FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Optimizing Trip Itinerary for Tourist Groups

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Kadri Sylejmani
Matrikelnummer 0728620

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dipl. -Inf. Dr. -Ing. Jürgen Dorn

Diese Dissertation haben begutachtet:

(Ao. Univ. Prof. Dipl. -Inf. Dr. -Ing. Jürgen Dorn)	(Assoc. Prof. Dr. Ing. Lule Ahmedi)

Wien, 05.09.2013

(Kadri Sylejmani)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Optimizing Trip Itinerary for Tourist Groups

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Kadri Sylejmani
Registration Number 0728620

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ. Prof. Dipl. -Inf. Dr. -Ing. Jürgen Dorn

The dissertation has been reviewed by:

(Ao. Univ. Prof. Dipl. -Inf. Dr. -Ing. Jürgen Dorn)	(Assoc. Prof. Dr. Ing. Lule Ahmedi)

Wien, 05.09.2013

(Kadri Sylejmani)

Erklärung zur Verfassung der Arbeit

Kadri Sylejmani
Ulpiana U1, H-1, Nr. 9, 10000, Prishtina, Kosova

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

During the period of my PhD studies, the support of the members of my family was crucial in order to come to this concluding stage. It was my wife Sadete, who would always support me in every dimension of my life, and would always try to create the atmosphere for me so that I could remain focused in my PhD work. Her love, courage and care have kept me always strong on dealing with challenges I have faced during this period. I thank her for the double role she has had to play to cover my absence in front of our son Adi and daughter Jeta, due to my frequent stays in Vienna. I thank Adi and Jeta for their love, patience and understanding in this period of intensive engagements. The backing and blessing of my mother was always present. I am thankful to her for the care and protection she has given to me throughout my life. I am grateful to my oldest brother Halil, who has been supporting me since my early stages of education. I also thank my sisters Qama and Sevdie, and my brothers Sahit, Sabit, Sadik, Sadri and Bedri and their wives for their support and help. In addition, I also want to thank all my nephews and nieces.

My success in this thesis comes as result of great support from my supervisor Prof. Jürgen Dorn, who has always been there for me in advising and listening to my proposals. His readiness in backing me up was essential for me to focus in achieving the real aimed goal in this thesis. His valuable feedback has always helped me get to a step further in understanding the research work. I have to thank him also for understanding and supporting me with various administrative letters that I have needed frequently over the last few years. I thank him and his wife Erika for hosting me and Sadete to their house and thus give us a rear possibility to get more insights about Austrian and German traditions and cultures.

There is professor that has given a tremendous support to me in two major directions. The first is his remarkable help for me to start building my research capacities along with his continuous and indispensible support in my PhD work. Meanwhile, he has helped me to stay in a working mood by providing to me the possibility to also get into the rhythm of the social life whenever I stayed in Vienna. I thank you a lot for this Prof. Nysret Musliu. In addition, I thank Prof. Nysret for reading this PhD thesis and for his valuable comments.

I thank professors Prof. Agni Dika, Prof. Lule Ahmedi and Prof. Blerim Rexha from Department of Computer Engineering at Faculty of Electrical and Computer Engineering, University Prishtina for their support. I am especially thankful to Prof. Lule Ahmedi who acted as the second evaluator of this thesis and who led the research project that was partially within the topic of this PhD thesis. I thank Prof. Agni Dika for the motivation given to me towards concluding my PhD studies. I also thank the dean of Faculty of Electrical and Computer Engineering Prof. Enver Hamiti and his team for the support given during my final stages of PhD

studies. I thank Ehat Qerimi, Zenun Kastrati, Vehbi Neziri and Artan Mazrekaj for covering me in my teaching hours during my stays in Vienna. In addition, I take the opportunity to thank all the other members of Department of Computer Engineering and the other staff of the faculty.

I thank director of IT at the rectorate of University of Pristhina Prof. Isak Shabani for providing the facilities for testing the algorithms developed in this thesis. I thank Atdhe Muhaxhiri for his support and reediness to talk to me about the issues related to my PhD work. I thank Bujar Krasniqi and Albin Ahmeti for their support and friendship during my stays in Vienna.

My research work was partially supported by Austrian Federal Ministry of Education under the project "Bertha von Suttner-Stipendien/Kosovo" and Ministry of Education, Sciences and Technology of Kosova under the research project "Tourist Tour Planning and Social Network Analysis". I am thankful for that.

Abstract

In this PhD thesis, we solve the problem of trip itinerary planning for tourist groups. We assume that each tourist has individual preferences for points of interest and preferences with whom in the group s/he would like to share travelling. This problem has practical relevance, since often tourists prefer going on trip in company, while aiming to visit points of interest that meet their individual preferences. Hence, the proposed solution can relieve the tourists from the task of itinerary planning. However, in terms of optimization, this problem is complex due to its objective to maximize the satisfaction of all tourists, while coping with various individual constraints.

We follow a path of three steps for solving the problem at hand. In the first step, we develop an algorithm based on Tabu search for solving the solo trip planning problem, which we compare in quality and time consumption against existing solutions in the literature. In the second step, we define the group trip problem that models the objective and the constraints of itinerary planning for a whole tourist group. Further, we use the algorithm developed in the first step to devise three straightforward approaches for solving group trip problem. The difference between the three approaches is whether tourist travel alone, in subgroups or altogether. Finally, in the third step, we extend our planning algorithm, to develop a sophisticated group trip planning algorithm, which allows tourists to have a personalised trip itinerary throughout the trip. In order to meet individual preferences of tourists, the group trip algorithm combines the itinerary of different tourists so that they are sometimes planned to travel alone, and at other times travel in groups.

In order to evaluate the proposed algorithms, we create a new benchmark for the group trip problem by extending the existing benchmarks in the literature for the solo trip problem. We conduct the computational experiments with four different proposed approaches by using the new benchmark. The proposed approaches are evaluated based on the quality and respective computation time. Our experiments show that our sophisticated algorithm finds always a better solution than the other three approaches for our extended benchmarks. In addition, we also make experiments with the aim of identifying two modes of algorithm execution, one which produces high quality solutions with a longer computation time, and the other one that produces slightly worse solutions, but with a much less computation effort.

Kurzfassung

In dieser Arbeit wird eine Lösung für die Planung eines Weges für Gruppen von Touristen vorgeschlagen. Wir nehmen an, dass jeder Tourist individuelle Präferenzen für interessante Punkte hat und auch individuelle Präferenzen hat, mit wem er gemeinsam reisen will. Dieses Problem hat praktische Relevanz, weil häufig Touristen es vorziehen in Gruppen zu verreisen, wobei sie aber stets auch ihre eigenen Präferenzen zumindest teilweise erfüllt haben möchten. Folglich kann die vorgeschlagene Lösung Touristengruppen von der Planung der Tour befreien, Aus der Sicht der Optimierungtheorie ist das Problem komplex, weil die maximale Erfüllung von Zielen aller Touristen unter Beachtung weiterer individueller Einschränkung erreicht werden soll.

Auf dem Weg zur Lösung haben wir drei wesentliche Schritte gesetzt. Im ersten Schritt haben wir eine Lösung zur Suche eines optimalen Weges für einen einzelnen Touristen entwickelt, die wir mit existierenden Lösungen in Bezug auf Qualität und Zeitverbrauch vergleichen. Im zweiten Schritt haben wir das Group Tour Problem definiert, das die Optimierung und Berücksichtigung aller Touristen in einer gemeinsamen Gruppe beschreibt. Wir stellen drei Ansätze vor, wie das Group Tour Problem mit dem im ersten Schritt entwickelten Algorithmus gelöst werden kann. Der Unterschied dieser drei Ansätze beruht auf der Entscheidung, ob für jeden Touristen einzeln ein Weg geplant wird, für Untergruppen oder für die ganze Gruppe. Schliesslich erweitern wir im dritten Schritt unseren Algorithmus, um ein verbessertes Verfahren zu erhalten, das es erlaubt, dass Touristen einen individualisierten Weg vorgeschlagen bekommen, Dieser Group Tour Algorithmus versucht die Wege der verschiedenen Touristen teilweise vereinen, um eine Maximierung aller Präferenzen zu erreichen.

Um die vorgeschlagene Lösung zu evaluieren, entwickeln wir einen neuen Benchmark Test indem wir die in der Literatur existierende Benchmarks für Einzeltouren erweitern. Wir führen Experiment auf Basis des Benchmark Tests mit den vier genannten Ansätzen durch und vergleichen die Ergebnisse in Bezug auf Qualität und benötigter Rechenzeit. Die Experimente zeigen, dass unser verbesserter Algorithmus für die erweiterten Benchmarks immer eine bessere Lösung findet als die drei anderen Ansätze. Zusätzlich führen wir Experimente durch um zwei Ausführungsmodi zu unterscheiden, bei denen entweder kurze Verarbeitungszeit oder höchste Qualität im Vordergrund steht.

Contents

CHAPTER 1

Introduction

The tourism, as defined by World Tourism Organization [93], includes "the activities of persons traveling to and staying in places outside their usual environment for not more than one consecutive year for leisure, business and other purposes". The tourists when visiting a certain travel destination might get engaged in different activities such as walking tours, site seeing, guided tours, shopping and other leisure related activities. In general, the tourism experience for tourists is embodied into three phases of tourists' life cycle (Figure 1.1), namely pre-tour, on-trip and post-trip phase. In the pre-trip phase, the tourist gets engaged in usual preparations for the trip such as booking an accommodation, purchasing traveling tickets, planning site visits, etc. Even though, in the on-trip phase, the tourists are concentrated on conducting the priory planned activities, they still might be in need for getting additional information such as ad hoc navigation information, information about Points of Interest (POIs), contextual information (weather, location, time), etc. The post trip phase is mainly characterized with social interactions about sharing the trip experience with other people and provision of feedback for certain activities accomplished during the trip.

The systems and tools of Information and Communication Technology (ICT) that are developed to support tourism activities in all three phases of tourist's life cycle, make up the electronic tourism (eTourism) research field. In [14], Buhalis and Law identify three main directions of eTourism research, namely: consumers and demand dimensions, technological innovation and

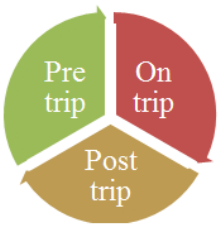


Figure 1.1: Tourists' life cycle [141]

industry functions. The consumer and demand dimension is about provision of access to tourism services for tourists, in an accurate and reliable way. The service might include accommodation reservation, tourism information search, travel and holiday package purchase, etc. The technological innovations, along to continuous improvement of hardware, software, network appliances, includes also development of advanced techniques [14] like interoperability and ontology building [141], multimedia, mobile and wireless technologies, wireless local area networks (WLANs), web design functionality and usability. The dimension of industry in eTourism is concerned with support in advancing business functions and processes of an organization operating in tourism sector, thus contributing in achieving its strategic and operational objectives. Furthermore, Buhalis and Law [14] indicate that, in future, eTourism research will be focused on consumer centric technologies that would enable organizations and the tourists to interact dynamically.

A research agenda in e-tourism that has recently gained a considerable attention is the problem of automatic planning of tourist trip itinerary [45]. This problem can be characterized with the situation when a tourist travels to a touristic destination with intention to visit a number of POIs during a period of limited time. In general, the number of POIs available in a touristic destination is much higher than the number of POIs that can be visited by a tourist. Moreover, the maximal satisfaction of a tourist could be achieved if POIs that suit the most her/his preferences are visited. Additionally, tourists might enforce several other constraints for their trip, like maximum amount of money they want to spent, or maximum number of POIs of certain category (e.g. religious art) they want to visit. Undermining these trip constraints, the objective would be to prepare a trip itinerary that includes the POIs that meet tourist preferences at a maximum extent.

Itinerary planning is a complex problem

The problem of creating the trip itinerary for a tourist is known as "tourist trip design problem" (TTDP) [135]. In its most basic model, the TTDP problem could be described in following details [45]:

- In a certain tourist destination there exist a number of POIs associated with a number of features (e.g. location, type, category, working hours, entry fee, etc.).
- The traveling time for the distances between POIs is calculated by considering different means of transportation (e.g. car, bicycle, walking or public transportation).
- The "score" of a POI could be calculated by using a weighted function for the objective (e.g. popularity) and subjective (e.g. user specific preferences) importance of its features.
- The estimation of visit duration to a POI could be calculated based on average visit duration and on the match of tourist preferences to the POI features (type and category).
- The number of days of the trip to the tourism destination determines the number of itinerary routes that must generated.

- The tourist specifies the time duration of each single route, which includes the times spent in visiting the POIs and the traveling times between them.

Based on different additional parameters and constraints, several versions of TTDP are derived. The Orienteering Problem (OP) [129] can be used to model the simplest form of TTDP. The OP is defined as an orienteering game, where a set of locations with scores have to be visited during a single route. A location can be visited at most once and the aim is to visit those points that maximize the total collected score, which have to be visited within the specified time limit for the route. The Team OP (TOP) [21] is used to model the trip consisting of multiple periods (e.g. multiple days), while TOP with Time Windows (TOPTW) [134] is used to additionally model the operating/working hours of POIs. Note that other possible extensions to OP are described in the next chapter.

Vansteenwegen et al. [133] summarize a number of complexity issues for solving OP. In [57], Golden et al. prove that OP is a NP-hard problem and underlined that no polynomial time algorithm could be designed to produce optimal solutions. In such occasions, systematic algorithms can be very time intensive and thus not be suitable for practical implementations where a quick response time is expected. Hence, the alternative would be tackling such problems by using heuristic algorithms. The difficulties for designing good heuristic algorithms, as indicated by Gendreau et al. [52], stand in the fact that score of a location and time to get to that location are independent and often contradictory. In this respect, a simple and straightforward heuristic function might lead the search process toward getting stack in local optimum and thus avoid searching the whole search space. Vansteenwegen et al. [130] argue that the most difficult test instances to solve are those that enable insertion of a little more than half of available points into the route plan. Further, finding the optimal route path becomes harder as the number of inserted points increases.

PROBLEM: Planning trip itineraries for tourists is a NP - hard problem.

Systems support itinerary planning

There is a plethora of tourist trip planning functionalities that are supported by different approaches in literature [117]. Basically, such functionalities enable the tourist to make automatic trip planning, including selections of POIs, routing between POIs, enforcing different planning constraints (e.g. maximum tour duration), and consideration of tourist context (e.g. weather conditions). In Table 1.1, the first column shows a list of most popular planning functionalities, while the second column depicts the respective description.

Functionality	Description
Estimation of tourist interest	Determining how much a certain POI meets tourist preferences
Automatic selection and routing of POIs	Automatic inclusion of POIs into itinerary and finding out the optimal order/sequence of visit
Mandatory POIs	POIs that are very popular may be denoted as ("must see")
Dynamic recalculation of itinerary	Possibility to regenerate the trip itinerary in case the pre-scheduled itinerary becomes infeasible or less satisfactory
Multiple day tour planning	Planning the trip itinerary for multiple days
Opening hours	Considering the working/operating hours of POIs
Budget limitations	Considering tourist budget limitations for visiting POIs
Weather dependency	Taking into account the weather conditions when planning the trip (e.g. outdoor POIs might become less desirable by tourists in rainy weather conditions)
Max n-types	Tourist might set up a limit for visiting a maximum n number of POIs of certain type
Mandatory types	Tourist can make some POI types mandatory (e.g. there must be at least one POI of type nature present in itinerary)
Scenic routes	A number of POIs might be located into a certain street (e.g. downtown of a city), known as "scenic routes", and their inclusion into trip itinerary might increase the tourist satisfaction
Hotel selections	Automatic selection of the accommodation for tourist, which would serve as the end point for actual tour and as the start point for the next tour
Public transportation	Taking into account the schedules of public transportation facilities such as bus, train and metro
Group profiles	Considering the interests of multiple tourists about POIs

Table 1.1: Planning functionalities

Diverse commercial systems or prototypal implementations support more or less the above presented planning features. For instance web or mobile systems like City Trip Planner[1], Your Tour[2], Plnnr [3], Trip Tern [4] and Mtrip[5] can be used to plan personalized trip itineraries for number of popular cities around the world. Souffriau et al. [117] compare the existing tour scheduling approaches that implement different functionalities presented above.

Although, many planning features are supported, none of the systems or approaches consider the problem of group trip planning, where a group of tourists (e.g. a group of friends) would

[1] http://www.citytripplanner.com/
[2] http://www.yourtour.com/
[3] http://plnnr.com/
[4] http://triptern.com/
[5] http://www.mtrip.com/

prefer to conduct a joint trip. In general, in such situations, most tourists might like to conduct the trip all together, but probably some of them would like to retain the option to get separated from the main group for some part of the trip so that they could visit some POIs of their own specific interest.

PROBLEM: Tourist trip planning systems do not enable trip planning for a group of tourists.

The scope of the thesis

The work in this thesis aims to solve the aforementioned trip itinerary planning problem for tourist groups. The problem at hand is about planning the detailed tour itinerary for a group of tourists who visit a certain number of POIs during a trip of multiple periods (days). Further, the envisioned problem is associated with some additional constraints about the trip, such as individual tourist budget, maximum number of POIs of certain POI category, and time windows of POIs. The objective is to plan a joint trip itinerary for all tourists, with options of separation in some parts of the trip, so that the overall tourists' satisfaction is maximized by enabling them to visit the most preferred POIs in company of the closest related group members.

A prerequisite to planning the trip itinerary, is the existence of a suitable method for the process of matchmaking [76] between the features of POIs (categories and types) and tourist preferences. The process of matchmaking estimates tourist interest in particular POIs based on her/his preferences about types (e.g. mosque, church, monument) and categories (e.g. archeology, architecture, nature, etc.). Note that it is out of scope of this thesis to build a new method for estimating the interest of tourists about POIs. Additionally, a function that reflects the mutual social relationship (connectedness) level between individual tourists in the group is needed, but the introduction of a new method for that purpose is again out of scope of this thesis.

In order to compare our solution with existing approaches in the literature, as far as possible we use the existing test instances of Solomon [116] for the Vehicle Routing and Scheduling Problem (VRSP) and Cordeau et al. [25] for Multi-Depot Vehicle Routing Problem (MDVRP). Further, we extend these instances to reflect the new problem of tourist groups, so that the applicability of the proposed method, for different situations that might be encountered in real life, can be assumed.

