geq 0$. The purpose of low and $high$ is to scale the interval of critical time values. For example , if $low = 2$, $high = 3$ and $mean\_dist = c \in R_{>0}$ then each element $T_i \in T$ gets a random value $T_i \in [2 \cdot c, 3 \cdot c]$. The test instances are named **ins _ n _ seed _ (low, high)**, with n, seed, $low, high$ as defined above.

5.2. Performance of the Insertion Heuristic

An upper bound on the number of subtours in a feasible solution is given by the IaR-heuristic in Chapter 3. Its solution serves as a start feasible solution for all programming models. In the IaR-heuristic, the TSP instances are solved by the IP presented in section 3.1.1. Tables 5.1, 5.2 and 5.3 show that the heuristic finds good feasible solutions for our test instances with $20, 25, 30, 35, 40$ in a very short amount of time. The GAP is computed as

$$\frac{|LB - IaR|}{|IaR|},$$

where the LB is the best lower bound on the optimal solution found by any programming model before terminating the solving process and IaR is the value of the heuristic solution. Many larger test instances, mostly with 35 and 40 nodes, were not solved to optimality by any model, hence the heuristic GAP may actually be even smaller. Together with the increasing size of the instances, this explains the increasing average GAP shown in Figure 5.1.

Table 5.1.: Heuristic solution for instances with $n \in \{20, 25\}$.

	$n = 20$				$n = 25$			
instances	IaR sol.	best bound	GAP (%)	time (s)	IaR sol.	best bound	GAP (%)	time (s)
ins_n_0_(2.0,3.0)	5	5	0.0%	0.1	6	5	16.7%	0.1
ins_n_1_(2.1,3.3)	4	4	0.0%	0.1	4	4	0.0%	0.1
ins_n_2_(2.2,3.6)	5	4	20.0%	0.1	6	5	16.7%	0.1
ins_n_3_(2.3,3.9)	5	4	20.0%	0.1	5	5	0.0%	0.1
ins_n_4_(2.4,4.2)	5	4	20.0%	0.1	6	5	16.7%	0.1
ins_n_5_(2.5,4.5)	4	3	25.0%	0.1	4	4	0.0%	0.1
ins_n_6_(2.6,4.8)	4	4	0.0%	0.1	4	4	0.0%	0.1
ins_n_7_(2.7,5.1)	4	4	0.0%	0.1	5	4	20.0%	0.1
ins_n_8_(2.8,5.4)	3	3	0.0%	0.1	3	3	0.0%	0.1
ins_n_9_(2.9,5.7)	4	3	25.0%	0.1	4	3	25.0%	0.1

Table 5.2.: Heuristic solution for instances with $n \in \{30, 35\}$.

	$n = 30$				$n = 35$			
instances	IaR sol.	best bound	GAP (%)	time (s)	IaR sol.	best bound	GAP (%)	time (s)
ins_n_0_(2.0,3.0)	7	6	14.3%	0.1	8	6	25.0%	0.1
ins_n_1_(2.1,3.3)	5	5	0.0%	0.1	5	5	0.0 %	0.1
ins_n_2_(2.2,3.6)	5	5	0.0%	0.1	6	5	16.7%	0.1
ins_n_3_(2.3,3.9)	6	5	16.7%	0.1	7	5	28.6%	0.1
ins_n_4_(2.4,4.2)	7	5	28.6%	0.1	7	5	28.6%	0.1
ins_n_5_(2.5,4.5)	4	4	0.0%	0.1	4	4	0.0 %	0.1
ins_n_6_(2.6,4.8)	4	4	0.0%	0.1	4	4	0.0 %	0.1
ins_n_7_(2.7,5.1)	5	4	20.0%	0.1	5	4	20.0%	0.1
ins_n_8_(2.8,5.4)	3	3	0.0%	0.1	4	3	25.0%	0.1
ins_n_9_(2.9,5.7)	4	3	25.0%	0.1	4	3	25.0%	0.1

Table 5.3.: Heuristic solution for instances with $n = 40$.

	$n = 40$			
instances	IaR sol.	best bound	GAP (%)	time (s)
ins_n_0_(2.0,3.0)	9	6	33.3%	0.1
ins_n_1_(2.1,3.3)	6	5	16.7%	0.1
ins_n_2_(2.2,3.6)	6	5	16.7%	0.1
ins_n_3_(2.3,3.9)	7	5	28.6%	0.1
ins_n_4_(2.4,4.2)	7	5	28.6%	0.1
ins_n_5_(2.5,4.5)	5	4	20.0%	0.1
ins_n_6_(2.6,4.8)	5	4	20.0%	0.1
ins_n_7_(2.7,5.1)	5	4	20.0%	0.1
ins_n_8_(2.8,5.4)	4	3	25.0%	0.1
ins_n_9_(2.9,5.7)	5	3	40.0%	0.1

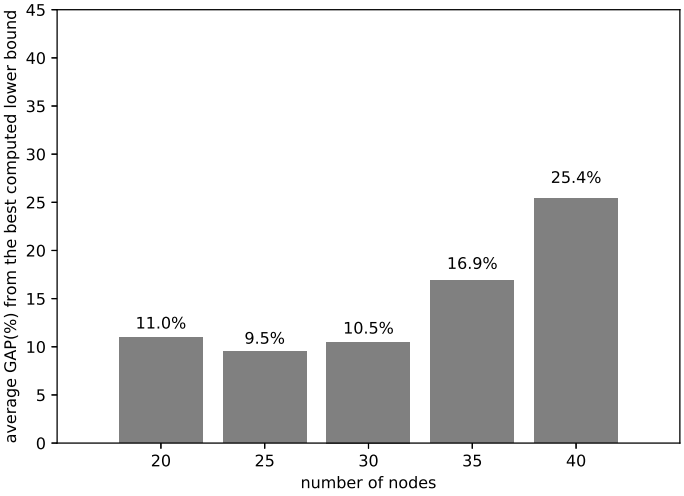


Figure 5.1.: Average GAP of the IaR-heuristic solution from the best lower bound computed within an hour of computation by any model.