In addition, we study the methods for determining the match between members in tourist groups based on their preferences about POIs and their mutual social relationship. This is done with the aim of finding out whether subgrouping the tourists into smaller groups could be beneficial when planning the trip itinerary.

The envisioned problem of this thesis, as an extension of MCTOPTW problem for modeling the group trip planning problem is characterized with far more huge search space than the original problem. Hence, in addition to basic procedures from the literature for exploring the search space, new procedures must be investigated and developed.

1.1 Research Question

In this thesis, we aim to tackle the problem of group trip itinerary planning that is an extension of the general Tourist Trip Design Problem (TTDP) [135]. As described earlier, the Multi Constraint Team Orienteering Problem with Time Windows (MCTOPTW) [43], as a particular version of TTDP problem used to model a version of trip planning problem for a single tourist, allows modeling of multiple day tours, multiple constraints about POI types or categories and single working hours of POIs. Hence, we focus in extending and solving the problem of MCTOPTW for multiple tourists. We name the extended version of MCTOPTW as Multi Constraint **Multiple** Team Orienteering Problem with Time Windows (MCMTOPTW). The difficulty in solving the proposed problem derives from the original problem of Orienteering Problem that is proven by Golden et al. [57] to belong to the class of NP hard problems.

Considering the situations where a group of tourists visits a city or region for a number of days, where the level of interest of tourists about POIs and the their mutual social relationship is known, we formulate the research question of this PhD thesis that reads as follows:

> *Can we improve trip planning systems for tourist groups by considering individual preferences and social relationship?*

In addition, in order to answer the proposed research question, we formulate the following three hypotheses:

1. *An existing method for solo trip itinerary planning can be utilized for planning the trip itinerary for a group of tourists*
2. *By clustering tourists into subgroups based on their preferences and social relationship the results could be improved*
3. *New specialized operators in local search techniques may further improve the solutions for tourist groups*

Note that the scope of this PhD thesis is not to develop a method for estimation of tourist interest about POIs, or to create a method for estimating the social relationship between different tourists.

1.2 Main Contributions

The modules and the flow of arrows in the block scheme represented in figure 1.2 show different variants for preparing the trip itinerary for a group of tourists. Each module in the block scheme reflects a distinct contribution of the work in this thesis as described in the following:

1. The solo trip planner is an optimization algorithm based on tabu search meta heuristic that enables planning a multiple day trip itinerary for a single tourist. The algorithm explores the search space by applying operators for insertion of new POIs, replacing existing POIs with new ones (from outside the itinerary), and swapping the existing POIs in the itinerary

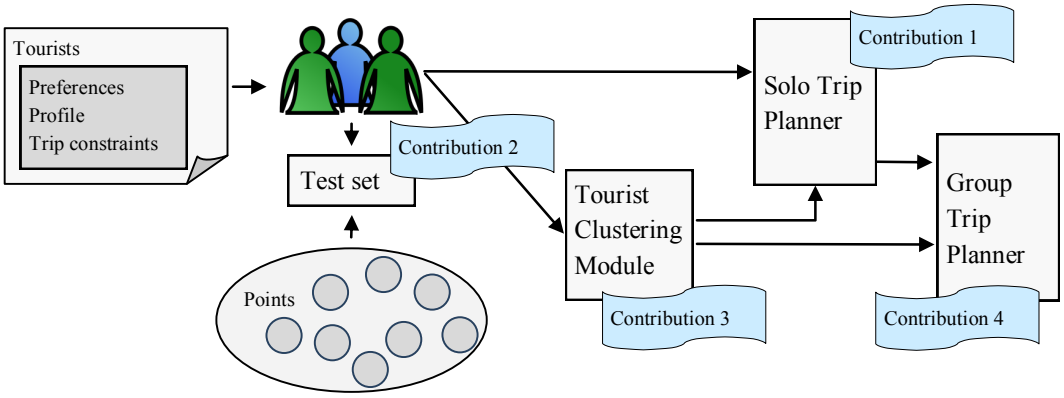


Figure 1.2: Main contributions of the thesis

between each other. The algorithm takes advantage of a tabu list for employing a perturbation and a diversification mechanism so that it does not get caught in local optimum. The algorithm performance is comparable to the existing solutions in the literature.

2. The existing test set in the literature (Solomon [116] and Cordeau et al. [25] originated test set) that can be used for modeling solo trip planning, is extended for modeling the tourist trip planning problem for a group of tourist. The existing test set is enriched with data for multiple tourists, including the data for social relationship between them. The generation of the data for new tourists is done randomly by following the same patterns as the existing data in the test set.

3. The tourist clustering module is based on *K-means* method and it aims finding an optimal clustering of tourists into subgroups based on their preferences about POIs and also on their mutual social relationship. The appropriateness of the number of clusters obtained by *K-means* is evaluated by using the so called "pseudo F-static" statistical functions of Calinski and Harabasz [18]. Depending on the trip constraints, the clustering module can serve as an added component priory to applying either the single or group trip planner.

4. The group trip planner acts as a meta algorithm that initiates the search process from solutions gained with the solo trip planner either for individual tourists or for tourist sub groups (obtained by the clustering module). The group trip planner is also based on tabu search and its task is finding optimal solution for the group as a whole. The search space is explored by using two new operators, namely *Separate* (a tourist from its subgroup) and *Join* (two tourists at a POI in the tour) and one standard *Insert* operator to fill any gap created in the tour with a POI.

A systematic experimental testing of the described planning modules is conducted by using the generated test set. The strengths of the proposed approach are clearly indicated by the experiments when it comes to planning the itinerary for a group of tourists.

1.3 Methodology

The research approach of design science promoted by Hevner et al. [62] has been adopted as a methodology to answer the research question of this thesis. This methodology is a common approach for research and development in the domain of information systems. Conceptually, this approach is based on the process of developing/building theories or artifacts for an identified problem, which than, based on the problem at hand, are justified/evaluated by performing activities such as analytical analysis, case studies, experimentation, field studies or simulation. This process includes a series of assessment and refinement steps until the targeted research results are reasoned. In general, the expected research results impact both, the appropriate environment (people, organization and technology) and knowledge base (foundations and methodologies). Conversely, the business needs of a targeted environment and the existing knowledge in that environment also impact the process of research and development in information systems.

In our case, as described in previous section, we contribute with three artifacts in the domain of research and development, namely solo trip planner, tourist clustering method and group trip planner. In line with the terminology used by Hevner et al. [62], an *"artifact"* equals to a method and an *instantiation* corresponds to the respective implementation. In addition, we also contribute with a test set that can be used for evaluating artifacts (in our case the method for group trip planning) in the domain of research and development.

Due to the high complexity of the problem at hand, systematic approaches can not be effective in tackling such problems. Hence, we base our method on one of the frequently utilized approaches (in different domains of application) from meta heuristic techniques, namely Tabu Search [56] (described in detail in Subsection 2.5.4 of Chapter 2). Our approach for planning the trip for a group of tourists is motivated on existing approaches for solo trip planning. As result, our research work starts by creating a Tabu Search based approach for the existing problem of solo trip planning. Next, the existence of social relationship between tourist group members, motivates us to take the approach of initially clustering tourists into subgroups, and then trying out plan the itinerary for each subgroup by applying the existing solution for a single tourist. In addition, in order to optimize the tourist group itinerary, on top of the solo trip planner method, we develop a group trip planner method, also based on Tabu Search, by utilizing two new operators, namely Separate and Join.

We use the generated test set described in previous section to fine tune the parameters of the algorithms and to evaluate their overall performance. Further, we compare the results of solo trip planner with the state of the art approach and then we analyze the results of each of proposed algorithms towards finding an optimal mode of operation for different trip constraints (e.g. number of tourists, number of tours, number of POIs, etc.). In order to ensure an equal environment of algorithm execution, we use the same machine for computation experiments of all three proposed algorithms. The experiments include the complete generated test set (148 test instances), where each instance is run ten times. The minimal, maximal and average values of individual instance executions are considered for analyzing and discussing the obtained results.

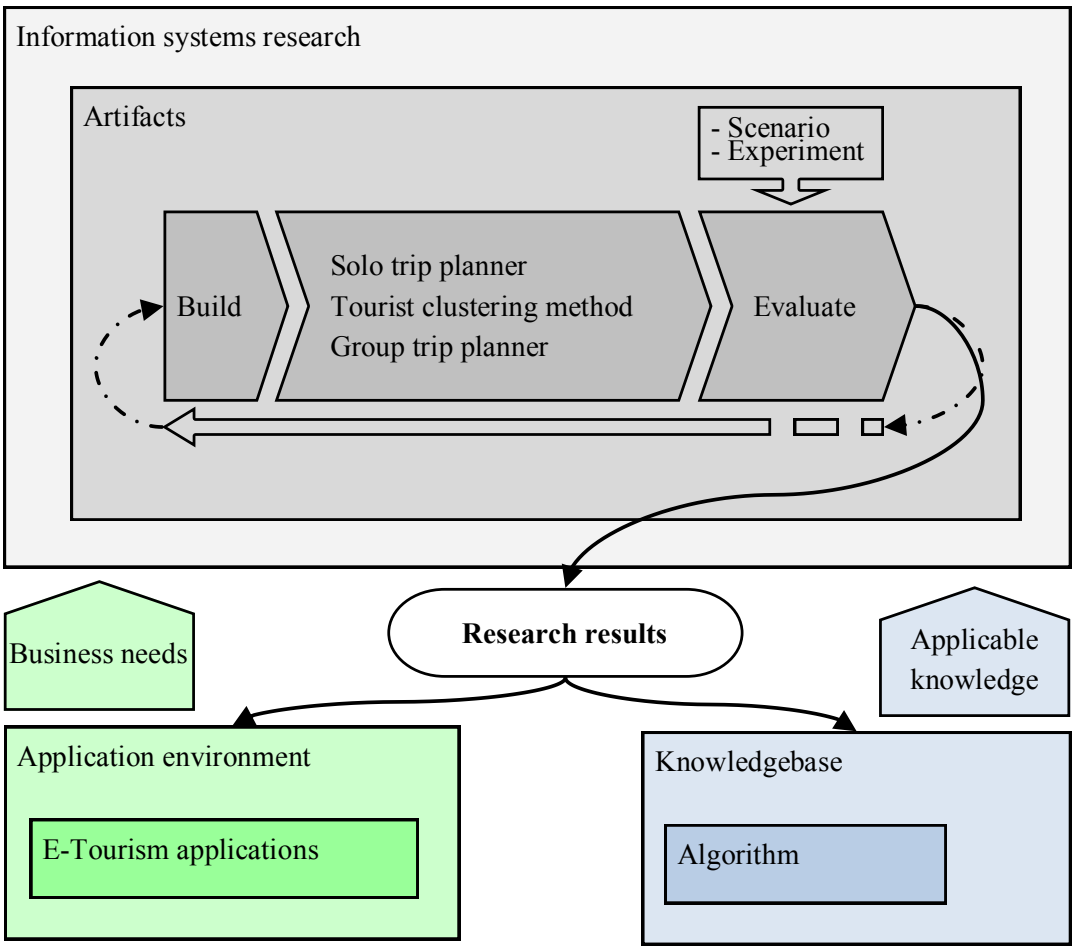


Figure 1.3: Research methodology framework

1.4 Scenario

In this section, we formulate a scenario that consists of three cases of tourist trips to city of Prishtina in Kosova. These cases are used to didactically elaborate the proposed methods throughout this thesis. Further, the cases of this scenario aim to show applicability of the proposed methods in planning the trip itinerary for tourist groups.

Case 1: A group of three friends make a trip to a Prishtina. Two of them are woman and the other one is a man. They have a close relationship between each other and mostly share common interests for visiting POIs that belong to the category of nature and architecture. In addition, the man is also interested in visiting POIs that have religious attributes, thats why he would not like to visit more than one POI of category nature. They decide to have a trip of two days and make sightseeing visits starting from 9 am until 11:30 am. The budget they are willing to spend is the same for all of them.

Case 2: A group of two couples plan two have a one day trip to Prishtina. In general, all four tourists share the same interest about visiting POIs that belong to category of archeology. In addition, the women are also interested in going to some shopping centers, although also the men would not mind if they go there. Nevertheless, the men are more interested in going to some architectural POIs. In addition, the men are not willing to spent as much money as the women. They plan to start the trip at 10 am and finish at 1 pm.

Case 3: A group of seven students of arts go for a one day trip to Prishtina. The social relationship between the members in the group is different, where some of them are more related to some members than to the others. Although all group members like visiting POIs that belong to religious arts, some of them are also interested in visiting POIs that belong to category of nature, while some others would prefer to visit archaeological sites. Different group members have different constraints abut the amount of money they are willing to spent for visiting POIs. They all agree to start the trip at 8:00 am and end it at 12:00 am.

In addition, in Appendix A, we give data of some arbitrary selected POIs in the city of Prishtina and the data of tourists of all three cases.

1.5 Structure of the Thesis

This thesis is divided into seven chapters. In the following, we present a short description for each of the remaining six chapters.

In the next chapter, we discuss the state of the art approaches existing in the literature for the domain of tourist trip planning. We first talk about Tourist Information Systems (TIS) in general and, and then specifically, about Recommender Systems that are related to tourism. Further, we describe shortly the local search techniques that are usually used for solving different variants of trip planning problems, and then we elaborate the Tabu Search metaheuristic in more details, since we use it as a framework for planning the trip itinerary for single and multiple tourists. Next, we elaborate the existing approaches in the literature that are closely related to the problem at hand. Planning the itinerary for a one day (period) tour is considered to be the most basic problem to be solved, even though, as stated earlier, it belongs to the class of NP hard problems. Multiple day (period) tour planning is further more complex, as the diapason of search space increases. Conversely, while introduction of operating hours of POIs restricts the research space to some extent, it makes the search process more complex, because the application of a number of operators become ineffective. In addition, other approaches that solve extended versions of OP are discussed.

In Chapter 3, we present a solo trip planning method that is based on Tabu Search meta heuristic. First, we give the mathematical formulation of the problem, and then we outline the fundamental characteristics of the developed algorithm. Further, we describe the operators that are used to explore the search space, as well as the techniques we employ for diversification of the search process. Finally, we present the experiments conducted for fine tuning the values of the algorithm parameters, and the comparison of the results of our approach with state of the art approaches.

Chapter 4 aims at discussing the possibilities to apply the solo trip planner method for planning the trip for a group of tourists. We start with formal definition of trip problem for tourist

groups and then introduce a clustering method that is aimed at subgrouping tourists into subgroups based on their preferences about POI types and categories and their mutual social relationship. Next, we apply the solo trip planning method for the problem of trip planning for tourist groups. The goal of the experimentation, presented at the end of this chapter, is to show whether, in general, it would be better for tourists to conduct the trip in solo, in subgroups of closely related group members, or all together.

In Chapter 5, we present a new method for planning the trip itinerary for a group of tourists. This method utilizes the solo trip planning method to create the initial solution. Further, new operators, built by combining the itinerary of different tourist group members, are used to search for better solutions that would improve the overall satisfaction of all group members. The experiments at the end of this chapter show the performance of the algorithm for different group trip modes, as well as the details for fine tuning the values of algorithm parameters.

Further, in Chapter 6, we present the evaluation of the three cases of the scenario for the different approaches of trip planning that are developed in the course of this thesis.

Chapter 7 concludes this thesis by revisiting the outlined research question and then discussing the corresponding answer based on the obtained results. The answers to the formulated hypothesis are also discussed based on the respective results achieved. Furthermore, we discuss the limitations of our PhD work and then shed light on implications and issues for future work to be challenged.

1.6 Grounding Material

The content of this thesis is partially based on two publications. Please refer to the Biography to see the full details about the listed publications.

Parts of Chapter 3 are written based on the paper:

- K. Sylejmani, J. Dorn, N. Musliu, A Tabu Search approach for Multi Constrained Team Orienteering Problem and its application in touristic trip planning, HIS 2012 [123]

Parts of Chapter 4 are written based on the paper:

- K. Sylejmani, J. Dorn, N. Musliu, Touristic trip planning: solo versus group traveling, PlanSIG 2012 [124]

CHAPTER 2

Related Work

This chapter presents the research work related to the field of tourist trip itinerary planning and it describes the particular optimization techniques used in solving the related problems. We first discuss about the tourist information systems in general (Section 2.1) and then focus in elaborating the recommender systems that are related to the domain of tourism (Section 2.2). Then, we discuss about tourist trip planning systems (Section 2.3) and social relations (Section 2.4). Afterward, we elaborate local search techniques (Section 2.5) in general and tabu search method in more details by giving more insights about modalities of its implementation and domains of successful application. Finally, in Section 2.6, we present a schema for theoretical modeling of various versions of tourist trip planning problem, and then, we review the related research work on approaches for solving single and multiple day tourist trip planning that consider: operating hours of POIs, multiple attribute constraints about POIs (e.g. entry fee, type, category, etc.) and selection of accommodation.

2.1 Tourist Information Systems

Acquisition of information is qualified as one of the most immediate needs for a tourist during her/his trip experience [35]. Hence, in the emerging age of Information and Communication Technologies (ICT), the Tourist Information Systems (TIS) play an essential role in increasing tourist satisfaction during her/his tourism experience. There exist two main desired functions to be provided by a TIS systems [9]. The first is extraction and fusing the information coming from different (and heterogeneous) information sources (e.g. web sites, tourist information providers, transport information providers, etc.), and the second is assuring that the information is up-to-date. The provided information needs to be found efficiently and conveniently [127] and should provide not only informative service, but also value added services such as for example on-line hotel reservation (e.g. booking.com), on-line route planning (e.g. Google maps), on-line flight ticket purchase (e.g. fluege.de), etc. In addition, in [97], authors identify three crucial aspects that seem to be essential for the success of TIS systems; these are quality of access, quality

of content, and ability to customize the whole system. Many actors in tourism sector such as, airlines, railway, car rentals, hotels, travel agencies, tour operators and tourist attractions all over the world, already have established their web pages for the services they offer [98] in reference to tourists needs.

There exist many TIS systems that support all three phases of tourists' life cycle [141], namely pre-trip, on-trip and post-trip. Nowadays, the social medias (e.g. blogs, wikis, social networks, virtual communities, etc.) are increasingly supporting the process of online tourism information search [142]. Recently, Emmanouilidis et al. [36] present a survey on the mobile guides existing in the literature, which can be used by tourists to get tourism information or make various transactions while they are on the move (on-trip phase).