5.3. Performance of the MI(NL)P Models

Our computational results are summarized in the tables below, which consists of the following columns:

- The first column states the names of the test instance.
- Vars/Cons denote the number of variables/constraints of the test instance.

- SEC denotes the number of subtour elimination constraints added during optimization.
- NodeCnt is the number of branch-and-cut nodes explored during optimization.
- UB is the objective value for the current solution, i.e., the best solution found before terminating the optimization process.
- LB is the best found lower bound on the optimal solution.
- GAP is the relative optimality gap, computed as

$$\frac{|LB - UB|}{|UB|}.$$

- Time(s) is the runtime of the current optimization (in seconds).

First, we compare the performance of our models on test instances with 20 nodes. In the following tables Model A, B, B+ use the subtour elimination constraints SEC.3. A comparison of SEC.1, SEC.2 and SEC.3 is presented in section 5.4. As shown in Table 5.4, Model A failed to solve any instance within our time limit of 3600 seconds. Model B proved for all instances a better optimality gap (Table 5.5). Nevertheless, only two out of ten instances were solved to optimality. Thus, no further computational results with these two models are shown. Results for Model A and B using SEC.1 and SEC.2 can be found in Appendix B.

Table 5.4.: Solutions using Model A with SEC.3
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	2460	125054	5	2	60.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	8036	183380	4	2	50.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	9535	84246	5	2	60.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	9076	126849	5	1	80.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	7400	92250	5	2	60.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	2038	94348	4	1	75.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3072	8809	388410	4	1	75.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	7476	78188	4	1	75.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2391	8157	121283	3	1	66.7%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	4784	48525	4	2	50.0%	3600.0

Table 5.5.: Solutions using Model B with SEC.3
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	67605	339879	5	3	40.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	27652	1228992	4	3	25.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	65405	305166	4	3	25.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	74455	155225	4	3	25.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	75425	267383	5	3	40.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	85876	102221	3	3	0.0%	1654.9
ins_20_6_(2.6,4.8)	1508/3072	85124	211033	4	3	25.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	60976	406159	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2391	49287	302501	3	2	33.3%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	38204	15777	3	3	0.0%	173.5

On the other hand, Model B+ and Model MTZ solved all 10 instances to optimality, as shown in Table 5.6 and 5.7, respectively. Model B+ solved each instance after at most 48.5 seconds and 15254 explored nodes. For Model MTZ the maximum solving time is 1793.6 seconds and the largest number of explored nodes is 5002260. We observe that for some instances (ins_20_6_(2.6,4.8) and ins_20_7_(2.7,5.1)) Model MTZ needs a lot more time than for all other instances. Model B+ solves those instances in 7.8 and 33.9 seconds, respectively.

Table 5.6.: Solutions using Model B+ with SEC.3
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3450	11365	4465	5	5	0.0%	8.3
ins_20_1_(2.1,3.3)	1036/2684	0	0	4	4	0.0%	0.2
ins_20_2_(2.2,3.6)	1615/4415	19610	894	4	4	0.0%	23.3
ins_20_3_(2.3,3.9)	1585/4215	13895	1213	4	4	0.0%	3.6
ins_20_4_(2.4,4.2)	1805/4685	15305	1357	4	4	0.0%	4.6
ins_20_5_(2.5,4.5)	1436/3840	16452	213	3	3	0.0%	2.9
ins_20_6_(2.6,4.8)	1508/3764	21188	1644	4	4	0.0%	7.8
ins_20_7_(2.7,5.1)	1532/3740	18960	15254	4	4	0.0%	33.9
ins_20_8_(2.8,5.4)	1167/2784	0	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1580/3712	60164	10958	3	3	0.0%	48.5

Table 5.7.: Solutions using Model MTZ
for test instances with 20 nodes

	Vars/Cons	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1445/4675	182270	5	5	0.0%	99.0
ins_20_1_(2.1,3.3)	1136/3640	0	4	4	0.0%	0.1
ins_20_2_(2.2,3.6)	1735/5930	18347	4	4	0.0%	15.7
ins_20_3_(2.3,3.9)	1705/5700	118	4	4	0.0%	13.5
ins_20_4_(2.4,4.2)	1925/6390	1877	4	4	0.0%	6.2
ins_20_5_(2.5,4.5)	1536/5196	13712	3	3	0.0%	10.9
ins_20_6_(2.6,4.8)	1608/5192	974956	4	4	0.0%	1793.6
ins_20_7_(2.7,5.1)	1632/5192	5002260	4	4	0.0%	1542.7
ins_20_8_(2.8,5.4)	1247/3891	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1680/5212	8599	3	3	0.0%	5.2

Next we solve with Model B+ and Model MTZ test instances with $n = 25$. Our computational results are presented in Tables 5.8 and 5.9, respectively. Model B+ solved all ten instances within our time limit. On the other hand, Model MTZ failed to prove optimality for three of them. These instances were solved with Model B+ after 81.7, 97.1 and 1213.1 seconds, respectively.

Table 5.8.: Solutions using Model B+ with SEC.3
for test instances with 25 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_25_0_(2.0,3.0)	2502/7035	26442	702	5	5	0.0%	42.2
ins_25_1_(2.1,3.3)	1620/4527	0	0	4	4	0.0%	0.2
ins_25_2_(2.2,3.6)	2982/8661	37092	21375	5	5	0.0%	81.7
ins_25_3_(2.3,3.9)	2475/7095	31220	85200	5	5	0.0%	97.1
ins_25_4_(2.4,4.2)	3282/8631	91344	35026	5	5	0.0%	1213.1
ins_25_5_(2.5,4.5)	2300/6583	11244	148	4	4	0.0%	36.6
ins_25_6_(2.6,4.8)	2292/6279	18640	1669	4	4	0.0%	12.2
ins_25_7_(2.7,5.1)	2915/7660	64795	4688	4	4	0.0%	141.4
ins_25_8_(2.8,5.4)	1827/4659	0	0	3	3	0.0%	0.4
ins_25_9_(2.9,5.7)	2476/6063	72784	2513	3	3	0.0%	44.4

Table 5.9.: Solutions using Model MTZ
for test instances with 25 nodes

	Vars/Cons	NodeCnt	UB	LB	GAP	Time(s)
ins_25_0_(2.0,3.0)	2677/9387	100	5	5	0.0%	38.9
ins_25_1_(2.1,3.3)	1745/6047	0	4	4	0.0%	0.2
ins_25_2_(2.2,3.6)	3157/11493	9463396	5	4	20.0%	3600.0
ins_25_3_(2.3,3.9)	2625/9445	6514111	5	4	20.0%	3600.0
ins_25_4_(2.4,4.2)	3457/11763	9769377	5	4	20.0%	3600.0
ins_25_5_(2.5,4.5)	2425/8783	13829	4	4	0.0%	20.3
ins_25_6_(2.6,4.8)	2417/8471	79663	4	4	0.0%	41.1
ins_25_7_(2.7,5.1)	3065/10450	296257	4	4	0.0%	202.2
ins_25_8_(2.8,5.4)	1927/6411	0	3	3	0.0%	0.4
ins_25_9_(2.9,5.7)	2601/8439	2048818	3	3	0.0%	566.2

Next, we solve larger instances with $n = 30$. As summarized in Table 5.10 by using Model B+ nine out of ten instances are solved to optimality. For the last one (ins_30_9_(2.9,5.7)) an optimality gap of 25% was proven. We notice that for five instances the solution generated by the heuristic is optimal. These instances are solved with Model B+ after at most 129.1 seconds. The solving time of the rest optimally solved instances is in general much higher and varies between 245.6 and 2009.8 seconds. Model MTZ solved eight out of ten instances to optimality (Table 5.11). Interestingly for instance ins_30_2_(2.1,3.3) no optimality was proven, even though the heuristic solution is optimal.

Table 5.10.: Solutions using Model B+ with SEC.3
for test instances with 30 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_30_0_(2.0,3.0)	4137/12655	64113	4457	6	6	0.0%	2009.8
ins_30_1_(2.1,3.3)	2925/9010	0	38	5	5	0.0%	11.3
ins_30_2_(2.2,3.6)	3675/11145	17050	16875	5	5	0.0%	129.1
ins_30_3_(2.3,3.9)	4266/12720	77934	14674	5	5	0.0%	877.3
ins_30_4_(2.4,4.2)	5411/14825	123081	5644	5	5	0.0%	1377.9
ins_30_5_(2.5,4.5)	3268/9914	0	37	4	4	0.0%	34.7
ins_30_6_(2.6,4.8)	3276/9738	37796	1445	4	4	0.0%	40.1
ins_30_7_(2.7,5.1)	4155/11875	101045	31943	4	4	0.0%	245.6
ins_30_8_(2.8,5.4)	2619/7056	0	0	3	3	0.0%	0.6
ins_30_9_(2.9,5.7)	3540/9226	124860	457946	4	3	25.0%	3600.0

Table 5.11.: Solutions using Model MTZ
for test instances with 30 nodes

	Vars/Cons	NodeCnt	UB	LB	GAP	Time(s)
ins_30_0_(2.0,3.0)	4377/16582	4066698	6	6	0.0%	2289.9
ins_30_1_(2.1,3.3)	3105/11785	11	5	5	0.0%	20.8
ins_30_2_(2.2,3.6)	3855/14670	5078386	5	4	20.0%	3600.0
ins_30_3_(2.3,3.9)	4476/16806	15495	5	5	0.0%	187.6
ins_30_4_(2.4,4.2)	5651/20026	1225406	5	5	0.0%	1762.3
ins_30_5_(2.5,4.5)	3418/13062	119	4	4	0.0%	70.2
ins_30_6_(2.6,4.8)	3426/12894	494171	4	4	0.0%	278.6
ins_30_7_(2.7,5.1)	4335/15880	106925	4	4	0.0%	195.1
ins_30_8_(2.8,5.4)	2739/9585	0	3	3	0.0%	0.7
ins_30_9_(2.9,5.7)	3690/12646	1665240	4	3	25.0%	3600.0

Since our models solve most test instances with $n = 30$, we continue with instances with $n = 35$. Model B+ solved six out of ten instances to optimality (Table 5.12), while Model MTZ only three (Table 5.13). All optimally solved instances are solved faster with Model B+ and in general much less nodes are explored with Model B+. However, for the instance ins_35_4_(2.4,4.2) Model MTZ proved a better optimality gap. Further, we notice that the three solved instances with Model MTZ, are the ones for which the heuristic produced an optimal solution. Both models solved these instance after at most 44.3 seconds, which indicates that a good quality heuristic solution speeds up the optimization process.

Table 5.12.: Solutions using Model B+ with SEC.3
for test instances with 35 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_35_0_(2.0,3.0)	6424/21073	128648	14694	7	5	28.6%	3600.0
ins_35_1_(2.1,3.3)	3965/13030	0	56	5	5	0.0%	23.4
ins_35_2_(2.2,3.6)	5790/18141	132936	92523	5	5	0.0%	1495.8
ins_35_3_(2.3,3.9)	6769/21329	132328	93408	5	5	0.0%	1512.2
ins_35_4_(2.4,4.2)	7329/20552	137921	6897	7	4	42.9%	3600.0
ins_35_5_(2.5,4.5)	4444/14745	0	34	4	4	0.0%	12.8
ins_35_6_(2.6,4.8)	4516/14573	18740	115	4	4	0.0%	20.5
ins_35_7_(2.7,5.1)	5725/17715	113955	219818	5	4	20.0%	3600.0
ins_35_8_(2.8,5.4)	4716/13193	170572	5083	3	3	0.0%	132.6
ins_35_9_(2.9,5.7)	4940/15485	131216	110084	4	3	25.0%	3600.0