Gretzel [59] outlines a number of types of emerging intelligent systems in tourism, such as recommender systems (recommendation of tourism services and products), context-aware systems (consideration of user context such as location, weather, type of device used, etc.), autonomous agents searching and mining Web resources (searching tourist information automatically on behalf of users), and ambient intelligence (systems that sens, through sensors, and adapt in response to the change of tourist context).

In order to outline the existence of a wide variety of TIS systems, in the following, we shortly describe some specific TIS systems that appeared recently in the literature.

The so called MOMIS (Mediator envirOnment for Multiple Information Sources) system [9] ingrates information wrapped from different tourism web sites. As such, it acts a mediator framework for heterogeneous and distributed data sources, and, in addition, it provides query management for the accumulated data into the integrated data repository. The system is based on a web services architecture by using XML technology and SOAP protocol, and thus enabling a semantic integration of its features.

In [98], the TIS system named TIScover is identified with three main characteristics in trying to achieve high quality tourism information content in terms of comprehensiveness, accurateness and certainty. The first characteristic includes a database technology with a decentralized structure, the second is about support of various retrieval methods, starting from simple navigation mechanism up to structured search and full text search capabilities, while the third one allows provision of customization capabilities about different tourism information providers, different geographic places (e.g. regions, countries, etc.). These features enable TIScover to provide complex and heterogeneous tourism information and products.

In [95], authors present a context aware TIS system, called CATIS, that is also implemented based on Web services and XML technologies. The tourist context is derived by considering different elements such as current tourist location, time of day, speed, direction of travel, personal preferences and type of device. These context elements are brought together to provide tourism information for tourists on the move. Moreover, the CATIS architecture includes a context manager that deals with both dynamic and static context elements.

In [70], a TIS system that runs on personal digital assistants (PDA), called Accessights and intended for persons with visual impairments, provides tourism information based on tourist actual location. The system uses a multi modal interface to support various user groups - each on their suitable manner. The system is based on Niccimon [5] platform, AIR3D sound framework and XML technologies.

2.2 Recommender Systems in Tourism

Resnick and Varian [103] describe recommender systems as tools to mimic the social process of recommendation that happens "in everyday life", where , "we relay on recommendations from other people either by word of mouth, recommendation letter, movie and book reviews printed in newspapers" [103]. A recommender system basically suggests products, services, or informations that are tailored to user needs and preferences, by taking into account the user actual situation and context [105]. Based on on the method used to predict the subjective interest of a user for an item, Burke [15] distinguishes four main categories of recommender systems, namely collaborative-based (analyzing correlations between users), content-based (analyzing the similarity of items which the user has rated in past), knowledge-based (derivation of user needs and preferences by explicit relation) and hybrid approach (combination of two or more techniques).

The state of the art recommendation systems in tourism "acquire the user needs and wants, either explicitly (by asking) or implicitly (by mining the user activities), and suggest destinations to visit, POIs, events/activities or complete tourist packages" [44]. The main aim of a recommender system in tourism is facilitation of the information search process for the tourist by suggesting useful information based on user profile and context. Ricci [104] discusses some earlier "travel recommender systems" such as TripMatcher [1] and Me-Print [2] that try to mimic the traditional travel agents when advising the tourist for possible holiday destinations. Further, Ricci in [105] surveys a range of recommender systems for mobile devices in regard to their user interface (e.g. starfield displays, map-based), tasks and functions supported (e.g. finding relevant attractions and services, exploring the city, route recommendation, information recommendation) and mobile computing models (e.g. context dependent, distributed models, proactive recommendation).

Recently, Gavalas et al. [44] briefly survey some representative web recommender systems in the field of tourism, such as TripAdvisor[3] , DieToRecs[4], Heracles[5] and TripSay[6], which could be merely used for recommending trips, locations and activities, and for selection of travel services. Further, Gavalas et al. [44] propose a schema for classification of mobile recommender systems in tourism based on three aspects, namely (1) chosen architecture (e.g. web based, standalone system, web-to-mobile) , (2) degree of user involvement in the delivery of recommendation (e.g. pull based, reactive, proactive), and (3) criteria considered for deriving recommendations (e.g. user preferences and constraints based, context awareness based, user critique/feedback based). Gavalas et al. [44] use the proposed schema to compare a plethora of recently developed systems in the field of mobile recommender systems.

[1] http://www.ski-europe.com/
[2] http://www.travelocity.com/
[3] http://www.tripadvisor.com/
[4] http://www.modul.ac.at/dietorecs
[5] http://www.isi.edu/integration/Heracles/
[6] http://www.tripsay.com/

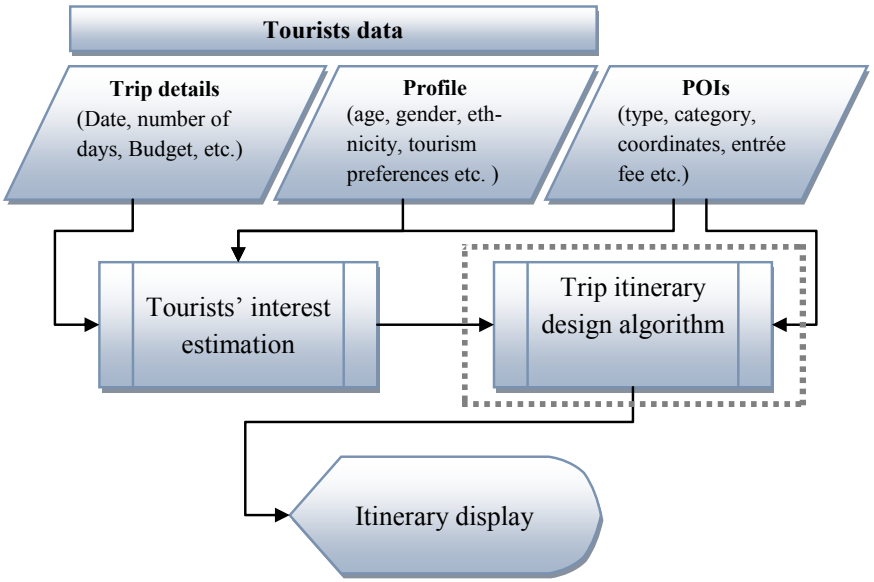


Figure 2.1: General block scheme of tourist trip itinerary design system

2.3 Tourist Trip Planning Systems

The problem of planning the trip itinerary for a tourist is known as "Tourist Trip Design Problem (TTDP)" [135]. The Tourist Trip Planning Systems (TTPS) that solve TTDP problems, are mainly based on tourist's personal interests about POI types (e.g. monument, tower, museum, etc.) and categories (e.g. archeology, architecture, classical art, etc.), on the information about POIs (e.g. opening hours, location, entry fee, etc.) and trip details (e.g. start date, duration, budget, etc.) located in the visited city or region. Figure 2.1 depicts the main components of a TTPS system. The tourist's interest estimation component estimates the match between the tourist preferences and POI features and based on that generates the score (satisfaction factor) values that reflect the satisfaction of the tourist with individual POIs. The trip designing algorithm uses the generated scores to design the itinerary so that the POIs with high scores have higher probability to be included into the itinerary.

As described earlier, there is a number of mobile and web based systems that enable designing the trip itinerary for a number of cities around the world, such as City Trip Planner, Your Tour, Plnnr, Trip Tern and Mtrip. These systems considers tourist preferences and constraints and the data about POIs to prepare a multiple day trip itinerary. In the simplest format, the generated trip itinerary would at least contain a list of sequenced visits to POIs that are annotated with respective start and end time, cost and traveling times between POIs. Moreover, the itinerary could include maps containing routes for visiting the scheduled POIs, navigation information for traveling between the consecutive POIs, descriptive information about POIs (e.g. textual, pictures, videos, augmented reality based services, etc.), etc.

Each of the systems, initially, utilizes some mechanism to estimate the tourist interest about

POIs, and then it applies some algorithm to plan the trip itinerary. Since, the focus of this PhD work is to deal with trip itinerary planning algorithms (see the component of trip itinerary design algorithm in Figure 2.1), in Section 2.6, we solely focus in describing the existing approaches that solve various versions of Tourist Trip Design Problem.

2.4 Social Relations

In social sciences, the social relations are defined as the relationship between any two or more persons. Kenny and La Voie [68] presents a social relations model for observing the relationship between two persons (suppose A and B) based on the effects caused by three different components, which are called *actor*, *partner* and *relationship*. The *actor* effect refers to the personality, skill and character of person A, whereas the *partner* effect refers to the amount of behavior that a person A elicits from person B. The *relationship* effect refers to situations when person A and B have already established a relationship, and an eventual action or response of person A to B, alters their social relation. The relationship effect between person A and B is directional. That is, adjustment effects of person A towards person B do not need to be the same as those of person B towards A. In addition, Berscheid [10], while defining the scope of "relationship science", states that "tissue" of a relationship " is the oscillating rhythm of influence observed in the interactions of two people". The oscillation of rhythm of relationship over the time, reflects that a relationship between two person is temporal rather than static. Finding the conditions that influence the rhythm of relationship between two persons is the core object of study of the field of "relationship science". In more practical terms, the interpersonal relationship between two or more persons might exist in different dimensions of their interaction, such as family, friendship, love, marriage, work, neighborhood, club, etc.

The presentation of social relations in form of a network was done by Zachary [143] in late 1970s. He described a social relations network between 34 members of a karate club within an university (see Figure [143]). The club members that have a social relation between each other outside their normal activities (karate classes and club meetings) are represented in the network of relations. The lines between the vertexes shows whether there exist a relation between two club members. The relation itself is considered to be symmetric (same level of relation between two members). The example shows that some members (p_1, p_{33} and p_{34}) have considerably more number of relations then the others, thus gathering certain members of the club around themselves and in that way forming groups. It is obvious that on one side p_1 can represent a group, whereas on the other side p_{33} and p_{34} can jointly represent another group. In addition, some other members (e.g. p_2) might act as "intermediaries" between the different groups.

Nowadays, many social network systems (e.g. Facebook, MySpace, Twitter, etc.) use social relation concepts to provide better services for their users. In such systems, besides provision of on-line communities of friends, users are also provided with facilities for on-line chatting, sharing of multimedia contents (pictures, audio, video), journal entries, etc. For the purpose of exploring the relations between the members in the on-line communities, there exist a number of methods that enable eliciting the social relationship level (closeness) between different community members. Wasserman [139] describes a variety of general methods from the field of social network analysis that can be used to elicit social relations between persons. Chun et al.

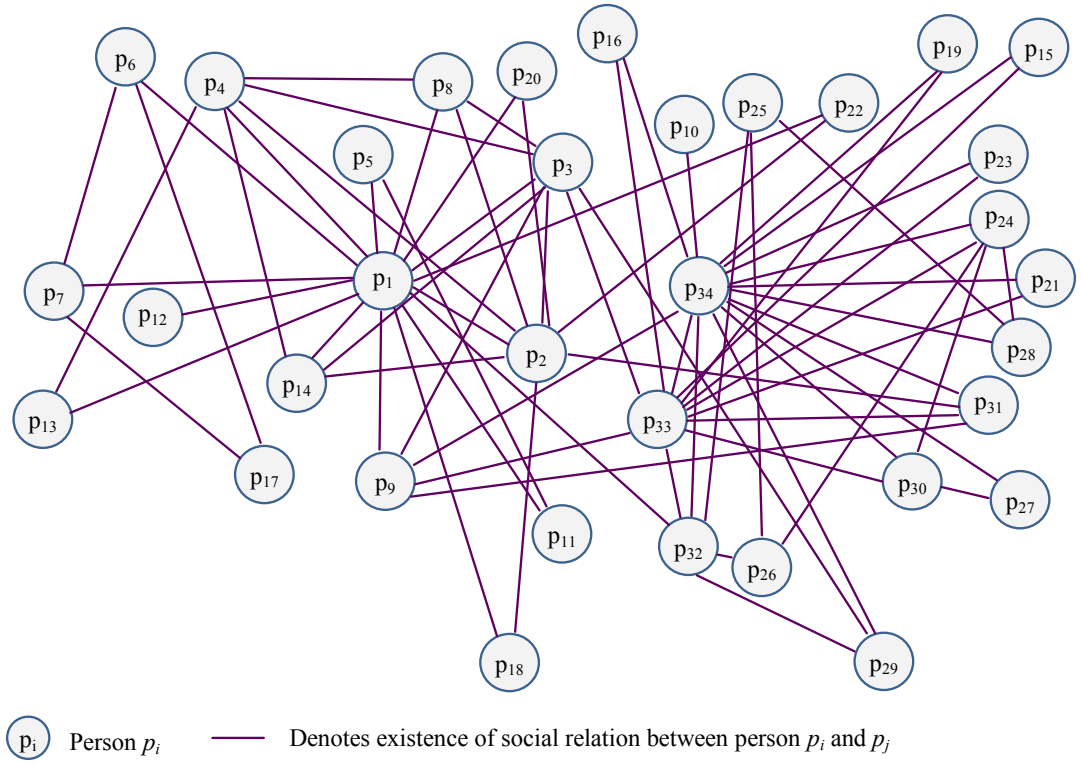


Figure 2.2: Social relations between 34 members of a karate club (adapted from [143]

[23] use some of the concepts from social network analysis to compare on-line social relations of users based on their activity and interaction.

In addition, more specialized systems and approaches that use social relations are presented in the literature in different domains of application. For instance, Guy et al. [60] present a recommender system that elicits social relations of user from social network systems, and recommends social software items (bookmarked web-pages, blog entries and communities) based on the social relations of users. Alshabib et al. [3] define ratings aggregation in recommender systems by introducing a model that combines context based ratings with the structure of a social network. Kwak et al. [72] use PageRank algorithm [94] to analyze social relations, with the aim of identifying users that are more influential in the community. Crampes et al. [26] present a method that visualizes social photos on Hasse diagram, which beside indexing new photos, can also be used for eliciting social relations between persons in photos. Nevertheless, Gretzel in [59], concludes that "most tourism research is still based on antiquated representations of the human mind as a protected storage container instead of a malleable entity that functions within a social context".

Ahmedi et al. [2] present a method for estimation of tourist interest about POIs based on concepts from the field of social network analysis (SNA) [31]. The tourist preferences about POI categories (e.g. archeology, architecture, nature, etc.) and her/his personal profile

(e.g. age, gender, profession, etc.) are compared against the preferences and profile of previous users (called "reviewers") by using a number of metrics from SNA. Then, the POIs that are visited/recommended by the reviewers that are in close relationship, in terms of preferences about POIs and personal profile, to the tourist, are proposed. The preliminary results on three test instances show that the method has an accuracy of more than 75% in estimating tourists interest about POIs.

To the best of our knowledge, except for the work of Ahmedi et al. [2] that uses social relations for estimating tourists interest on POIs, there is no research published in the literature that uses social relations within the algorithm of itinerary planning for a tourist group. Therefore, our approach for considering the social relationship of tourists in the algorithms for trip itinerary planning, represents a new dimension of research in the envisioned field.

2.5 Local Search Techniques

Before we give more insights about modeling Tourist Trip Planning Problems, we discuss, in general, about local search techniques that are used in solving highly complex combinatorial problems.

Local search techniques belong to the class of optimization techniques known with the name metaheuristics. The word "meta" before the word "heuristics" is used to indicate that such methods usually serve as a higher level frameworks for lower level search algorithms/heuristics (e.g. A* algorithm, Dijkstra's algorithm, Branch & Bound algorithm, etc.) or partial search algorithms.

Local search techniques focus on searching for a better solution in the neighborhood of a current solution, that is why some authors (e.g. [1] and [102]) refer to them with the name 'neighborhood search'. The search process begins with the construction of an initial solution, which is used as the starting solution by an improvement procedure that runs for a specified number of iterations. The initial solution is evaluated by using an objective/evaluation function that is defined based on the specific problem being solved. At a given iteration, the improvement procedure applies the so called operators (or moves) to transform the current solution into neighbor solutions. The neighborhood of a current solution is denoted with *N(S)* and often contains all neighborhood solutions that can be obtained by applying the predefined transformation operator (see Figure 2.3). The defined objective function is used to evaluate the neighbor candidates, where one or more of them are accepted as new current solution/s. The accepted solution is considered to be local optimal solution (local optimum) as it is the best found solution in the explored neighborhood. A local search technique usually iterates until an acceptable or optimal solution is obtained. A solution is considered to be optimal (global optimum) when it is the best solution in the solution space. In order to search for a global optimal solution, a local search procedure repeats a number of predefined iterations.

The procedure for construction of the initial solution is done by generating randomly a candidate solution or by using some heuristic algorithm in order to generate a good starting solution. More advanced techniques for construction of initial solution might hybridize randomization approach with the heuristic algorithm approach.

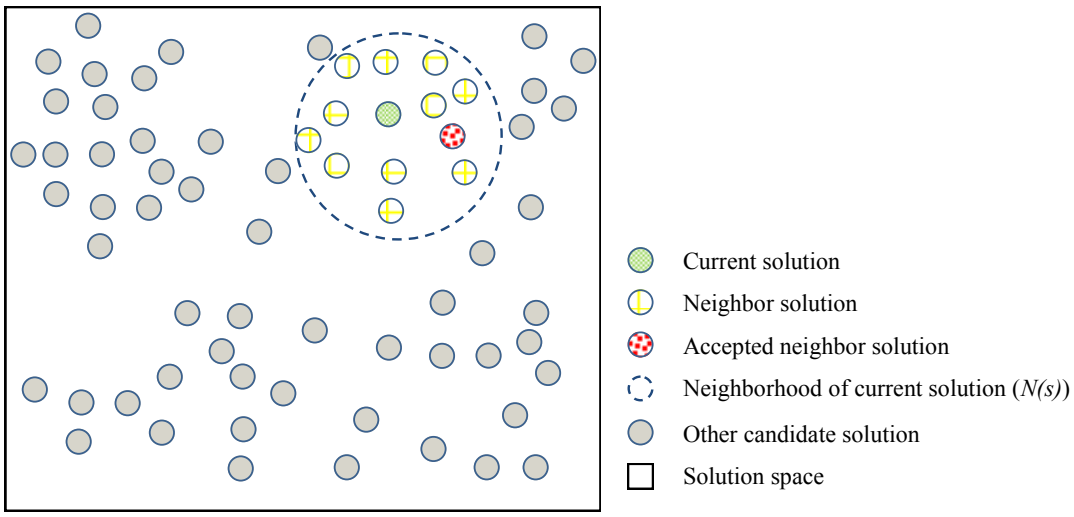


Figure 2.3: Sample neighborhood representation in local search techniques

In general, the difference between various local search techniques stands in the principles that they use to explore the neighborhood and on the way they make the decision for accepting the descendant of the current solution [92]. Some techniques apply complete search of neighborhood, whereas some other only explore a small subset of neighbors. Regarding the selection of the descendant of the current solution, in some techniques, a neighbor is accepted as current solution only if it better than the existing current solution, whereas on some other cases, a neighbor solution that is worse than the current solution might be accepted.