Table 5.13.: Solutions using Model MTZ
for test instances with 35 nodes

	Vars/Cons	NodeCnt	UB	LB	GAP	Time(s)
ins_35_0_(2.0,3.0)	6739/27217	86968	7	5	28.6%	3600.0
ins_35_1_(2.1,3.3)	4175/16820	48	5	5	0.0%	44.3
ins_35_2_(2.2,3.6)	6035/23721	1554516	6	5	16.7%	3600.0
ins_35_3_(2.3,3.9)	7049/27853	1051661	6	5	16.7%	3600.6
ins_35_4_(2.4,4.2)	7609/27636	3893108	6	5	16.7%	3600.0
ins_35_5_(2.5,4.5)	4619/19049	27	4	4	0.0%	23.5
ins_35_6_(2.6,4.8)	4691/18949	1385	4	4	0.0%	35.1
ins_35_7_(2.7,5.1)	5935/23265	5036241	5	4	20.0%	3600.0
ins_35_8_(2.8,5.4)	4891/17769	2117674	4	3	25.0%	3600.0
ins_35_9_(2.9,5.7)	5115/20285	4013750	4	3	25.0%	3600.0

Lastly for $n = 40$, no instance was solved to optimality within our time limit (Table 5.14, 5.15) For two out of ten instances a better optimality gap was found with Model MTZ and for one instance with Model B+. For the rest the solutions with both models have the same optimality gap.

Table 5.14.: Solutions using Model B+ with SEC.3
for test instances with 40 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_40_0_(2.0,3.0)	9459/31755	183186	12181	9	5	44.4%	3600.0
ins_40_1_(2.1,3.3)	6282/21636	106626	333820	6	5	16.7%	3600.0
ins_40_2_(2.2,3.6)	7470/24714	119478	7966	6	5	16.7%	3600.0
ins_40_3_(2.3,3.9)	8855/29086	124922	3068	7	4	42.9%	3600.0
ins_40_4_(2.4,4.2)	9359/27301	231721	35088	7	5	28.6%	3600.0
ins_40_5_(2.5,4.5)	7155/24310	146385	185989	5	4	20.0%	3600.0
ins_40_6_(2.6,4.8)	7295/24025	175795	160354	5	4	20.0%	3600.0
ins_40_7_(2.7,5.1)	7475/24005	117625	103392	5	4	20.0%	3600.0
ins_40_8_(2.8,5.4)	6076/18108	194712	102676	4	3	25.0%	3600.0
ins_40_9_(2.9,5.7)	7735/22370	150985	2793	5	3	40.0%	3600.0

Table 5.15.: Solutions using Model MTZ
for test instances with 40 nodes

	Vars/Cons	NodeCnt	UB	LB	GAP	Time(s)
ins_40_0_(2.0,3.0)	9859/40854	56486	9	5	44.4%	3600.0
ins_40_1_(2.1,3.3)	6562/27678	2830969	6	5	16.7%	3600.0
ins_40_2_(2.2,3.6)	7750/31944	217192	6	4	33.3%	3600.0
ins_40_3_(2.3,3.9)	9175/37661	631299	6	5	16.7%	3600.0
ins_40_4_(2.4,4.2)	9679/36380	702004	6	5	16.7%	3600.0
ins_40_5_(2.5,4.5)	7395/31265	2269265	5	4	20.0%	3600.0
ins_40_6_(2.6,4.8)	7535/31120	1429439	5	4	20.0%	3600.0
ins_40_7_(2.7,5.1)	7715/31280	1455413	5	4	20.0%	3600.0
ins_40_8_(2.8,5.4)	6276/24024	1396860	4	3	25.0%	3600.0
ins_40_9_(2.9,5.7)	7975/29905	779146	5	3	40.0%	3600.0

After testing all models we can conclude that Model B+ is more reliable than Model A, B and MTZ on solving instances with 20, 25, 30, 35 nodes. Model A and B cannot solve most instances with 20 nodes. Model MTZ ourperforms Model A and B, but in general it is much slower than Model B+ and fails to solve three instances with 25 nodes, one instance with 30 nodes and three instances with 35 nodes, which are solved by Model B+.

5.4. Performance of the Subtour Elimination Constraints

In this section we compare the performance of Model B+ with the subtour elimination constraints SEC.1, SEC.2 and SEC.3 for test instances with $n \in 25, 40$. For $n = 25$ only Model B+ with SEC.3 solved all instances to optimality (Table 5.8). By using SEC.1 only five out of ten are solved to optimality (Table 5.16) and by using SEC.2 nine out ten (Table 5.17). We observe that by using SEC.1 and SEC.2 the number of subtour elimination constraints added during optimization is on average much lower, especially by using SEC.2. Furthermore, with Model B+ much less nodes are explored.

Table 5.16.: Solutions using Model B+ with SEC.1
for test instances with 25 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_25_0_(2.0,3.0)	2502/7035	8496	17447	5	5	0.0%	38.7
ins_25_1_(2.1,3.3)	1620/4527	0	0	4	4	0.0%	0.2
ins_25_2_(2.2,3.6)	2982/8661	20154	3438284	5	4	20.0%	3600.0
ins_25_3_(2.3,3.9)	2475/7095	19935	7943684	5	4	20.0%	3600.0
ins_25_4_(2.4,4.2)	3282/8631	32280	3682375	5	4	20.0%	3600.0
ins_25_5_(2.5,4.5)	2300/6583	240	1790	4	4	0.0%	20.9
ins_25_6_(2.6,4.8)	2292/6279	1964	23579	4	4	0.0%	17.5
ins_25_7_(2.7,5.1)	2915/7660	91430	500910	5	3	40.0%	3600.0
ins_25_8_(2.8,5.4)	1827/4659	0	0	3	3	0.0%	0.4
ins_25_9_(2.9,5.7)	2476/6063	83164	1166163	4	3	25.0%	3600.0