Another difference between various local local search techniques is that some of them work only with one current solution, whereas others use a number of current solutions concurrently. The earlier techniques are attributed as single solution exploration oriented methods, while the latter techniques are known as population based methods.

In the following, we discuss in more details about the objective function and the procedure of neighborhood exploration. Next, we shortly describe a number of frequently used meta heuristics, and then describe in more details the tabu search metaheuristic and some of its successful applications.

2.5.1 Objective Function

The objective function is used to evaluate the quality (also called fitness) of a solution being examined. Different problems use different objective functions when evaluating a candidate solutions. For instance, let us suppose a Traveling Salesman Problem (TSP) with $N=10$ points, where the goal is to find the shortest path for visiting all N points. The solution can be represented as an array that contains the sequence of points to be visited. An arbitrary current solution could be written as $S_c=\{4, 5, 1, 9, 10, 6, 2, 3, 7, 8\}$. In this case, the objective function would measure the travel distance needed to traverse the sequence of points in current solution S_c.

In general, there might be optimization problems where the aim is to find a solution with a minimal evaluation (such as the case of the presented TSP example), whereas in some other cases the goal might be to find a solution with a maximal evaluation (e.g. Utility Maximization Problem [120]). In such cases, the objective function would be called a minimization or maximization function, respectively. In addition, there exist objective functions that are used to measure a single feature of a problem, such as the measurement of traveling distance in the above described TSP problem, while there also exist multi objective functions that evaluate the fitness of solution by measuring two or more features of the problem being optimized. For instance, an example of multi objective optimization can also be the case of tourist trip planning, where the aim can be defined so that, in one side it is tried to include the POIs that have the highest score, and on the other side the minimization of the total traveling time for visiting the POIs is aimed. The evaluation of solution in multi objective problems is usually done by giving weights to the different components of the objective function, such as for instance the case of trip planning, where the objective function can be defined as $f = W_s * E_s + W_t * E_t$. In this case, E_s and E_t represent the evaluation components of satisfaction and travel time, whereas W_s and W_t represent the respective weights.

In some optimization problems, the calculation of the value of the objective function can be costly in terms of computation time. Since, in local search techniques, the evaluation of neighbor candidates of current solution needs to be done in every iteration, a straightforward implementation of the objective function might add unnecessary load into overall computation time. For instance, if in the presented TSP problem, a neighbor solution S_n is created by exchanging two of its points, calculating the whole traveling distance for traversing all points is not necessary. Instead, a deviation (delta) function can be used to calculate the traveling distance of S_n by adding the travel distance of the path that is added and subtracting the travel distance of path that is removed form the traveling distance of current solution S_c.

2.5.2 Neighborhood Exploration

The exploration of the neighborhood of a given current solution is made by using the so called operators (or moves). In order to describe the concepts standing behind operators (or moves), let us considers again the presented example of TSP problem and the given current solution S_c=*{4, 5, 1, 9, 10, 6, 2, 3, 7, 8}*. A possible operator in the current solution S_c would be swapping points in positions *i* and *j*. The neighborhood of the current solution S_c would contain all neighbor solutions that can be obtained by swapping points in position *i* and *j*, where *i* and *j* can take integer values from 1 to 10. If for example points in positions *i=2* and *j=5* are swapped, the corresponding neighbor of S_c is S_n=*{4, **10**, 1, 9, **5**, 6, 2, 3, 7, 8}*.

In general, different operators could be defined (for solving different problems). Some operators might be more general by being able to be applied to a range of problems (e.g. *k-Opt* for reducing travel time between to points), while some others might be specifically designed for specific problems. Principally, the operators with large neighborhood, where all possible combinations are tried out, have higher probability to obtain good quality solutions, but would require a longer computation time. On the other hand, the operators that do not to explore all possible neighbor are more quick in terms of computation time, but in general tend to produce worse solutions in terms of quality.

In addition, some operators explore only neighbors that satisfy the hard constraints (feasible neighbors) of the problem being solved. This has the advantage of limiting the number of neighbors that are explored, therefor reducing the computation time, while the disadvantage stands in the possibilities of getting stack in a particular area of search space, which might be disjoint from other feasible areas.

2.5.3 Meta Heuristics

In this subsection, we briefly describe some of the most utilized metaheuristic techniques, whereas in the next subsection, we present a detailed description of tabu search technique, because of its application in solving the envisioned problem of this PhD thesis.

Simulated Annealing (SA) - is based on theories from thermodynamics where creation of the structure of crystals depends merely on the cooling function, which should not be so quick or so slow as defects in the crystal might appear. Analogical, in the field of optimization, SA uses a "cooling function" to determine whether the algorithm should be able to also select random solutions that might be worse then the current solution, or it should only accept better solutions, and thus act more as a hill climber. The attribute of selecting solutions that are worse than the current solution, enables the SA algorithm to escape from getting stuck in local optima. This approach was initially presented by Kirkpatrick et al. in [69].

Iterated Local Search (ILS) - builds a sequence of local optimum solutions with the aim of storing previous search experience that is used to avoid getting stuck in local optima. At the start of a given iteration, one of the stored local optimal solutions is first perturbed/modified and then used as current solution in a local search procedure. The strength of this method stands at the way the perturbation/modification mechanism is implemented, which should enable searching only those areas of search space that seem to be promising in finding better solutions. This approach was first introduced by Lourenço et al. in [82].

Variable Neighborhood Search (VNS) - is attributed as a method that can explore neighbors of current solution that are distanced between each other. VNS accepts a new current solution only when it encounters a better one. The method iterates several times so that it does not get caught in local optima. The method was introduced by Mladenović and Hansen in [85].

Greedy Randomized Adaptive Search Procedure (GRASP) - during the course of an iteration performs two steps, namely construction and iterative improvement. The construction step creates a current solution by adding elements based on a ranking generated by a greedy function that ranks elements in accordance to the improvement they are expected to produce. The randomness is achieved by shortlisting the most promising elements, and then adding them randomly into the solution being constructed. The iterative improvement step undergoes a local search procedure to search the neighborhood of current solution. The method was first proposed by Feo and Resende in [37].

Guided Local Search (GLS) - enforces penalties for frequent moves to enable the local search process to escape from getting stuck in local optimum. GLS utilizes a problem specific mechanism for modification of the objective function when the algorithm get stuck in local optimum. The algorithm works with a modified objective function until it gets out of local optimum. The strength of this method stands on the way that the objective function is modified. GLS was originally proposed by Voudouris and Tsang in [136].

Large Neighborhood Search (LNS) - searches the solution space by applying complex neighborhood procedures based on problem specific heuristics. The exploration of large neighborhoods enables the algorithm to make a more systematic searching and thus make a better traversal of solution space. Different variants of LNS appear in literature, such as variability in the depth of neighborhood search or adaptivity of the size of neighborhood search. However, searching large neighborhoods might lead to larger computation times, hence various filtering techniques are enforced to avoid exploring neighbors that do not seem to result in any improvement. LNS was first introduced and applied by Shaw in [114].

Genetic Algorithms (GA) - are inspired by processes of nature evolution. The concepts such as inheritance, mutation, selection and crossover are used for implementing certain parts of the algorithm. The initial population of candidate solutions can be made randomly or based on some problem specific heuristic. GA maintains a population of candidate solutions throughout the evolution process, which usually undergoes a predefined number of generations (iterations). In a single generation, new members of population are created by altering the old members of population through application of mutation and crossover operators. A mutation operator creates a new candidate solution from a single existing solution, whereas a crossover operator creates a new candidate solution by combing two or more existing solutions. The first noticeable application of GA is made by Rechenberg in [101], although other authors have preliminarily used concepts of nature evolution for dealing with different optimization problems.

Ant Colony Optimization (ACO) - is inspired by the behavior of ants for finding paths from their colony to the sources of food. The process starts when some ants mark a new path to a source of food by laying down pheromone trails. Other ants follow the marked path for bringing food to their colony. As the path is used more and more by the ants, the pheromone trails get dissolved, hence new (most probably) shorter paths are created. As results, the new shorter paths are used by the majority of ants. In optimization techniques terms, the concept of "dissolving pheromone trails" is understood as escaping from areas of search space that appear to contain local optimum solutions, whereas the paths that are frequented more by the ants are taken as areas of search space that contain high quality solutions. ACO is a population based method, where a pheromone represents a candidate solutions. In general, more pheromones come from shorter paths of ants, whereas few pheromones are selected from longer paths, in order to retain a degree of diversification of search process. ACO was proposed by Colorni et al. in [24].

Particle Swarm Optimization (PSO) - is based on social behavior of living beings who move by following the movement of the majority of other living beings (particles) in the swarm. In optimization terms, the swarm of particles represents the population of candidate solutions. The swarm movement is dictated by the local best particle position and by the best known particle position in the environment of movement (search space). In addition, the velocity of swarm movement also impacts its movement direction. When new better positions in the swarm and in overall environment are identified, they are adopted so that the other members of the swarm start following it. In this way, the particles of the swarm are expected to move towards areas of environment that are likely to produce better solutions. PSO was developed by Eberhart and Kennedy [33] and Eberhart et al. [34].

Harmony Search (HS) - is motivated from the processes of a group of musicians/instruments who should be harmonized while playing their instruments. In the HS algorithm, a musician

is equivalent to a solution variable. Since a group of musicians (population of solutions) play instruments simultaneously, they all generate solution values concurrently with aim of generating best possible value (global optimum). HS is attributed as a method that can escape from local optimum. Some advanced variants of HS can work without input parameters. HS was introduced by Geem et al. in [47].

Memetic Algorithms (MAs) - are usually composed of a mix of algorithms, where a particular population based search method (e.g. Genetic Algorithms) is combined with local search methods and/or multi-agent systems. Hence, MAs belong to population based methods, where each member of population is a tentative solution for the problem under consideration. MAs frequently apply operators that are based on the acquired knowledge about specific problem (instance) being solved. Maintenance of diversity in the population of MAs can be difficult, thats why many practitioners use specific algorithms to preserve the diversity in the search process. MAs were first introduced by Moscato in [89].

Artificial Neural Networks (ANNs) - try to mimic biological nervous system, which functions through interaction of billions of neurones in the so called neural network. Hence, ANNs consist of a set of interconnected nodes (neurons) that have processing capabilities. The interconnection segments between the nodes are associated with weights, which are controlled and adapted during the learning/searching process. The nodes are usually divided into a number of communicating layers. In the simplest case, the nodes are organized into three layers, namely input nodes, output nodes and hidden nodes. The communication between different layers starts from the input layer and goes through the hidden layer up to the output layer. Usually, ANNs are characterized with three types of features: model of interconnection between different layers, the learning/searching process that updates interconnection segments, and the function used for transformation of input weight of a node to the corresponding output weight. ANNs are widely applied in the fields of machine learning and pattern recognition. Nevertheless, ANNs are occasionally applied as searching techniques in different domains of application (an example is shown in Subsection 2.6.1).

2.5.4 Tabu Search

The Tabu Search meta heuristic is a local search method that is used for solving highly constrained combinatorial problems. It was initially proposed by Glover and McMillan in [56] and then further formalized by Glover in [53]. The main principles of tabu search consist of storing information about previous search experience and based on that take actions either in intensifying the search process in direction of some promising search space area, or diversifying the search process towards less explored sections of search space. The search information is stored in the so called tabu memories, which keep information about moves that are forbidden or tabu for a certain period of time (or number of iterations). In general, tabu search uses three sorts of memories, namely short term, intermediate and long term memory.

The short term memory (also called recency memory) keeps record of the recency of application of a certain move (e.g. swap between to points in TSP problem) during the search process. This memory constitutes the core of the method, as it facilitates the search process to escape from getting caught in local optimum. The so called Tabu List Size parameter specifies the number of iterations a certain move can not be utilized (is tabu).

The intermediate and long term memory (also called frequency memory) keep record of how often certain moves are applied during the search process. The intermediate memory is used to intensify the search process in a promising section of search space by giving priority to frequently used moves. Conversely, the long term memory is applied when the diversification (exploration of rarely visited regions of search space) of search process is needed. This is done by prioritizing less frequent used moves.

The pseudo code in Algorithm 2.1 [84] shows the basic steps of tabu search method. After initialization of tabu memories/lists, a random initialization of initial solution is usually made, and then the evaluation of the initial solution is performed by using the respective fitness function of the problem at hand. The method runs for a number of iterations as specified by the actual termination condition parameter. At each iteration, the neighborhood of the current solution will be examined by means of different operators for the problem being solved. For instance, Knox [71] used the "2-interchange" operator for interchanging two vertexes to solve the TSP problem, while Gendreau et al. [49] used Remove and Insert operators for insertion and removal of a certain vertex from a route, when tackling the Vehicle Routing Problem. The best tabu and non tabu solution are always searched in the neighborhood of current solution. The tabu memories, both short term and long term, are consulted when deciding which best neighbor solution to adapt as next current solution. The first choice is always accepting the best non tabu neighbor, but if a good tabu neighbor that fulfills some "aspiration criteria" is found, then it is accepted as next solution for exploration. The concept behind the "aspiration criteria" is about allowing adaption of some possible tabu neighbors that are promising such as for example, a neighbor that is the best found ever during the search process or a neighbor that has unique properties (e.g. represents an unexplored area of search space) in comparison to the other neighbor candidates. The last choice is the acceptance of the best non tabu neighbor that might not necessarily be the best neighbor found. At the end of each iteration, respective memories are updated accordingly.

Depending on the implementation, the method might use various parameters, but a basic tabu search implementation uses at least four parameters as in following:

- Tabu List Size - Determines the number of iterations that a certain move can not be used (is tabu)
- Number of Iterations - Specifies the number of iterations the algorithm would run
- Penalty Coefficient - Used for penalizing certain moves when intensifying/diversifying the search process
- Memory Horizon - Determines the reset frequency of intermediate and long term memories

The data structures for implementation of memories might range from the most simple implementations like arrays and FIFO (First In First Out) lists up to some more advanced data structures like hash tables. For instance, as presented in [84], a possible memory for the TSP problem with N vertexes could be a square matrix with N-1 rows and columns. In this example, the individual matrix members would represent either the number of next iterations a swap between two points can not be performed (Figure 2.4 a) or the number of swaps that are made by

input : Method parameters
output: Solution

begin
  Initialize tabu list;
  Generate randomly Initial Solution S_c;
  Evaluate S_c;
  while *termination condition not met* **do**
    Generate all neighbor candidates of solution S_c;
    Find best candidate S_x in neighborhood of solution S_c;
    if S_x *is not tabu solution* **then**
      $S_c = S_x$;
    else if *aspiration criteria is fulfilled* **then**
      $S_c = S_x$;
    else
      Find best not tabu solution in the neighborhood S_{nt};
      $S_c = S_{nt}$;
    end
    Update tabu list;
  end
end

Algorithm 2.1: Pseudo code of Tabu Search [84]

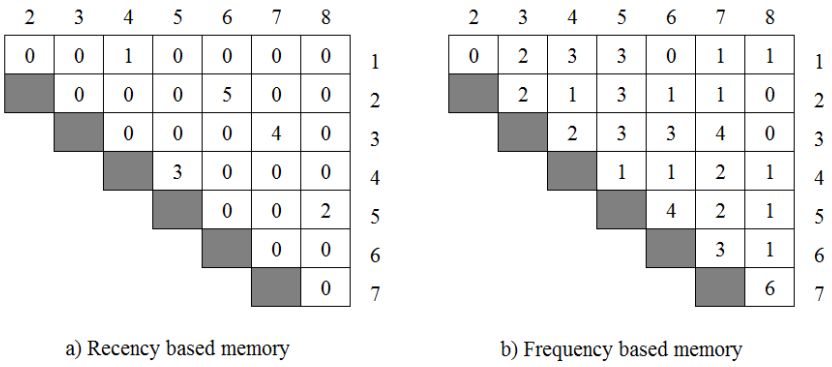


Figure 2.4: A sample tabu memory for TSP problem

any two points during the search process (Figure 2.4 b). In [28], authors used a FIFO memory strategy when applying tabu search to solve job scheduling problem.

In [65], a special memory structure is used which consists of a hash table and a self height balancing trinary (three branch) tree. The explored solutions are indexed into the hash tables and in the tree based on their objective function value. This enables a fast scanning process for the best solutions that are placed into the tree and are pointed to by the hash table.

Some advanced implementations of tabu search utilize a variable tabu list size throughout the

search process. Battiti and Tecchiolli [8] are the first to introduce the reactive tabu search where the appropriate tabu list size is derived automatically. In addition, they showed that this kind of technique could be implemented in a trivial way by using hash tables and digital trees. The advantages of reactive tabu search implementation were demonstrated by doing a comparison to static format of tabu memory for the Knapsack and Quadratic Assignment Problem.

In [108], Rochat and Taillard present a tabu search implementation for Vehicle Routing Problem enriched with probabilistic features. The diversification, intensification and parallelization mechanisms are relied on strategic embodiment of probabilistic elements rather than use of simple randomized features. As concluded by the authors, the probabilistic technique is suitable for implementation for any local search method by making it more robust, and hence, enabling it to converge more often toward solutions that have more quality and are close to best known solutions.

2.5.5 Applications of Tabu Search

Tabu search is successfully applied in solving, (near) to optimality, many problems in different domains of application such as scheduling, computer channel balancing, cluster analysis, space planning, assignment, etc. [54]. In the following, we report on some successful applications of tabu search in some of the above mentioned domains.