Table 5.17.: Solutions using Model B+ with SEC.2
for test instances with 25 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_25_0_(2.0,3.0)	2502/7035	2718	4839	5	5	0.0%	10.4
ins_25_1_(2.1,3.3)	1620/4527	0	0	4	4	0.0%	0.2
ins_25_2_(2.2,3.6)	2982/8661	6456	1445526	5	5	0.0%	1726.4
ins_25_3_(2.3,3.9)	2475/7095	3250	11488927	5	4	20.0%	3600.0
ins_25_4_(2.4,4.2)	3282/8631	11472	1869403	5	5	0.0%	959.6
ins_25_5_(2.5,4.5)	2300/6583	340	220	4	4	0.0%	15.5
ins_25_6_(2.6,4.8)	2292/6279	668	11804	4	4	0.0%	11.5
ins_25_7_(2.7,5.1)	2915/7660	18965	103759	4	4	0.0%	582.2
ins_25_8_(2.8,5.4)	1827/4659	0	0	3	3	0.0%	0.7
ins_25_9_(2.9,5.7)	2476/6063	7108	98779	3	3	0.0%	54.0

For larger instances with $n = 40$ no instance was solved to optimality. We observe that with SEC.3 (Table 5.14) a much larger number of subtour elimination constraints is added,

compared to SEC.1 (Table 5.18) and SEC.2 (Table 5.19). With SEC.1 and SEC.2 a larger number of nodes is explored. Furthermore, SEC.2 proves a better gap for two instances compared to SEC.3 and a better for one instance compared to SEC.1. A conclusion for the best performing subtour elimination constraints cannot be made. Using Model B+ with SEC.3 seems to perform better for smaller test instances, since more are solved compared to SEC.1 and SEC.2. For larger instances however, the large amount of subtour elimination constraints SEC.3 seems to harm the performance of the Model B+. Further computation results for Model B+ with SEC.1 and SEC.2 can be found in Appendix B.

Table 5.18.: Solutions using Model B+ with SEC.1
for test instances with 40 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_40_0_(2.0,3.0)	9459/31755	41364	116658	9	6	33.3%	3600.0
ins_40_1_(2.1,3.3)	6282/21636	23478	1400228	6	5	16.7%	3600.0
ins_40_2_(2.2,3.6)	7470/24714	35040	165170	6	4	33.3%	3600.0
ins_40_3_(2.3,3.9)	8855/29086	56273	267806	7	5	28.6%	3600.0
ins_40_4_(2.4,4.2)	9359/27301	100212	395854	7	5	28.6%	3600.0
ins_40_5_(2.5,4.5)	7155/24310	68560	1033488	5	4	20.0%	3600.0
ins_40_6_(2.6,4.8)	7295/24025	64820	1243703	5	4	20.0%	3600.0
ins_40_7_(2.7,5.1)	7475/24005	53815	1093971	5	4	20.0%	3600.0
ins_40_8_(2.8,5.4)	6076/18108	107420	602595	4	3	25.0%	3600.0
ins_40_9_(2.9,5.7)	7735/22370	96235	389962	5	3	40.0%	3600.0

Table 5.19.: Solutions using Model B+ with SEC.2
for test instances with 40 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_40_0_(2.0,3.0)	9459/31755	41886	208845	9	6	33.3%	3600.0
ins_40_1_(2.1,3.3)	6282/21636	5658	1298621	6	5	16.7%	3600.0
ins_40_2_(2.2,3.6)	7470/24714	11544	103392	6	5	16.7%	3600.0
ins_40_3_(2.3,3.9)	8855/29086	21462	122161	7	5	28.6%	3600.0
ins_40_4_(2.4,4.2)	9359/27301	32914	397799	7	5	28.6%	3600.0
ins_40_5_(2.5,4.5)	7155/24310	10935	1821021	5	4	20.0%	3600.0
ins_40_6_(2.6,4.8)	7295/24025	12975	1252476	5	4	20.0%	3600.0
ins_40_7_(2.7,5.1)	7475/24005	13500	1393356	5	4	20.0%	3600.0
ins_40_8_(2.8,5.4)	6076/18108	6624	3091542	4	3	25.0%	3600.0
ins_40_9_(2.9,5.7)	7735/22370	29960	499825	5	3	40.0%	3600.0

5.5. Performance of the Valid Inequalities (C.13) - (C.16)

Since Model B+ outperforms Model A and B, we examine which of the valid inequalities (C.13) - (C.16) contribute at most to this improvement. Therefore, we solve the test instances with $n = 20$ with Model B and the addition of (C.13) - (C.16) separately. In Table 5.20 we add the symmetry breaking inequalities (C.13) to Model B. In Table 5.21 we add the lower bound valid inequality (C.14). In Table 5.22 we add the inequalities (C.15) derived from the conflict graph and in Table 5.23 we add the 3-node infeasible subset inequalities (C.16). Surprisingly, in comparison with Model B (Table 5.5) the addition of (C.13) negatively affects the optimization process as shown in Table 5.20. No instance is solved to optimality and the optimality gap of one of the unsolved instances gets worse.

Table 5.20.: Solutions using Model B with SEC.3 and valid inequalities (C.13) for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2709	68115	278679	5	3	40.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2131	26748	1112376	4	3	25.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3229	91095	245224	4	3	25.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3669	76795	183690	5	3	40.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2931	78992	278364	4	3	25.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3075	74240	510190	4	3	25.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3123	63228	467315	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2393	44649	1843004	3	2	33.3%	3600.0
ins_20_9_(2.9,5.7)	1580/3219	102736	340812	4	3	25.0%	3600.0

By including the lower bound (C.14) in Model B, four instances are solved to optimality and the gap of ins_20_0_(2.0,3.0) improves as shown in Table 5.21. With Model B however, only two instances are solved to optimality.

Table 5.21.: Solutions using Model B with SEC.3 and valid inequalities (C.14) for test instances with 20 nodes.

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2706	62480	333366	5	4	20.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2129	0	0	4	4	0.0%	0.2
ins_20_2_(2.2,3.6)	1615/3286	1610	763047	4	3	25.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3226	9000	602146	4	3	25.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3666	83080	124601	5	3	40.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2929	73212	22324	3	3	0.0%	351.4
ins_20_6_(2.6,4.8)	1508/3073	20616	634817	4	3	25.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3121	30472	466102	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2392	0	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1580/3217	64272	17774	3	3	0.0%	300.9

Next, compared to Table 5.5, the solutions in both Tables 5.22 and 5.23 show a significant improvement. For both variants seven out of ten instances are solved to optimality. On the other hand, with Model B only two out of ten instances are optimally solved. This means that the addition of the 3-node infeasible subsets inequalities and those derived from the cliques in the conflict graph contribute at most to the improved performance of Model B+ and MTZ in comparison to Model A and B.

Table 5.22.: Solutions using Model B with SEC.3 and valid inequalities (C.15) for test instances with 20 nodes.

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3026	27330	2139668	5	4	20.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2401	0	0	4	4	0.0%	0.1
ins_20_2_(2.2,3.6)	1615/3506	38820	20354	4	4	0.0%	250.1
ins_20_3_(2.3,3.9)	1585/3451	25960	49684	4	4	0.0%	256.1
ins_20_4_(2.4,4.2)	1805/3786	47515	44241	4	4	0.0%	274.3
ins_20_5_(2.5,4.5)	1436/3033	30044	2689	3	3	0.0%	21.8
ins_20_6_(2.6,4.8)	1508/3145	41408	1340339	4	3	25.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3189	44172	1106349	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2428	0	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1580/3269	48104	2669	3	3	0.0%	83.8

Table 5.23.: Solutions using Model B with SEC.3 and valid inequalities (C.16) for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3125	51585	225032	5	3	40.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2408	18152	1599191	4	3	25.0%	3600.0
ins_20_2_(2.2,3.6)	1615/4190	42550	47688	4	4	0.0%	622.2
ins_20_3_(2.3,3.9)	1585/3985	42100	98276	4	4	0.0%	1248.5
ins_20_4_(2.4,4.2)	1805/4560	29400	34273	4	4	0.0%	287.6
ins_20_5_(2.5,4.5)	1436/3732	27828	15958	3	3	0.0%	55.2
ins_20_6_(2.6,4.8)	1508/3688	43696	521995	4	4	0.0%	2082.2
ins_20_7_(2.7,5.1)	1532/3668	46660	688585	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2745	0	45	3	3	0.0%	1.0
ins_20_9_(2.9,5.7)	1580/3656	67448	9666	3	3	0.0%	147.2

Lastly, by excluding (C.13) from Model B+ or, equivalently, by solving Model B with the addition of (C.14), (C.15) and (C.16), we observe that all instances are solved to optimality (Table 5.24). However, by including also (C.13), i.e, by using Model B+, all instances are solved faster to optimality as shown in Table 5.6.

Table 5.24.: Solutions using Model B with SEC.3 and valid inequalities (C.14), (C.15) and (C.16) for test instances with 20 nodes.

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3446	11610	36636	5	5	0.0%	74.7
ins_20_1_(2.1,3.3)	1036/2681	0	0	4	4	0.0%	0.1
ins_20_2_(2.2,3.6)	1615/4411	21575	2274	4	4	0.0%	82.0
ins_20_3_(2.3,3.9)	1585/4211	21360	1956	4	4	0.0%	56.3
ins_20_4_(2.4,4.2)	1805/4681	17150	6373	4	4	0.0%	9.5
ins_20_5_(2.5,4.5)	1436/3837	20988	191	3	3	0.0%	19.1
ins_20_6_(2.6,4.8)	1508/3761	20292	10802	4	4	0.0%	27.8
ins_20_7_(2.7,5.1)	1532/3737	19436	628193	4	4	0.0%	726.5
ins_20_8_(2.8,5.4)	1167/2782	0	0	3	3	0.0%	0.2
ins_20_9_(2.9,5.7)	1580/3709	45112	9672	3	3	0.0%	51.5

6. Conclusion

In this thesis various aspects of the node partitioning and subtours creation problem were discussed. In this section we give a short summary of the thesis and suggest possible future work.

After formally defining NPSC on a graph $G_{\mathcal{I}}$, we introduced preprocessing techniques to reduce the size of $G_{\mathcal{I}}$ by deleting edges, which cannot be included in any feasible solution. Moreover, by using the conflict graph of $G_{\mathcal{I}}$ we are able to find sets of nodes, which cannot lie in the same set in a feasible solution. We also proved that the size of a maximum clique in the conflict graph is a lower bound for NPSC. In Chapter 3, we discuss a heuristic, similar to the cheapest insertion heuristic for TSP, which produces a feasible solution for NPSC. This solution is an upper bound to the optimal objective value, and is used as a starting solution for our four programming models presented in Chapter 4. The differences between our models are the following: The first one, model A, contains non-linear constraints, which are linearised in model B. The third model, model B+, additionally includes valid inequalities derived from the maximal cliques in the conflict graph, the symmetry of NPSC solutions, and further subsets of nodes that cannot lie in the same set in any feasible solution. All previous models share the same subtour elimination constraints, which are added during the optimization in a lazy fashion (section 4.2). Additionally, we derived a compact model, based on the MTZ formulation of TSP, which includes beforehand all subtour elimination constraints. In Chapter 5 we tested the heuristic and all programming models on randomly generated test instances, and presented our computational results. The best performing models on our test instances are Model B+ and Model MTZ.

6.1. Future Work

Similarly to other combinatorial optimization problems like TSP and VRP, many different variations of NPSC can be created and are left for the future. For example, by removing condition (1.1), more subsets of a feasible solution can contain the same nodes, which is meaningful, since in many cases, for example in surveillance applications, multiple vehicles are allowed to patrol same areas at different moments, see Drucker et al. [10]. It is also important to examine the performance of our models on test instances based on real problems and not random numbers. Hence, testing such instances is of interest. Furthermore, the problem input may change with time, for example by slightly adjusting the critical times or by adding new nodes to a problem. It is interesting to know if we can use known optimal solutions, in order to find feasible solutions for slightly modified problems faster. Further, it is worthwhile trying to improve our heuristic or create a better one, since a good quality starting solution can speed up the optimization process. Lastly, since the 3-node infeasible subsets valid inequalities and the ones derived from the conflict graph made Models B+ and MTZ perform much better than Model A and B, finding more infeasible combinations of nodes could beneficial for their performance.