In [11], Blazewicza et al. compare tabu search with Simulated Annealing (SA) and Variable Neighborhood Search (VNS) when solving two-machine flow-shop problem with weighted late work criterion and common due date. The efficiency of meta-heuristics was evaluated by using 20 problem instances. The fine tuning of tabu search parameters yielded to an improvement performance of about 7 %. Nevertheless, this study showed that SA approach could generate better schedules in a shorter time.

In [121], Subrata et al. discuss application of tabu search and Genetic Algorithms (GA) in load balancing in computational grids. Their results show that tabu search and GA achieve similar results, while, on the other hand, both methods outperform other tested algorithms like Best-fit, Random, Min-min, Max-min, and Sufferage. This is further more obvious when the number of iterations the algorithm runs is increased. The reported drawback is that both tabu search and GA need more storage space and processing power in the scheduling mechanism.

Sung and Jin [122] apply tabu search in clustering a data set into natural and homogeneous subsets such that each subset is composed of elements that are similar to one another but different from any other subset. The efficiency and effectiveness of the tabu search implementation is further extended by introducing a procedure that comprises of a packing and a releasing step. The task of packing step is to bind a subset of elements together, while the releasing step tries to separate any packed elements form each other. The numerical testing of the proposed method shows that, in overall, it outperformed other implementations existing priorly, such as basic tabu search, K-means and simulated annealing. The authors outline different possible applications of the presented algorithm, such as patient classification, product distribution center allocation, and government service branch organization.

Gendreau et al. in [50] presents an approach based on tabu search for solving the ambulance location problem. The problem modeling is done in accordance to the rules set by the United States Emergency Medical Services Act of 1973. The actual implementation uses the funda-

mental tabu search features for escaping from the local optimum, intensifying and diversifying the search process. The algorithm is tested on a set of random instances, as well as on instances generated from the data derived from the Island of Montreal. It is concluded that the tabu search implementation achieves near to optimal solution in a relatively short ("modest") computation time.

In [75], Laguna et al. describe a tabu search approach for solving multilevel generalized assignment problem (MGAP), which could be used for solving practical problems in manufacturing such as lot sizing (finding the optimal combination for achieving maximal production). The tabu search implementation is powered by using a neighborhood search procedure based on "ejection chains", that allow more effective and efficient use of machine capacities. The "ejection chains" were first introduced by Glover in [55]. Based on authors, the algorithm has shown high level performance when tested on difficult Single - level GAP instances. In addition, the authors claim that the proposed method does not significantly change its performance if the parameter values are refined. At the time of paper publication, this implementation was a substantial move ahead in comparison to the other existing scientific and commercial approaches. For more detailed information about application of tabu search for the domain of assignment problems and others, the reader is referred to [91].

2.6 Modeling Tourist Trip Planning Problems

The modeling of tourist trip planning problem depends on the kinds of constraints that are considered when solving a particular problem. Figure 2.5 shows possible modelings to the tourist trip planning problem. The most basic modeling is the game called "Orienteering Problem" (OP) [129], which is played in a place that has a number of points, each associated with a score. Due to the limited time, not all points can be visited, and the goal is to visit those points that maximize the total collected score. A point can be visited at most one time. It is obvious that OP can be used to model a single day (period) trip with a limited duration, where the score associated with points would represent the satisfaction factor of a tourist with the POI. Note, that the selective traveling salesman problem (STSP) [77] is related to OP, except it also considers compulsory points to be included in the trip.

Various additional constraints could be modeled to the trip planning problem when other extensions of OP are applied. The operating/working hours of POIs in tourist trip planning problems could be modeled by applying OP with Time Windows (OPTW) [66], whereas multiple day (period) trip could be modeled by using Team OP (TOP) [21]. The TOPTW [134] models both, multiple tours and operating hours, at the same time. The constraints about maximum allowed budget to be spent and maximum allowed POIs of certain type or category to be visited, could be modeled by utilizing Multi Constraint TOPTW (MCTOPTW) [43]. Different means of traveling between POIs (e.g. walking, bicycle, public transport, etc.) could be modeled by using Time Dependent OP (TDOP) [39]. In case the operating hours of POIs need to be taken into account, the Time Dependent OPTW (TDOPTW) has to be considered. In some occasions a POI is characterized with a number of features (e.g. beauty, cultural background, historical relevance, etc.) and as result a range of scores for each of the features is associated with that POI. This situation can modeled by using Generalized Orienteering Problem (GOP)

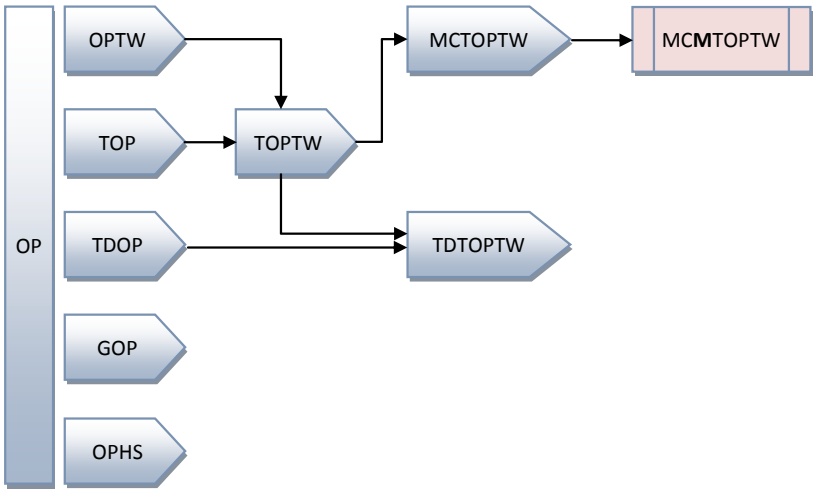


Figure 2.5: Theoretical models of tourist trip planning problems

[99], which allows modeling multiple score for each POI. In addition, based on the location of last visited POI in the trip, an accommodation could be selected automatically at the end of the each trip. This problem extension is coined as OP with Hotel Selection (OPHS) [30].

To the best of our knowledge, there is no model published in the literature that can be used for modeling the trip planning problem for a group of tourists. Since, an itinerary for each tourist in the group needs to be planned, we propose to extend MCTOPTW to Multi Constraint **Multiple** Team Orienteering Problem with Time Windows (MC**M**TOPTW). This would define a new extension to OP, in which multiple trips become possible, and where each trip would have multiple tours.

In the following subsections, we present the mathematical modelings and the existing approaches from the literature for solving the OP and its respective extensions. Further, related approaches for solving similar problems to OP (e.g. STSP) are discussed. The related work that is presented in the remaining part of this section, is primarily based on survey papers of Vansteenwegen et al. [133] and Gavalas et al. [45].

2.6.1 Orienteering Problem

In OP, a set of N points i is given, each associated with a score S_i. The starting point (i=*1*) and the ending point (i=N) of the tour are fixed. Each point i can visited at most once. The traveling time between points i and j is known for all i,j=*1, ..., N*. The duration of the tour is limited to the maximum value T_{max}, therefore often not all points can be visited. In the time frame specified by T_{max}, the aim is to find a route that passes by those points that maximize the total collected score.

In the following, we present the OP formulated as an integer programming problem [133]. The boolean variable x_{ij} is set to *1* when a visit to point i is followed with a visit to point j,

otherwise x_{ij} is set to *0*. The integer variable u_i shows the index of visit to point *i* out of all selected visits throughout the route.

$$Max \sum_{i=1}^{N-1} \sum_{j=2}^{N} S_i x_{ij}, \tag{2.1}$$

$$\sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1, \tag{2.2}$$

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1; \ \forall k = 2, ..., N-1, \tag{2.3}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max}, \tag{2.4}$$

$$2 \leq u_i \leq N; \ , \forall i = 2, ..., N, \tag{2.5}$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}); \ \forall i, j = 2, ..., N, \tag{2.6}$$

$$x_{ij} \in \{0, 1\}; \ \forall i, j = 1, ..., N, \tag{2.7}$$

The objective function of OP (Constraint 2.1) is maximization of the total collected score. As formulated by Constraint 2.2, the tour should always start at point *1* and end at point *N*. Equation 2.3, on one side enforces the tour connectivity by constraining current visit to point *i* with a next visit to point *j*, and on the other side it allows maximum one visit to a given point. Constraint 2.4 sets tour's maximum allowed time budget to T_{max}. Constraints 2.5 and 2.6 make sure that no subtours occur throughout the tour.

Since OP is a NP hard problem [57], exact algorithms are very computation intensive, and therefore highly time consuming. Hence, for practical implementations, heuristic approaches are considered. In addition, Gendreau et al. [52] emphasizes that the score of any point and the traveling time to go to that point, can be often contradictory. In this regard, straightforward heuristics might lead the search process in getting stuck into local optimum, and as result make the search process avoid the global optimum. Further, it is indicated [130] that instances that enable selection of around half of existing points, are the most difficult to solve, since the highest number of combinations are to be computed. In addition, Vansteenwegen [130] shows that as the number of selected points increases, it becomes more difficult to obtain the shortest path in the tour.

In regard to exact approaches, Laporte and Martella [77] and Ramesh et al. [100] apply a branch-and-bound algorithm to solve STSP, respectively OP, while Gendreau et al. [51] and Fischetti et al. [38] use branch-and-cut algorithm to solve STSP, respectively OP. Recently, Gavalas et al. [45] survey a number of approximation approaches that tackle OP and its variants

with high complexity. Further, Gavalas et al. [45] outline additional details about possibilities to approximate OP and its variants.

In order to consider more realistic situations, for practical applications, a number of approaches based on heuristic methods has been proposed. In the following, we discuss some of the most break through approaches in the literature.

Tsiligirides [129] presents two heuristic approaches for OP. The first uses stochastic algorithms based on Monte-Carlo techniques to initially create multiple tour solutions and then select the best tour based on the total collected score. The second one is a deterministic algorithm that searches for tour solutions in different local geographic regions that are limited inside circles that have different diameters.

Golden et al. [57] introduces a three step approach based on the center of gravity heuristic. The first step creates tours by inserting points based on their score and on the time consumption on the tour. The second step initially applies *2-Opt* operator to arrange the sequence of points in a way that the total consumed time gets minimized, and then it tries out to insert new points based on minimal time consumption heuristic. The third step applies the center of gravity heuristic by inserting a new point based on the ratio of the score of point over the distance to the center of gravity that is calculated by considering the previous tour. In addition, Golden et al. [58] further improve their method by also adding the concepts of randomness, sub gravity and learning.

Ramesh and Brown [99] apply a four-phase heuristic for the generalized OP. The first phase keeps the time budget constraint relaxed while inserting new points into the tour. The second phase applies *k-Opt (k=2,3)* operator with the aim of narrowing down the consumed time in the tour. In the following phase, the time budget constrained is enforced by deleting extra points from the tour. In the fourth phase, if room is available in the tour, new points are tried to be inserted. This procedure repeats for a number of iterations as specified by the corresponding control parameters of the algorithm.

OP is also solved by using an approach from artificial neural networks [137]. A fourth order convex energy function is used along with a learning algorithm, and both are put into work by using a Hopfield based neural network.

Chao et al. [20] apply a five phase heuristic by considering only the points that lie inside a defined ellipse in a Euclidean space. The foci of the ellipse is represented by the start point and end point, while, for the major axis of the ellipse, the time budget value T_{max} is adapted. In the first phase, the initial solution is created by considering a pool of candidate solutions that are created with a cheapest point insertion heuristic. In order to achieve the required diversity into the pool of candidate solutions, a single candidate starts with point far away from the start point and ends with a point far away from the end point. The best candidate from the pool of solutions is accepted as the current solution. In the second phase, the a two-point exchange operator is applied between points included in the current solution with points to other less evaluated candidate solutions. In the third phase, a move operator will try to move a point from one solution to another solution in the pool of candidate solutions. Next, in the fourth phase the *2-Opt* operator tries to reduce the consumed time in the current solution. In the last phase, some points that have a low ratio of score over the insertion cost are removed from the current solution. The five phases are repeated for several times until the termination condition of the algorithm is met.

A tabu search approach is applied by Gendreau et al. [52] through a repetitive procedure for insertion of cluster of points and removal of sequence of points. This approach has advantages of escaping from local optima and abilities to include points with high scores that might be located far from the points currently inserted into the tour.

Schilde et al. [110] introduce two approaches for the multi objective OP, which are also applied for the single objective OP. The first approach is a Pareto ant colony optimization algorithm that uses three basic operators for iterative improvement of candidate solution, namely *2-Opt* (to shorten the time consumption of the tour), *insert* (to insert a new point into the tour) and *exchange*. The second approach is a Pareto variable neighborhood search algorithm that consists of two main phases, namely *shaking* (based on a two point exchange move) and iterative improvement (based on *2-Opt*, *insert* and *exchange*). In addition, both of the approaches are hybridized with a path relinking method.

2.6.2 Team Orienteering Problem

In the TOP [21] [126], the objective is to find a number of tours M, each limited to T_{max} that visit a subset of points so that the total collected score is maximized. TOP is equivalent to Multiple Tour Maximum Collection Problem introduced by Butt and Cavalier [16].

TOP can be formally described as an integer programming problem [133] by utilizing two boolean variables and one integer variable. The boolean variable x_{ijm} is set to *1* when in tour m a visit to point i is followed with a visit to point j, otherwise x_{ijm} is set to *0*. The boolean variable y_{im} is set to *1* if point i is visited in tour m, otherwise x_{ijm} is set to *0*. The integer variable u_{im} shows the index of visit to point i in tour m.

$$Max \sum_{m=1}^{M} \sum_{i=2}^{N-1} S_i y_{im}, \tag{2.8}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{N} x_{1jm} = \sum_{m=1}^{M} \sum_{i=1}^{N-1} x_{iNm} = M, \tag{2.9}$$

$$\sum_{m=1}^{M} y_{im} \leq 1;\ \forall i = 2, ..., N-1, \tag{2.10}$$

$$\sum_{i=1}^{N-1} x_{ikm} = \sum_{j=2}^{N} x_{kjm} = y_{km},\ \forall k = 2, ..., N-1,\ \forall m = 1, ..., M, \tag{2.11}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ijm} \leq T_{max},\ \forall m = 1, ..., M, \tag{2.12}$$

$$2 \leq u_{im} \leq N;\ ,\forall i = 2, ..., N,\ \forall m = 1, ..., M, \tag{2.13}$$

$$u_{im} - u_{jm} + 1 \leq (N-1)(1 - x_{ijm});\ \forall i, j = 2, ..., N,\ \forall m = 1, ..., M, \tag{2.14}$$

$$x_{ijm}, y_{im} \in \{0,1\};\ \forall i,j = 1,...,N,\ \forall m = 1,...,M, \tag{2.15}$$

The objective function of TOP (Constraint 2.8) is maximization of the total collected score in all tours. Constraint 2.9, ensures that each tour starts at point *1* and ends at point *N*. Equation 2.10 makes sure that a point can be maximally visited once at any tour. Equation 2.11 enforces the tour connectivity by constraining current visit to point *i* in tour *m* with a next visit to point *j* in the same tour. Constraint 2.12 limits the maximum allowed time budget to T_{max} for any tour *m*. Equations 2.13 and 2.14 make sure that no subtours occur in any of the tours.

But and Ryan [17] solve TOP by using an exact algorithm based on a procedure for column generation and constraint branching. In addition, Boussier et al. [13] also take the approach of exact algorithms by presenting a branch-and-price approach, which is a coupling of column generation and branch-bound algorithms. The algorithm includes some specific branching rules and acceleration techniques to deal with particularities of the orienteering problem. The branching rules are about points and arcs between them. The acceleration techniques include a halting strategy for reducing the computation time, application of limited discrepancy search (LDS) heuristic [61] to help search towards promising points, and *label loading* and *meta extensions* techniques that are used to speed up the algorithm for column generation.

Chao et al. [21] make two modification in their five phase approach [20] for OP to use it to solve TOP. Instead of returning the best tour from the pool of candidate solutions, the best *M* tours are returned and the frequency of re-initialization in the algorithm is doubled.

Tang and Miller-Hooks [126] tackle TOP by using a tabu search approach that is embedded in an Adaptive Memory Procedure (AMP). The search process of the approach could be divided into three main phases, namely initialization, improvement and evaluation of solution. In the initialization phase, the algorithm parameters are tuned to allow a rather narrow neighborhood of searching for creating the initial solution. The improvement phase is characterized with switching between large and small neighborhood modes, consideration of feasible and infeasible candidate solutions, and enforcement of random and greedy methods for selection of best neighbor solutions. In the evaluation phase, as usual for tabu search, the best non tabu solution is selected as next solution, whose neighborhood would be explored in the next iteration. The size of the neighborhood in the next iteration depends on current solution neighborhood size and quality. AMP combines different existing candidate solutions for preparing the initial solution for tabu search.

Archetti et al. [4] approach the problem by using tabu search and variable neighborhood search (VNS). In both approaches, the non-included points are organized in form of regular tours. Authors develop two variants of tabu search, one that accepts infeasible solutions and the other one that works only with feasible solutions. Both tabu search variants use two operators, namely *1-Move* that moves one point from a tour to the other, and *swap* that swap one point of a tour with one point of some other tour. A jump operator, used in both variants, swaps two sets of points from the tour with two sets of points from tours with non-included points. The infeasible solutions are corrected by a number of correction procedures. The VNS approach also comes in two variants, the slow variant that is intended for a high quality solutions and the fast variant that is intended for quick computation time. Both VNS variants, use tabu search (variant that considers only feasible solutions) as local search method, but with much less number of

iterations. In addition, *2-Opt* operator is used, in both VNS variants,to reduce the tour time consumed. In average, the approaches presented by Archetti et al. [4] perform better than the approaches described earlier.

Ke et al. [67] introduce an Ant Colony Optimization (ACO) approach for TOP. This approach constructs the candidate solutions based on four methods, namely, the sequential (tours are filled up with points sequentially), deterministic-concurrent (a point is inserted in one of the tours according to some fixed order), random-concurrent (a point is inserted in one of the tours in a random order) and simultaneous (at each iteration a point is inserted in one of the tours). The local search procedure consists of a *2-Opt* operator that tries to shorten the time consumed in the tour and of an *insert* operator that tries to insert as many feasible points as possible. This local search procedure is repeated until no improvement can be obtained in the searched neighborhood. At each iteration, each ant constructs a feasible candidate solution and the described local search procedure is applied to improve the solution. Afterwards, the pheromone trails are updated. The algorithm stops iterating when a predefined maximum number of iterations has been reached. According to [133], the results obtained by this method are as good as results achieved by Archetti et al. [4], but with less computation effort. This is specially the case of ACO approach with sequential tour completion.