Bibliography

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[2] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[3] H.-J. Bockenhauer, J. Hromkovic, J. Kneis, and J. Kupke. The parameterized approximability of TSP with deadlines. *Theory of Computing Systems*, 41(3):431–444, 2007.

[4] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* London: The Macmillan Press Ltd, 1976.

[5] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[6] O. Burdakov. Node Partitioning and Subtours Creation (NPSC): Problem formulation and its MIP model. *Private Communication* , 2018.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3 edition, 2009.

[8] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale Traveling Salesman Problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[9] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[10] N. Drucker, M. Penn, and O. Strichman. Cyclic routing of unmanned aerial vehicles. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 125–141. Springer, 2016.

[11] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An optimal algorithm for the Traveling Salesman Problem with Time Windows. *Operations research*, 43(2):367–371, 1995.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[13] Gurobi Optimization LLC. Gurobi optimizer reference manual, 2018. `http://www.gurobi.com`.

[14] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations*, volume 12. Springer Science & Business Media, 2006.

[15] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[16] H. M. Ho and J. Ouaknine. The Cyclic-Routing UAV problem is PSPACE-Complete. In A. Pitts, editor, *Foundations of Software Science and Computation Structures*, pages 328–342. Springer, 2015.

[17] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[18] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[19] J. Lee and S. Leyffer. *Mixed Integer Nonlinear Programming*, volume 154. Springer Science & Business Media, 2011.

[20] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[21] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 5 edition, 2013.

Appendices

A. TSP Subtour Elimination Constraints

The subtour elimination constraints (2.2.2) are included during optimization in a lazy fashion, as explained in the Gurobi optimizer manual [13] and in **Algorithm 4**.

Algorithm 4 Pseudocode

```
1: function SubtourElimination
2:     x ← current solution
3:     C ← {(i, j) ∈ E | x_ij = 1}                    ▷ get all edges
4:     if C contains multiple subtours (⋆) then        ▷ x is not feasible
5:         choose smallest cycle C_s in C
6:         S ← nodes of C_s
7:         model ← addLazyConstraints (⋆⋆)             ▷ cut off infeasible solutions
```

($\star$) In order to detect subtours, we compute the shortest cycle C_s with edges in C. If C_s has $|V|$ edges, then a tour with of minimal length satisfying all constraints is found. Otherwise, if the cycle has less than $|V|$ edges, we eliminate these subtours by adding the following constraint to the model:

$$(\star\star) \qquad \sum_{i,j \in S, i \neq j : \{i,j\} \in E} x_{ij} \leq |S| - 1 \,.$$

Then, the optimization process continues until the shortest cycle has length $|V|$ and hence no subtours exist.

B. Further Computational Results for Models A, B, B+

Table B.1.: Solutions using Model A with SEC.1
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	1150	195383	5	2	60.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	100	173909	4	2	50.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	9700	40123	5	2	60.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	255	124204	5	2	60.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	1190	176970	5	2	60.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	73	75950	4	2	50.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3072	29	84231	4	2	50.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	40	82483	4	2	50.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2391	123	84539	3	1	66.7%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	340	115642	4	1	75.0%	3600.0

Table B.2.: Solutions using Model A with SEC.2
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	330	149135	5	2	60.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	48	185643	4	2	50.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	4335	35404	5	2	60.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	565	133636	5	2	60.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	280	194751	5	2	60.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	324	71146	4	1	75.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3072	49	76742	4	2	50.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	3124	34014	4	2	50.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2381	213	84782	3	1	66.7%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	110	133668	4	1	75.0%	3600.0

Table B.3.: Solutions using Model B with SEC.1
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	51945	586731	5	3	40.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	71172	773017	4	3	25.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	74335	470021	5	3	40.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	68515	459197	5	3	40.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	58945	594597	5	3	40.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	68324	349745	4	2	50.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3072	79896	388410	4	2	50.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	106676	755772	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2391	94911	540977	3	2	33.3%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	72364	349597	3	2	33.3%	3600.0

Table B.4.: Solutions using Model B with SEC.2
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/2705	17475	1810162	5	3	40.0%	3600.0
ins_20_1_(2.1,3.3)	1036/2128	4296	7872607	4	3	25.0%	3600.0
ins_20_2_(2.2,3.6)	1615/3285	22100	1095764	4	3	25.0%	3600.0
ins_20_3_(2.3,3.9)	1585/3225	16330	1361639	3	3	25.0%	3600.0
ins_20_4_(2.4,4.2)	1805/3665	29965	686958	5	3	40.0%	3600.0
ins_20_5_(2.5,4.5)	1436/2928	15184	2246678	4	3	25.0%	3600.0
ins_20_6_(2.6,4.8)	1508/3072	20576	924214	4	3	25.0%	3600.0
ins_20_7_(2.7,5.1)	1532/3120	16868	3251604	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2391	6900	4163612	3	2	33.3%	3600.0
ins_20_9_(2.9,5.7)	1580/3216	28212	345629	3	2	33.3%	3600.0

Table B.5.: Solutions using Model B+ with SEC.1
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3450	3665	153027	5	5	0.0%	43.7
ins_20_1_(2.1,3.3)	1036/2684	0	0	4	4	0.0%	0.3
ins_20_2_(2.2,3.6)	1615/4415	12665	227064	4	4	0.0%	76.7
ins_20_3_(2.3,3.9)	1585/4215	2080	4317	4	4	0.0%	4.4
ins_20_4_(2.4,4.2)	1805/4685	6475	18307	4	4	0.0%	24.4
ins_20_5_(2.5,4.5)	1436/3840	20572	16898	3	3	0.0%	29.2
ins_20_6_(2.6,4.8)	1508/3764	41748	408335	4	4	0.0%	939.5
ins_20_7_(2.7,5.1)	1532/3740	27656	4148918	4	3	25.0%	3600.0
ins_20_8_(2.8,5.4)	1167/2784	0	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1580/3712	148152	823063	4	3	25.0%	3600.0