Vansteenwegen et al. [131] [132] present two approaches for solving TOP with the aim of obtaining quality solutions in a short period of computation time. The first approach is based on Guided Local Search (GLS) framework, whereas the second approach is based on Skewed Variable Neighborhood Search (SVNS) framework. The GLS approach uses basic operators like *Insert, Replace, 2-Opt* and *Swap*, while the SVNS approach, in addition to this basic orators, uses some extra operators like *TwoInsert, TwoReplace* and *Change* [133]. Both approaches implement two variants of diversification and intensification mechanisms. The idea of removal of sequence of points from the tour constitutes the both variants of the mechanism for diversification of search process. The first variant of intensification mechanism tries to increase the overall score, whereas the second variant tries to minimize the spent time in a tour. The experimental results [132] show that SVNS outperforms GLS, both, in terms of solution quality and computation time.

Souffriau et al. [118] present an approach to solve TOP that is based on Greedy Randomized Adaptive Search Procedure enhanced with a Path Relinking (PR) method. This method comes in two variants, the slow variant intended for high quality solutions, and the fast variant intended for quickly generated near to high quality solutions. At each iteration, GRASP with PR performs four steps, namely construction of initial solutions, application of local search procedure, application of PR method and the update of the "pool of elite solutions". The generation of initial solution could be a random or a greedy one, as determined by the so called "greediness" parameter, which prescribes a precise probability ratio between greediness and randomness. The local search procedure alternates between reducing the total spent time in all tours and maximizing the total collected score. At each iteration, the local search procedure applies the operators of *2-Opt, Swap, Replace* and *Insert*. The Path Relinking method introduces a long term memory called "pool of elite solutions", which contains the best solutions found during the search process. This memory is used with the aim of avoiding the complete independence of creating the initial solution in different executions of the original GRASP procedure. The PR method

considers the candidate solutions in the search space that can be virtually composed by using current local search solution and each solution residing the memory of "pool of elite solutions". The fourth step updates the memory of "pool of elite solutions" with newly found good solutions. According to [133], the slow variant of GRASP with PR obtains comparable results to approaches of Archetti et al. [4] and Ke et al. [67].

Bouly et al. [12] propose a memetic algorithm (MA) to solve TOP. The algorithm uses an Optimal Splitter developed by using a modified version of the Program Evaluation and Review Technique/Critical Path Method (PERT/CPM). In addition, a heuristic for initialization of population termed "Iterative Destruction/Construction Heuristic" (IDCH) has been proposed, which is based Construction/Destruction principles described by Ruiz and Stuezle [109], combined with priority rules and local search techniques. A small part of initial solution is created by using IDCH technique, whereas the major part is created randomly. At each iteration, a couple of parents is selected among the population by using the Binary Tournament technique. The LOX [96] crossover operator is used to produce a child chromosome, while newly created chromosomes are evaluated by using the Optimal Splitter procedure. The population is updated with new chromosome, while its size remains constant and the redundancy between chromosomes is avoided. The chromosomes with higher score and lower travel time would constitute the population. A child chromosome has a certain probability to get mutated by using a set of local search techniques, namely *shift, swap* and *construct/destruct* operators. The stop condition of the algorithm is bound to a predefined number of iterations without improvement.

Dang et al. [27] present a Particle Swarm Optimization (PSO) approach based on Memetic Algorithms (PSOMA) for tackling TOP. The algorithm uses the techniques of Optimal Splitting, IDCH and local search operators *shift* and *destruction/repair*, which are also used in the algorithm of Bouly et al. [12]. Moreover, a genetic crossover operator is used to update the positions of particles in the swarm. In order to avoid a premature convergence of best local particles in the swarm, an update procedure based on set of specific rules [112] is applied. The major part of particle positions in the swarm, including local best positions, are initialized in a random sequence. In order to accelerate the algorithm, a small part of the particle swarm have their positions generated by using IDCH technique. The algorithm terminates when a predefined number of iterations have not produced any improvement in the new local best solution. The experimental results show that the proposed approach is competitive with priory introduced approaches for TOP.

More insight about the performance of the approaches presented above (except for Bouly et al. [12] and Dang et al. [27]) that solve TOP can be found in Vansteenwegen et al. [133], which summarize a comparison between respective approaches based on 157 benchmark instances for this problem.

2.6.3 Orienteering Problem with Time Windows

In OPTW, each point is associated with a time window determined by an opening time O_i and a closing time C_i. This time window constraints the start of visit in between its specified boundaries. Based on notations for OP and TOP, the OPTW can also be expressed as integer programming problem [133] by using the following variables: x_{ij} is set to *1* when a visit to

point *i* is followed with a visit to point *j*, otherwise x_{ij} is set to *0*; t_{ij} represents the traveling time between point *i* and point *j*; s_i start of the visit at point *i*; *Q* is a large constant.

$$Max \sum_{i=1}^{N-1} \sum_{j=2}^{N} S_i x_{ij}, \tag{2.16}$$

$$\sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1, \tag{2.17}$$

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1; \ \forall k = 2, ..., N-1, \tag{2.18}$$

$$s_i + t_{ij} - s_j \leq Q(1 - xij); \ \forall i, j = 1, ..., N, \tag{2.19}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max}, \tag{2.20}$$

$$O_i \leq s_i; \ \forall i = 1, ..., N, \tag{2.21}$$

$$s_i \leq C_i; \ \forall i = 1, ..., N, \tag{2.22}$$

$$x_{ij} \in \{0, 1\}; \ \forall i, j = 1, ..., N, \tag{2.23}$$

The objective function of OPTW (Constraint 2.16) is maximization of the total collected score. Constraint 2.17, limits the start of the tour at point *1* and the end at point *N*. Equation 2.18, on one side enforces the tour connectivity by constraining current visit to point *i* with a next visit to point *j*, and on the other side it allows maximum one visit to a given point. Constraint 2.19 ensures that the assigned time slots of visits to points do not overlap between each other or with traveling times between them. Constraint 2.20 sets tour's maximum allowed time budget to T_{max}. Constraints 2.21 and 2.22 limit the start of the visit to a point only in margins of the respective time window.

Kantor and Rosenwein [66] are the first to deal with OPTW. They developed a depth-first-search algorithm for construction of partial tours. The insertion of points into the tour is done by considering the heuristic function that is calculated as ratio between score of the point and the respective required time for insertion. The point time window is taken into account when it is inserted into the tour. The algorithm prunes the search process from the areas of search space where it encounters infeasible solutions or where current solution quality worsens.

Righini and Salani [106] solve OPTW by using a bi-directional and bounded dynamic programming algorithm. The bi-directional search procedure is implemented by extending the number of explored states in two opposite direction, going forward from start point and coming backward to end point. The extension of explored states is stopped at stage, when it is guaranteed

that the current state belongs to the optimal tour, and the remaining part of the the tour has been or will be generated in the opposite direction of search. In addition, if needed the search process for an optimal tour might continue as long as the critical resources do not surpass the half of the overall quantity of available resources. Further, in a second variant of the algorithm, Righini and Salani [107], with the aim of decreasing the number of explored states, apply the mechanism of Decremental State Space Relaxation (DSSR).

Gavalas et al. [45] describe a number of approximation algorithms for some variants of OPTW that are proposed in the literature. Bansal et al. [6] present a $O(log^2{}_n)$ approximation algorithm for Vehicle Routing Problem with Time Windows (VRPTW). Chekuri et al. [22] give a $O(log^3 OPT)$ approximation algorithms for TSP with time windows. Lately, Frederickson et al. [40] present an approximation algorithm for Traveling Repairman Problem (TRP). The algorithm has a 3-approximation ratio and a running time of $O(n^4)$ when the input graph is a tree. If the input is a general graph, the approximation ratio is (6+ϵ). and the running time is $n^4 \cdot n^{O(1/e^2)}$. Note that in above notations, variable n represents the number of nodes.

2.6.4 Team Orienteering Problem with Time Windows

In TOPTW, there are M tours for visiting a subset from N available points, each associated with a time window that is determined by an opening time O_i and a closing time C_i. This time window constraints the start of visit in between its specified boundaries. Based on notations for TOP and OPTW, the TOPTW can be formulated as an integer programming problem [133] by using the following variables: x_{ijm} is set to *1* when in tour m a visit to point i is followed with a visit to point j, otherwise x_{ijm} is set to *0*; y_{im} is set to *1* if point i is visited in tour m, otherwise x_{ijm} is set to *0*; t_{ij} represents the traveling time between point i and point j; s_{im} start of the visit at point i of tour m; Q is a large constant.

$$Max \sum_{m=1}^{M} \sum_{i=2}^{N-1} S_i y_{im}, \tag{2.24}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{N} x_{1jm} = \sum_{m=1}^{M} \sum_{i=1}^{N-1} x_{iNm} = M, \tag{2.25}$$

$$\sum_{m=1}^{M} y_{im} \leq 1;\ \forall i = 2, ..., N-1, \tag{2.26}$$

$$\sum_{i=1}^{N-1} x_{ikm} = \sum_{j=2}^{N} x_{kjm} = y_{km};\ \forall k = 2, ..., N-1;\ \forall m = 1, ..., M, \tag{2.27}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ijm} \leq T_{max};\ \forall m = 1, ..., M, \tag{2.28}$$

$$s_{im} + t_{ij} - s_{jm} \leq Q(1 - x_{ijm});\ \forall i, j = 1, ..., N;\ \forall m = 1, ..., M, \tag{2.29}$$

$$O_i \leq s_{im};\ \forall i = 1, ..., N;\ \forall m = 1, ..., M, \tag{2.30}$$

$$s_{im} \leq C_i;\ \forall i = 1, ..., N;\ \forall m = 1, ..., M, \tag{2.31}$$

$$x_{ijm}, y_{im} \in \{0, 1\};\ \forall i, j = 1, ..., N;\ \forall m = 1, ..., M, \tag{2.32}$$

The objective function of TOPTW (Constraint 2.24) is maximization of the total collected score in all tours. Constraint 2.25, ensures that each tour starts at point *1* and ends at point *N*. Equation 2.26 makes sure that a point can be maximally visited once at any tour. Equation 2.27 enforces the tour connectivity by constraining current visit to point *i* in tour *m* with a next visit to point *j* in the same tour. Constraint 2.28 limits the maximum allowed time budget to T_{max} for any tour *m*. Constraint 2.29 ensures that the assigned time slots of visits to points, in each tour *m*, do not overlap between each other or with traveling times in between them. Constraints 2.30 and 2.31 limit the start of the visit to a point only in margins of the respective time window.

Montemanni and Gambardella [86] solve TOPTW by using an Ant Colony System (ACS) approach that models the solution based on a hierarchic generalization of the original problem. The generalized problem is called Hierarchical TOPTW (HTOPTW), which requires the computation of an ordered list of non-overlapping elementary tours. The first *m* tours belong to the original TOPTW problem, whereas the remaining tours are ordered hierarchically and contain the points that are not part of solution for TOPTW. The organization of tours in form of hierarchy helps in keeping fragments of tours somehow pre- optimized. Such tour fragments are used by local search operators to perform exchanges/insertions of points, aiming at improving the quality of the hierarchy of tours. Further, representation of an ant as a giant tour [41], makes the HTOPTW problem more similar to traveling salesman problem. This enabled the authors to borrow the construction phase from their prior approach for Vehicle Routing Problem with Time Windows (VRPTW). The approach outperformed the algorithm of Mansini et al. [83] for the instances of OPTW.

Vansteenwegen et al. [134] tackle TOPTW by developing their approach based on the Iterated Local Search (ILS) framework. The algorithm combines an insert step with a shake step to escape from local optima. The insert step adds consecutively new points into the tours. The insertion feasibility is quickly done by using two recorded values about each already included point, namely *Wait* and *MaxShift*. The *Wait* value represents the waiting time at a point, in case the arrival occurs before the opening of time window, whereas the *MaxShift* value indicates the maximum possible shift of a visit to a point without making other visits into the tour infeasible. The shake step removes one or more visits from each tour. The place of removal and number of consecutive visits to be removed are determined by two controlling parameters, namely R_d respectively S_d . Due to different tour lengths, the value of S_d is different for different tours. This makes the shake step more effective in escaping from local optima. The computation time of this algorithm is, with a factor several hundred, faster than the other presented approaches for TOPTW. In terms of solution quality, the average gap to prior approaches is only 2%.

Tricoire et al. [128] name an extended variant of TOPTW as Multi Period Orienteering Problem with Multiple Time Windows (MuPOPMTW). In this modeling, every point can have

multiple time windows, where each time window can be different in different days. The proposed solution is based on the framework of Variable Neighborhood Search (VNS) algorithm and it is embedded with an exact algorithm to cope with feasibility evaluations. A deterministic heuristic based on stochastic local search of *best insertion* [116] is developed for construction of initial solution which is than used by VNS algorithm. The main steps of VNS algorithm are *shaking*, *iterative improvement* and *acceptance decision*. The *shaking* step is based on three operators, namely *cross-exchange* [125], *optional exchange 1* (exchanges a number of optional customers with number of other unplanned customers) and *optional exchange 2* (removes a number of optional customers from the tour). The *shaking* step makes the neighborhood exploration process very distinct in consecutive iterations, and as such, the algorithm has a higher probability to escape from local optimum. In the *iterative improvement* step, *3-Opt* operator re-optimizes the tour spent time after utilization of *cross-exchange* and *optional exchange 1* operators, or fills up the vacant places in the tour by using a *sequential best insertion* procedure after utilization of *optional exchange 2* operator. In the *acceptance decision* step, a newly found solution can be accepted as current solution if it is better than the running solution, or if it meets a threshold accepting criteria that is correlated to particular operators used during the *shaking* step. The experimental results show that VNS approach obtains high quality solutions in about one minute of computation time.

Later, Montemanni et al. [87] enhance the performance of their ACS approach by adding two new operations. The first operation is about the constructive phase, which deals with both diversification (exploring new regions of the search space) and intensification (searching very intensely a given region of the search space). The second operation regards the integration between the constructive phase itself, and the local search procedure. The enhanced ACS outperforms the previous ACS approach and obtains new best results for number of instances in the literature for TOPTW problem.

Garcia et al. [42] solve a variant of OPTW called Time Dependent Orienteering Problem with Time Windows (TDOPTW), which models the variability of traveling time between points. The authors use TDOPTW to model the traveling time of a tourist when she/he moves between different points of interest either by walking or by public transportation. Authors present a hybrid meta-heuristic based on two different algorithms, Dijkstra's Shortest Path algorithm and Iterated Local Search (ILS). The Dijkstra's algorithm is used to calculated the average travel times between all points available. Due to the constraint for a real time response, the traveling times between points are done in an off-line-mode and saved for letter use by the optimization logic. When the traveling times have been calculated, the TDOPTW converts to regular OPTW. Next, the ILS algorithm presented by Vansteenwegen et al. [134] is used to solve the derived problem. The experimental performed with the data of city of San Sebastian in Spain, prove that the approach is able to generate valid routes in real time.

Labadie et al. [74] introduce a hybridized evolutionary local search approach for TOPTW. The approach combines the greedy randomized adaptive search procedure (GRASP) with the evolutionary local search (ELS). ELS is used to generate multiple distinct child solutions using a mutation mechanism. Each child solution is further improved by a local search procedure. In each iteration of ELS, GRASP is used for construction of starting solution. The initial solution is constructed by using five similar methods, where three of them are insertion operators, whereas

the other two are variants of sweep algorithm (at first points are clustered into subgroups than for each subgroup a tour is built). GRASP, as a local search procedure, alternates between two different neighborhood exploration procedures. The first procedure aims at reducing the tour length through *k-opt* operator (in three different variants) and *swap* operator, whereas the second procedure enables introduction of unvisited points into the current solution by applying a *replace* operator to replace points in the tour with unvisited points. ELS has the task to diversifying the search process through a perturbation phase. This is realized by randomly removing a sequence of points in each tour and then creating new sequence of points (from unvisited points) to fill the created gaps in the respective tours. The experimental results show, this approach succeeds in improving several best known results from the benchmark instances in the literature

Labadie et al. [73] propose a LP-based granular variable neighborhood search (FVNS) for TOPTW. The method is focused on improving the efficiency, without loosing the effectiveness, of search process by proposing a granular (instead of complete) exploration of search space. The construction of initial solution is done by using the same method as described earlier in the hybridized ELS [74]. The VNS algorithm, first uses a routine for removing a sequence of k nodes and replacing it with a new sequence of no predefined length, and then applies a local search procedure. As in the hybridized ELS [74], the local search procedure comes with two variants of neighborhood exploration. The first tries to shorten the tour length, whereas the second one tries to increase the total collected score. Since, the second variant of neighborhood is quite large, a granular variant is introduced with the aim of reducing the size of analyzed neighborhood space. The idea stands at excluding non promising arcs during the construction of the sequence of points in the local search procedure. In general, this procedure can be described in two steps. The first step is using a graph algorithm to solve to optimality an assignment problem, and the second step is using a dual information function to identify more promising arcs. The method performs quite good on standard benchmark instances by allowing to improve the best known values for 25 test instances.

Lin and Yu [79] solve TOPTW by using a standard Simulated Annealing (SA) approach. The structure of neighborhood is random and consist of three types of operators, namely *swap, insert* and *inverse*. The *swap* operator randomly selects two points into the tours and exchanges their positions. The *insert* operator randomly selects a point X from the tour and then tries to insert it before some other randomly selected point Y. The *inverse* operator randomly selects two points, and then reverses the sequence of points that are between the selected points (including the selected points). Each operator has a probability of 1/3 for adaption to generate the neighborhood candidates at a particular iteration. The algorithm can be fine tuned with the help of five parameters such as, number of iterations at a particular temperature, minimal temperature, maximum allowable computation time, maximum allowable temperature reduction without improvement and speed of cooling schedule. Depending on parameter settings, the algorithm can work in a fast mode (FSA) or in slow mode (SSA). The experimental results show that the FSA mode is competitive to the ILS approach of Vansteenwegen et al. [134], both in terms of computation time and solution quality. Moreover, the SSA mode outperforms the prior approaches by finding 33 new best solutions for the existing instances in the literature.

Mota et al. [90] apply Genetic Algorithms (GA) to solve a variant of TOPTW named TOPdTW, which, in addition to time windows for points, also models time windows for tours.

This problem is motivated from the real problem of operating rooms in the hospital. The algorithm is characterized with three components, namely *chromosome* (representing the set of tours), *evolutionary process* (executing mutation and crossover operators) and *evaluation* (fitness and feasibility check). The crossover operator is applied between two tours that are selected based on the principle of roulette-while. Fragments of tours are selected randomly and are used to create two new tours (chromosomes). The mutation operator randomly removes a customer from randomly selected tour and then attempts to add one or more unvisited customers. The presented preliminary results indicate that the algorithm has the potential of being used to solve problems of practical relevance.

Gavalas et al. [46] tackle TOPTW by using two related approaches from cluster-based heuristics called Cluster Search Cluster Ratio (CSCRatio) and Cluster Search Cluster Routes (CSCRoutes). The presented approaches aim at encouraging visits to topology areas that consist of a high density of "good" points. Both heuristics employ a clustering procedure to organize points into clusters based topological distance. This increases the probability that more visits would take place inside individual clusters, and as result, two positive effects could arise, the decrease of duration of tours and minimization of number of transfers between different clusters. Both CSCRatio and CSCRoutes employ the global *k-means* algorithm [78] to build the clusters of points. In the initialization phase, in order to start from diverse positions in the search space, the k tours of the solution are constructed by using points from different clusters. Further, in the local search phase, both approaches use *insert* and *shake* operators that are originated by Vansteenwegen et al. [134] for exploring the neighborhood of current solution. CSCRatio heuristic is designed to favor tours with more points inside individual clusters, whereas CSCRoutes heuristic is designed to construct tours that visit each cluster at most once. This approach improves in terms of quality in comparison to ILS [134], while keeping the computation time in comparable level.

Recently, Hu and Lim [64] present an iterative three-component heuristic for TOPTW. The first two components consist of a local search procedure and a simulated annealing procedure that discover a set of routes and store them in the so called route "pool", whereas the third component recombines the routes into the "pool" to identify better solutions. The first two components initially use a neighborhood operator called *eliminator* that randomly removes some customers (points) from some tours, and then inserts unvisited customers (points). Next, with the aim of improving the local search solutions, a *post-processing* procedure iteratively applies seven operators. These seven different operators could be grouped into three types, *relocate* (insert a customer into tour), *exchange* (swap two customers) and *2-opt* (separates two tours into four parts). The *route combination* component receives the routes from the "pool" and recombines them by deriving a set packing formulation. The computational results indicate that this approach outperforms the existing approaches in the literature in average performance by 0.41%. Further, 35 new best solutions for the existing instances in the literature are presented.

2.6.5 Multi Constraint Team Orienteering Problem with Time Windows

In Multi Constraint TOPTW (MCTOPTW), each point, in addition to the inherited attributes (score S_i, visit time T_i, opening time O_i, closing time C_i), is also associated with a set of e_{iz} attributes, where $z \in \{1, ..., Z\}$. Attribute e_{iz} might represent different features of point i such

as, entry fee, capacity, type, category, etc. The notations of MCTOPTW [43] are the same as those of TOPTW presented in previous subsection, except for the addition of an extra maximum attribute constraint that constrain the selection of point *i* based on its e_{iz} attribute value and on E_z value as given in Equation 3.6.

$$\sum_{i=1}^{N} e_{iz} y_{im} \leq E_z;\ \forall z = 1, ..., Z;\ \forall m = 1, ..., M \tag{2.33}$$

Garcia et al. [43] are the first to introduce and solve the MCTOPTW. They extend the original approach of ILS [134] for TOPTW to use and solve MCTOPTW problem. After the phase of initial solution construction (through local search), the ILS metaheuristic iteratively executes the sequence of three main procedures, namely *perturbation, local search,* and *acceptance criterion*. The implementation of *local search* procedure is based on an *insert* operator that tries to include new points into the tour by using the concepts of *Wait* and *MaxShift* proposed by Vansteenwegen et al. [134]. The *perturbation* procedure is implemented with the help of a *shake* operator that excludes a number of consecutive visits from the tour. The heuristic for evaluation of candidate solutions differs form ILS, since it has to also consider the added attribute constraints. Moreover, different versions of fitness function are proposed and then experimented. The experimental results show that this approach is competitive when applied to solve the instances of the related problems in the literature.

Another approach for the extended variant of MCTOPTW with multiple time windows (MCTOPMTW) is presented by Souffriau et al. [119]. The proposed approach for solving MCTOPMTW is a hybridization between ILS and GRASP algorithms. This approach is also based on the ILS approach of Vansteenwegen et al. [134] for TOPTW. In this regard, the *insert* and *shake* operators are used as part of the local search procedure of ILS. Since the problem at hand is highly constraint, an extra component to deal with these constraints is added to ILS. This component is based on GRASP and it performs a number of consecutive iterations, each consisting of a constructive procedure followed by local search mechanism. This hybrid approach has an average computation time of only 1.5 seconds, whereas the quality of solutions has a gap of only 5.2% in comparison to the best known solutions. Further, the ILS-GRASP approach outperforms the approach of Garcia et al. [43].

2.6.6 Generalized Orienteering Problem

The Generalized Orienteering Problem (GOP) extends OP by introducing a range of S_i attribute scores (instead of one) for each point. Depending on particular application, an adequate function needs to be selected for calculation of the final score of the tour. For instance, in [138] a weighted function that specifies W_i for each of *m* attribute scores has been adopted such that that if fulfills the Constraint 2.34. If *X* out of *N* points of an instance are included into the tour, then the score is calculated by using the function expressed by Equation 2.35 for some non-negative exponent *k*. If *k* goes toward the infinite value, then the value of the given function becomes the sum of the maximum scores attained by *X* points for each of their attribute scores. In the special case, when *k=1* and *m=1* then GOP converts to OP.

$$\sum_{i=1}^{M} W_i = 1, \tag{2.34}$$

$$S_N = \sum_{i=1}^{m} W_i [\sum_{j=1}^{X} S_i(j)^k]^{1/k} \tag{2.35}$$

A number of heuristic approaches have been proposed for solving GOP. The first approach is presented by Ramesh and Brown [99] based on a four phase heuristic that consist of point insertion, cost improvement, point deletion, and maximal insertion. All four phases are integrated into an algorithmic framework, which could be guided by set of five parameters. The experimental results show that the algorithm is able to find near to optimal solutions with minimal computation effort.

Geem et al. [48] tackle GOP by using a Harmony Search (HS) algorithm. There are five main steps that comprise the algorithm: (1) *initialize algorithm parameters*, (2) *initialize HS memory*, (3) *improvise new harmony*, (4) *update HS memory*, and (5) *check stopping criterion*. The algorithm is tested against test instances created by using cities from eastern part of China. The experimental results show that HS obtains comparable results with the earlier proposed approaches.

Silberholz and Golden [115] propose an algorithm based on large neighborhood search [113] and on ruin and recreate principle [111]. The algorithm can be controlled with the help of two parameters and it maintains a single solution during the search process. There are two distinctive steps applied: *initialization* and *iterative modification*. In the *initialization* step, first the points are appended iteratively at the end of the tour until no more time space is left. Then, a *2-Opt* operator is applied to reduce the tour length. Further, a local search *path tightening* operator adds new points at the end of the tour whenever theres is new space created as result of application of *2-Opt* operator. In the *iterative modification* step, first a number of points are removed, then a modified version of *path tightening* operator is applied by given less priority to points that were just removed, afterwards the *2-Opt* operator is applied, and finally the unmodified *path tightening* operator is enforced. This procedure iterates until the returned solution is worse than the previous solution. The proposed approach outperforms the previous approaches for the GOP.

2.6.7 Orienteering Problem with Hotel Selection

Divsalar et al. [30] extend OP to Orienteering Problem with Hotel Selection (OPHS). In this problem, a set of points with score and set of hotels are given. The objective is to find a fixed number of connected trips that visit a subset of points that maximize the total collected score. Each trip has a limited duration and it starts and ends in one of the available hotels. Based on the original formulation of OP, Divsalar et al. [30] formulate OPHS as a mixed integer programming problem by using the following variables and equations: $x_{ij,d}$ is set to *1* when in trip *d* a visit to point *i* is followed with a visit to point *j*, otherwise $x_{ij,d}$ is set to *0*; u_i shows the index of visit to point *i* in tour.

$$Max \sum_{d=1}^{D} \sum_{i=0}^{H+N} \sum_{j=0}^{N+H} S_i y_{ijd}, \tag{2.36}$$

$$\sum_{l=1}^{H+N} x_{0,l,1} = 1, \tag{2.37}$$

$$\sum_{k=0}^{H+N} x_{k,1,D} = 1, \tag{2.38}$$

$$\sum_{h=0}^{H} \sum_{l=0}^{H+N} x_{hld} = 1;\ \forall d = 1, ..., D, \tag{2.39}$$

$$\sum_{h=0}^{H} \sum_{k=0}^{H+N} x_{khd} = 1;\ \forall d = 1, ..., D, \tag{2.40}$$

$$\sum_{k=0}^{H+N} x_{khd} = \sum_{l=0}^{H+N} x_{hl(d+1)};\ \forall d = 1, ..., D-1;\ \forall h = 0, ..., H, \tag{2.41}$$

$$\sum_{i=0}^{H+N} x_{ikd} = \sum_{j=0}^{H+N} x_{kjd};\ \forall k = H+1, ..., H+N;\ \forall d = 1, ..., D, \tag{2.42}$$

$$\sum_{d=1}^{D} \sum_{j=0}^{H+N} x_{ijd} =\leq 1;\ \forall i = H+1, ..., H+N, \tag{2.43}$$

$$\sum_{i=0}^{H+N} \sum_{j=0}^{H+N} t_{ij} x_{ijd} =\leq T_d;\ \forall d = 1, ..., D, \tag{2.44}$$

$$u_i - u_j + 1 =\leq (N-1)(1 - \sum_{d=1}^{D} x_{ijd});\ \forall i, j = H+1, ..., H+N, \tag{2.45}$$

$$u_i \in \{1, ..., N\};\ \forall i = H+1, ..., H+N, \tag{2.46}$$

$$x_{ijd} \in \{0, 1\};\ \forall i, j = H+1, ..., H+N \mid i \neq j;\ \forall d = 1, ..., D, \tag{2.47}$$

The objective function of OPHS (Constraint 2.36) is maximization of the total collected score. Constraint 2.37 makes sure that the tour commences at the starting hotel, whereas Constraint 2.38 guarantees the tour ends at the ending hotel. Constraints 2.39 and 2.40 ensure that each trip starts and ends in one of the available hotels. Constraint 2.41 makes sure that when a trip ends at a given hotel, the next one starts from the same hotel. Constraint 2.42 enforces the connectivity by constraining current visit to point *i* in trip *d* with a next visit to point *j* in the same trip. Constraint 2.43 makes sure that a point can be maximally visited once. Constraint

2.44 sets tour's maximum allowed time budget to T_d. Constraint 2.45 eliminates the subtours in the tour.

In addition to the formulation of OPHS problem, Divsalar et al. [30] propose a Skewed VNS (SVNS) approach for obtaining high quality solutions. In general, in SVNS slightly worse current solutions might be accepted, provided the that solution is far from the incumbent. Nevertheless, in [30] the distance form current solutions and the incumbent is not taken into consideration. The algorithm consists of three phases: *initialization, improvement* and *re-centering*. In the *initialization* phase, first a number of feasible combinations of hotels for the tour is calculated and stored in the list called "Number of Used Feasible Combination of hotels (NUFC)" for later use, and then a sub-OP problem is solved for each trip of the tour by using a local search procedure that include *insert, replacement, 2-Opt* and *Move-best* operators. The *improvement* phase consists of two sub phases, namely *shaking* and local search. The *shaking* phase performs the "shaking" of points (by deleting the half of points from a trip) and the "shaking" of hotels (by replacing current combination of hotels with another combination from NUFC list). The *local search* phase applies a variety of operators, such as: *insert* (inserts a non-included point), *move-best* (moves a points to a best possible position), *2-Opt* (reduces the travel time in each trip), *swap-trips* (exchanges two points between two different trips) and *extract-n insert* (removes *n* points from each trip and tries to insert new points in the vacant place). Finally, the *re-centering* phase decides which solution would be accepted as the next solution, the current solution that is obtained during the improvement phase or the one that was used at the start of current iteration.

Further, Divsalar et al. [30] design a testbed of 224 instances with known optimal solutions based on instances of Tsiligirides [129] and Chao et al. [19]. The proposed solution finds the optimal solutions for 102 instances, whereas the average gap from optimal solutions over all instances is 1.44%. The average computation time is 1.91 seconds.

Later, in form of a preliminary work, Divsalar et al. [29] rename the OPHS with the name Orienteering Problem with Intermediate Facilities (OPIF), and propose e new metaheuristic approach based on memetic algorithms (MA). The structure of proposed MA to tackle OPIF is a combination of genetic algorithms (GA) and variable neighborhood search (VNS) method . The GA deals mainly with the optimization of intermediate facilities. The algorithms consists of two major steps: *initialization* and *improvement*. In the *initialization* step, the OP is solved between the intermediate facilities and a number of feasible combinations of intermediate facilities is created by using a local search procedure. In the *improvement* step, GA is applied by using two crossover operators and one mutation operator with the aim of diversification of search space by creating different combination of intermediate facilities. The preliminary results, show that the MA approach performs even better than SVNS approach, both in terms of solution quality and computation time.

2.7 Summary

The tourism information provided by a TIS has a crucial role in enhancing the tourist experience [35] throughout her/his tourism life cycle [141]. In addition, to provision of common services like hotel reservation, transport ticket purchasing, navigation, etc., we underlined that the role of a TIS system might also be the recommendation of tourism information and prod-

ucts, such as trips, points of interest and activities. More specialized tourist products could be the trip itineraries that include a list of POIs to be visited during a limited time period, which are planned by taking into account tourist preferences (interest about POI categories and types) and constraints (budget, maximum number of POIs of certain category). We outlined a number of prototypical and commercial systems that enable tourist itinerary planning for a single tourist. Nevertheless, based on our best knowledge, there exist no any system the enable trip planning for a group of tourists, where both the individual preferences of tourists about POIs and their mutual social relationship are considered.

The solo trip planning problem is known with name Tourist Trip Design Problem (TTDP) [135]. We noted that OP can be used to model the simplest variant of TTDP, where single day itinerary could be planned. Based on Golden et al. [57] OP belongs to the class of NP hard problems and as result systematic approaches are not suitable to solve the problem when considering realistic situations with many number of points. Moreover, the various extended variants of OP, such OPTW, TOP, TOPTW, TDOP, TDOPTW, MCTOPTW, TDTOPTW, GUP and OPHS are at least as complex as OP. In this regard, most of the authors tackle the afore mentioned problems by using various optimization techniques from the field of meta-heuristics. Some of the most wildly used metaheuristics include TS, ILS, VNS, SVNS, GA, ACO, HS, GRASP, SA, GLS and a number of hybridized approaches. The underlined metaheuristics explore the neighborhood mostly based on operators like *Insert* (a new point), *Remove* (a point from tour), *Swap* (two points inside tour), *Replace* (an existing points with a new point), *k-Opt* (remove k edges and add k new edges) and *Move* (a point to a new position). Note that it is common that different approaches implement these operators in slightly different variants.

Tabu search (TS), as one of the most frequently used metaheuristic in different domains of application, is known with its abilities for escaping from local optima by using a "tabu" memory that saves information about frequency and recency of used moves throughout the search process. The saved information is usually used to either diversify the search process toward unexplored areas of search space, or intensify the search process to dig more deeply in some particular are of search space. Our motivation for using TS as primary approach for solving single and group tourist trip planning problem is based on prior successfully application of TS for solving OP [52] and its TOP extension [126] [4].

CHAPTER 3

Planning Solo Trip Itinerary

This chapter presents an algorithm that can be used for planning the trip itinerary by considering single tourist preferences. Based on foundations about current modelings and approaches from the literature, for the problem of solo trip itinerary planning, we consider as it follows: multiple day tours, single time windows for points, no waiting time in between consecutive visits, maximum budget limit of the tourist, and multiple constraints that can be expressed by the tourist about categories of points. In this regard, a solo trip itinerary planning problem that is expressed with above mentioned constraints can be qualified as Multiple Constraint Team Orienteering Problem with Time Windows (MCTOPTW) [43], with the extension of the constraint for not allowing any waiting time in between consecutive visits.

If we consider Case 1 of the previously presented scenario, the envisioned algorithm in this chapter would be able to plan the trip itinerary separately for each tourist or jointly for all of them. In the separate approach, the algorithm has to make three separate runs for each of the three tourists in the example and the itinerary of tourists would be optimized by considering individual preferences of tourists abut POIs, without taking into account the mutual social relationship factor between the tourists. In the joint approach, the algorithm needs to runs only once as all tourists would be planned to travel together with same itinerary, whereas also their social relations would be taken into account.

In the following we initially present a formal modeling (Section 3.1) of the envisioned problem, and then present the our tabu search metaheuristic implementation (Section 3.2) for solving the problem. Afterwards, we show and discuss the experimental results (Section 3.3) that are obtained by executing the algorithm on standard test sets from the literature. Finally, we draw our conclusions (Section 5.4) about the performance of our algorithm in comparison to the state of the art approaches in terms of solution quality.

3.1 Mathematical Modelling

A number of N points are given in a certain geographic location. Each point is characterised with a score S_i, an entry fee E_i, a visit duration T_i, an opening O_i and closing C_i time and a range

of Z logical values e_{iz} that indicate the different categories that the point belongs to. Each point can be visited at most once. The starting point *1* and ending point N are fixed for each tour. The traveling times t_{ij} between points i and j are known for all pairs of points. The start of a visit to point i in tour m is denoted as s_{im}. The trip consists of M tours, each limited to a maximal duration of T_{max}. Further, the tour has a range of Z maximum values E_z that limit the number of selected points of category z. The aim is to visit a subset of points that would maximise the overall trip score by complying with the underlined hard constraints. Higher trip scores imply better fitness to tourist preferences.