Table B.6.: Solutions using Model B+ with SEC.2
for test instances with 20 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_20_0_(2.0,3.0)	1325/3450	880	62407	5	5	0.0%	35.7
ins_20_1_(2.1,3.3)	1036/2684	0	0	4	4	0.0%	0.1
ins_20_2_(2.2,3.6)	1615/4415	2630	5606	4	4	0.0%	20.2
ins_20_3_(2.3,3.9)	1585/4215	2480	14199	4	4	0.0%	16.4
ins_20_4_(2.4,4.2)	1805/4685	1345	4208	4	4	0.0%	4.1
ins_20_5_(2.5,4.5)	1436/3840	764	1506	3	3	0.0%	2.6
ins_20_6_(2.6,4.8)	1508/3764	792	46464	4	4	0.0%	20.9
ins_20_7_(2.7,5.1)	1532/3740	2380	86823	4	4	0.0%	37.6
ins_20_8_(2.8,5.4)	1167/2784	0	0	3	3	0.0%	0.1
ins_20_9_(2.9,5.7)	1580/3712	2532	2340	3	3	0.0%	5.5

Table B.7.: Solutions using Model B+ with SEC.1
for test instances with 30 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_30_0_(2.0,3.0)	4137/12655	52703	403106	7	5	28.6%	3600.0
ins_30_1_(2.1,3.3)	2925/9010	0	38	5	5	0.0%	11.4
ins_30_2_(2.2,3.6)	3675/11145	2735	4863017	5	4	20.0%	3600.0
ins_30_3_(2.3,3.9)	4266/12720	38472	567425	5	5	0.0%	1707.2
ins_30_4_(2.4,4.2)	5411/14825	96173	670530	6	5	16.7%	3600.0
ins_30_5_(2.5,4.5)	3268/9914	0	34	4	4	0.0%	36.1
ins_30_6_(2.6,4.8)	3276/9738	5016	2929793	4	4	0.0%	930.3
ins_30_7_(2.7,5.1)	4155/11875	60400	986452	5	4	20.0%	3600.0
ins_30_8_(2.8,5.4)	2619/7056	0	0	3	3	0.0%	0.6
ins_30_9_(2.9,5.7)	3540/9226	94436	909998	4	3	25.0%	3600.0

Table B.8.: Solutions using Model B+ with SEC.2
for test instances with 30 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_30_0_(2.0,3.0)	4137/12655	7273	107066	6	6	0.0%	827.2
ins_30_1_(2.1,3.3)	2925/9010	0	38	5	5	0.0%	11.5
ins_30_2_(2.2,3.6)	3675/11145	705	2272156	5	5	0.0%	3116.0
ins_30_3_(2.3,3.9)	4266/12720	14880	315874	5	5	0.0%	871.0
ins_30_4_(2.4,4.2)	5411/14825	22925	25164	5	5	0.0%	445.0
ins_30_5_(2.5,4.5)	3268/9914	0	34	4	4	0.0%	36.3
ins_30_6_(2.6,4.8)	3276/9738	852	4856	4	4	0.0%	13.1
ins_30_7_(2.7,5.1)	4155/11875	13470	1979432	5	4	20.0%	3600.0
ins_30_8_(2.8,5.4)	2619/7056	0	0	3	3	0.0%	0.7
ins_30_9_(2.9,5.7)	3540/9226	11680	1213837	4	3	25.0%	3600.0

Table B.9.: Solutions using Model B+ with SEC.1
for test instances with 35 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_35_0_(2.0,3.0)	6424/21073	33712	176204	8	6	25.0%	3600.0
ins_35_1_(2.1,3.3)	3965/13030	0	56	5	5	0.0%	23.3
ins_35_2_(2.2,3.6)	5790/18141	48804	945628	6	5	16.7%	3600.0
ins_35_3_(2.3,3.9)	6769/21329	42244	716985	7	5	28.6%	3600.0
ins_35_4_(2.4,4.2)	7329/20552	60571	197191	7	5	28.6%	3600.0
ins_35_5_(2.5,4.5)	4444/14745	0	34	4	4	0.0%	12.6
ins_35_6_(2.6,4.8)	4516/14573	12156	12210747	4	3	25.0%	3600.0
ins_35_7_(2.7,5.1)	5725/17715	26563	3942851	5	4	20.0%	3600.0
ins_35_8_(2.8,5.4)	4716/13193	114404	659651	4	3	25.0%	3600.0
ins_35_9_(2.9,5.7)	4940/15485	84452	676718	4	3	25.0%	3600.0

Table B.10.: Solutions using Model B+ with SEC.2
for test instances with 35 nodes

	Vars/Cons	SEC	NodeCnt	UB	LB	GAP	Time(s)
ins_35_0_(2.0,3.0)	6424/21073	32768	387565	7	6	14.3%	3600.0
ins_35_1_(2.1,3.3)	3965/13030	0	56	5	5	0.0%	23.4
ins_35_2_(2.2,3.6)	5790/18141	10074	21184	5	5	0.0%	373.6
ins_35_3_(2.3,3.9)	6769/21329	28826	1040733	6	5	16.7%	3600.0
ins_35_4_(2.4,4.2)	7329/20552	22085	487049	7	5	28.6%	3600.0
ins_35_5_(2.5,4.5)	4444/14745	0	34	4	4	0.0%	12.6
ins_35_6_(2.6,4.8)	4516/14573	348	256	4	4	0.0%	11.1
ins_35_7_(2.7,5.1)	5725/17715	8786	2191867	5	4	20.0%	3600.0
ins_35_8_(2.8,5.4)	4716/13193	12764	133712	3	3	0.0%	182.5
ins_35_9_(2.9,5.7)	4940/15485	9852	2744694	4	3	25.0%	3600.0