In the following, we rewrite the mathematical formulation of the MCTOPTW problem made by Garcia et al. [43]. The definition of two Boolean variables, namely x_{ijm} and y_{im}, is done with following functions:

$$x_{ijm} = \begin{cases} 1, & \text{if a visit to point } i \text{ is followed with a visit to point } j \text{ in tour } m \\ 0, & \text{otherwise} \end{cases}$$

$$y_{im} = \begin{cases} 1, & \text{if point } i \text{ is visited in tour } m \\ 0, & \text{otherwise} \end{cases}$$

The defined variables x_{ijm} and y_{im} are used in the following formulation equations of MCTOPTW problem:

$$Max \sum_{m=1}^{M} \sum_{i=2}^{N-1} S_i y_{im}, \tag{3.1}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{N} x_{1jm} = \sum_{m=1}^{M} \sum_{i=1}^{N-1} x_{iNm} = M, \tag{3.2}$$

$$\sum_{i=1}^{N-1} x_{ikm} = \sum_{j=2}^{N} x_{kjm} = y_{km};\ \forall k = 2, ..., N-1;\ \forall m = 1, ..., M, \tag{3.3}$$

$$(x_{ijm} = 1) \implies (s_{jm} = s_{im} + T_i + t_{ij});\ \forall i, j = 1, ..., N;\ \forall m = 1, ..., M, \tag{3.4}$$

$$\sum_{m=1}^{M} y_{im} \leq 1;\ \forall i = 2, ..., N-1, \tag{3.5}$$

$$\sum_{i=1}^{N} e_{iz} y_{im} \leq E_z;\ \forall z = 1, ..., Z;\ \forall m = 1, ..., M \tag{3.6}$$

$$O_i \leq s_{im};\ \forall i = 1, ..., N;\ \forall m = 1, ..., M, \tag{3.7}$$

$$s_{im} \leq C_i;\ \forall i = 1, ..., N;\ \forall m = 1, ..., M, \tag{3.8}$$

$$\sum_{i=1}^{N-1}\left(T_i y_{im} + \sum_{j=2}^{N} t_{ij} x_{ijm}\right) \leq T_{max};\ \forall m = 1, ..., M, \tag{3.9}$$

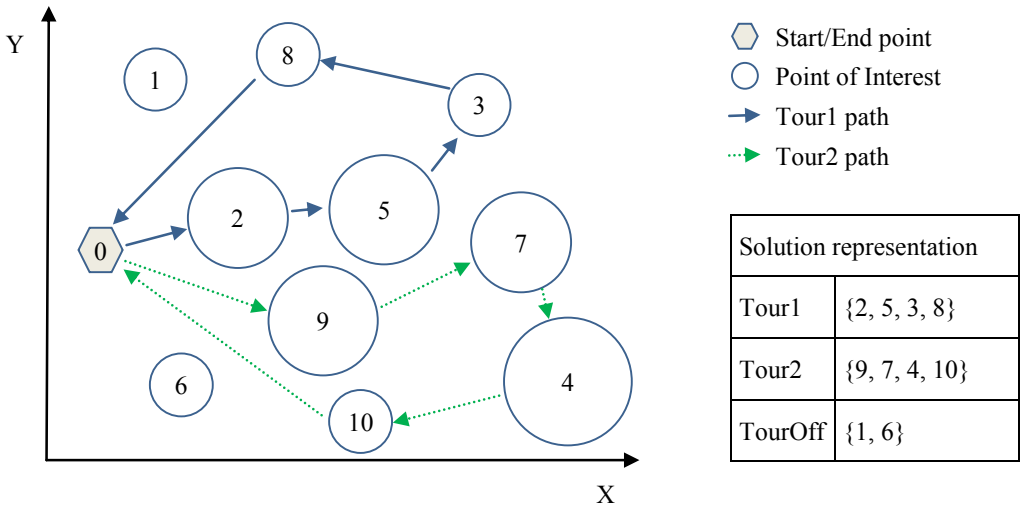
Equation 3.1 expresses the objective function in solving the MCTOPTW problem. Start of each tour at point *1* and its end at point *N* is ensured by Constraint 3.2. Constraint 3.3 makes sure that each visit to a point is followed with another visit to a next point. Constraint 3.4 allows no waiting times in between consecutive visits, which, as pointed out earlier, differentiates from the original problem defined in [43] and [119], where a waiting time before starting a visit at a certain point could occur. Constraint 3.5 allows maximum one visit to any visited point, while 3.6 limits the number of selected points of certain category to an upper bound. Constraint 3.7 and 3.8 bound the start of the visit to a point to its time window, while 3.9 constraints the tour duration to its specified limit.

3.2 Solution Approach

In tourist trip itinerary planning applications, a short computation time of few seconds is expected. Therefore, considering that MCTOPTW is a highly constrained and NP hard problem, exact approaches are not expected to obtain solutions in such short computation time. In such situations, the optimization techniques from the filed of metaheuristics are frequently used for acquiring good solutions in a short period of time. Successful applications of tabu search by Gendreau et al. [52] to solve OP and Tang and Miller-Hooks [126] to solve TOP, indicate that tabu search could be a good alternative for tackling MCTOPTW problem too. Based on that, we opt to use tabu search as a framework for our algorithm to solve MCTOPTW. The algorithm is characterized with two main components, namely *neighborhood exploration* (for an efficient search of a particular area of search space) and *search diversification* (to ensure a broader exploration of search space). In the following subsections, we initially describe the solution representation, discuss the component of *neighborhood exploration*, show insights about the process of feasibility evaluation of the candidate solutions, and then present the *search diversification* process. At the end, we elaborate the general structure of our tabu search implementation.

3.2.1 Solution Representation

In our encoding, we represent the solution by using a list for each of the *m* tours, where each list contains the sequence of points scheduled for visits. In addition, a separate list is used to keep record of points that are not inside the solution. Considering Example 1 of the scenario, which has *N=10* points and *m=2* tours, a candidate solution for tourist *p1* might be represented by *Tour1={2, 5, 3, 8}, Tour2={9, 7, 4, 10} and TourOff {1, 6}* (see Figure 3.1). The score of a point is illustrated with the size of the respective circle. The larger the circle the higher the score.



Solution representation	
Tour1	{2, 5, 3, 8}
Tour2	{9, 7, 4, 10}
TourOff	{1, 6}

Figure 3.1: Sample representation of a solution with two tours

3.2.2 Neighborhood Exploration

The neighborhood structure of the approach of Souffriau et al. in [119] consists of two basic operators, namely *Insert* (a non-included point in tour) and *Remove* (an existing point from the tour). In our approach, we also apply *Insert* operator, whereas we use also the *Replace* and *Swap* operators. As part of the mechanism for diversification of search process, we apply a *Delete* operator which is equivalent to the *Remove* operator that was applied by Souffriau et al. in [119]. In the following, we give a short description for each of the applied operators.

Insert operator tries to insert one of the non included points into the current solution by considering all possible positions into all tours. In case of a solution with *M* tours, the insert operator would pick each non included point and try to insert it before each existing point into each of *M* tours and at their end. The time consumption for insertion of a point *k* before point *i* at a particular tour is $\triangle t = t_{(i-1)k} + T_k + t_{ki} - t_{(i-1)i}$ (see Figure 3.2). If $\triangle t > 0$, all points after point *i* are shifted towards the end of the tour, otherwise if $\triangle t < 0$, the points are shifted towards the beginning of the tour.

Replace operator replaces an existing point into one of the tours with a non-included point. All possible combinations are tried out by considering each non-included point to replace each point currently in the solution. The time consumption for replacing point *i* with point *k* at a particular tour is $\triangle t = t_{(i-1)k} + T_k + t_{k(i+1)} - t_{(i-1)i} - T_i - T_{i(i+1)}$. If $\triangle t > 0$, all points after point *i* are shifted forward, otherwise if $\triangle t < 0$, the points are shifted backward.

Swap operator is applied between any two points inside current solution. The points might belong to the same tour or to different tours ($x = y$ respectively $x \neq y$ in Figure 3.4). The time consumption for tour *x* is $\triangle t_x = t_{(i-1)k} + T_k + t_{k(i+1)} - t_{(i-1)i} - T_i - T_{i(i+1)}$, whereas for tour *y* it is $\triangle t_y = t_{(k-1)i} + T_i + t_{i(k+1)} - t_{(k-1)k} - T_k - T_{k(k+1)}$. Depending on values of $\triangle t_x$ and $\triangle t_y$, whether they are positive or negative, the shift of the following points in the respective tours could be forward respectively backward.

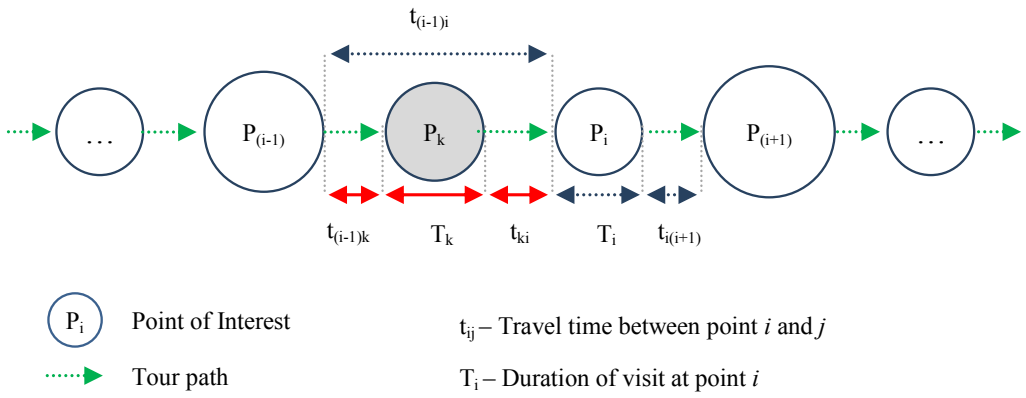


Figure 3.2: Insertion of a new point

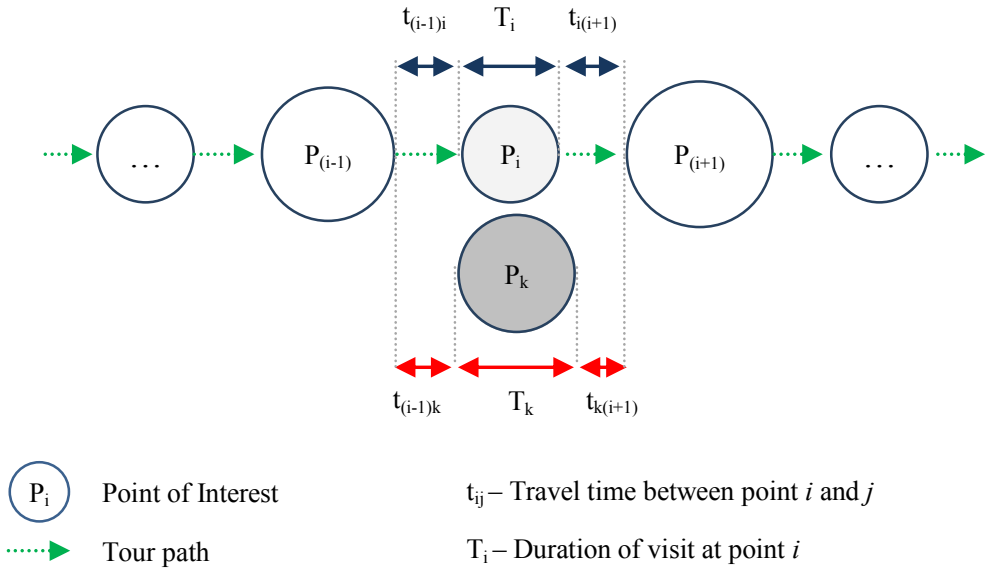


Figure 3.3: Replacement of an existing point with a new point

Each of the operators always returns two feasible neighbors, one of them being 'best non-tabu' and the other one 'best tabu'. The two best neighbors are created by choosing two combinations ('tabu' and 'non tabu') that have the highest scores. In case there are more combinations that have the same highest score, then one of them is selected randomly.

3.2.3 Feasibility Evaluation

As formally presented earlier, the original MCTOPTW problem is associated with a number of hard constraints, such as maximum tour duration, the limit for the maximum number of points

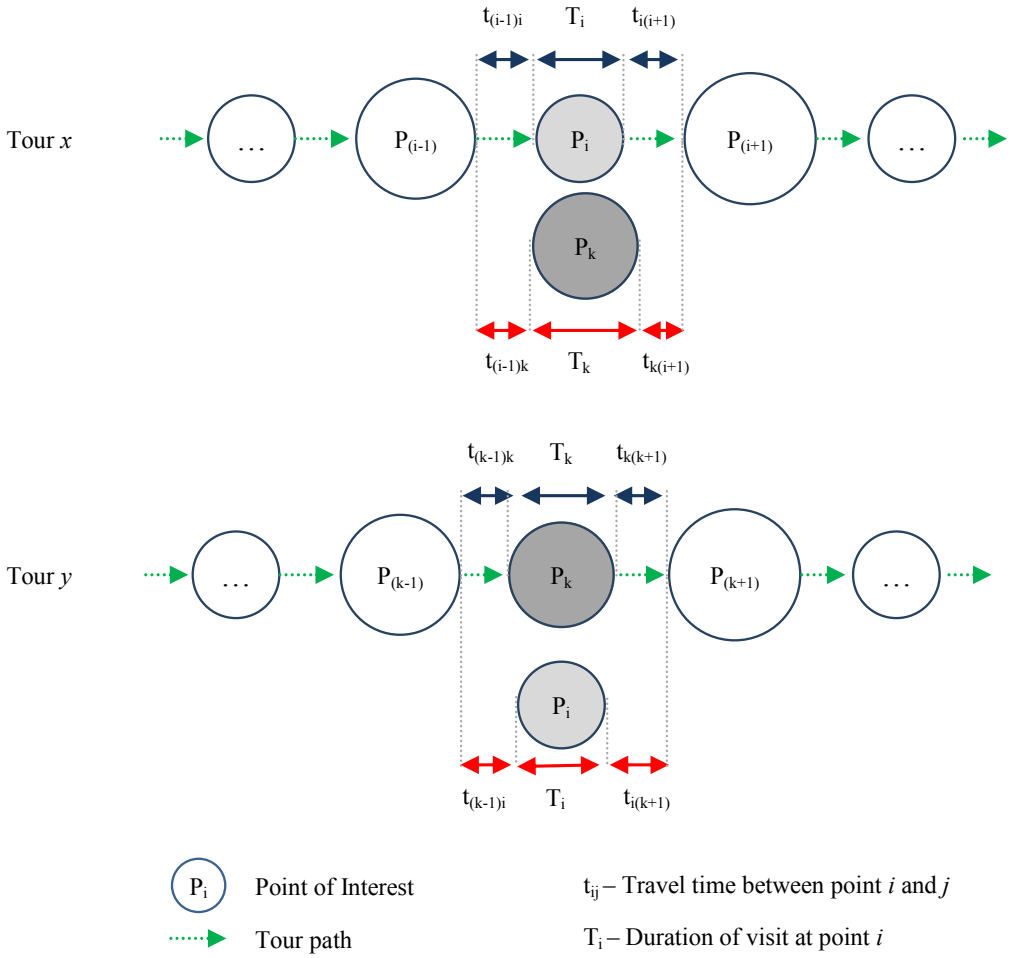


Figure 3.4: Swapping between two existing points inside current solution

of particular categories, and the time windows. The first two constraints are evaluated by using an efficient deviation/incremental function when a particular move is applied. The evaluation of feasibility of Time Window (TW) constraint is the most time consuming action. Vansteenwegen et al. [134] record a *Wait* and *MaxShift* value to speed up the time window feasibility check process. The *Wait* value indicates how much time a person has to wait before starting a visit, while the *MaxShift* value shows how much time a certain visit could be delayed. In our approach, we do not utilize a *Waiting* time value, since we do not allow any waiting time prior to starting a visit. Instead, a pair of two values, namely *Max Forward Shift* (MFS_i) and *Max Backward Shift* (MBS_i), is saved for each point inside the trip itinerary. These variables show how much time a point at position i in tour m could be shifted forward respectively backward.

Suppose there is a point x that either replaces/swaps point i or it is inserted before point i of tour m. Equation 3.10 is a function that defines the consumption time $\triangle t$ that is applicable for

Replace, Swap and *Insert* operators. Constraint 3.11 bounds the consumption time $\triangle t$ in the boundary values of the pair of -MBS_{i+1} and MFS_{i+1}.

$$\triangle t = \begin{cases} t_{(i-1)x} + T_x + t_{x(i+1)} - t_{(i-1)i} - T_i - t_{i(i+1)}, & \textit{Replace, Swap} \\ t_{(i-1)x} + T_x + t_{x(i+1)} - t_{(i-1)i}, & \textit{Insert} \end{cases} \tag{3.10}$$

$$-MBS_{i+1} \leq \triangle t \leq MFS_{i+1} \tag{3.11}$$

Whenever a new point is either inserted, replaced or swapped in tour *m*, the corresponding MBS_i and MFS_i values of all points in tour *m* need to be updated. This technique is useful when evaluating the feasibility of time windows of candidate solutions, since it eliminates the need to check the time window feasibility of points after point *i+1* thus reducing the computational effort to a great extent.

3.2.4 Search Diversification

The mechanism for diversification of search process is done by using four different additional operators, namely *Delete, Perturbate, Restart* and *Penalise*, which are described in the following.

The *Delete* operator removes two points from the tour that is the most constrained by the TW constraint. The most constrained position in the tour is found by considering the MBS_i and MFS_i values. In case of a tie between most constrained tours, a random selection of one of them is made.

Perturbation is realized by applying the *Delete* operator in the actual best found solution and then continuing the search from that perturbed solution. At times, as specified by one of the algorithm parameters, the search process is re-initiated from a random solution.

The algorithm consists of two usual tabu lists, namely *recency* and *frequency* memories. The *recency* memory is used to record the latest iteration index when a certain move (*Insert, Replace* or *Swap*) between two points is applied. The Tabu List Size - TLS (described in next subsection) parameter determines how many iterations the applied move would remain tabu. On the other hand, the *frequency* memory keeps track of the number of iterations a certain move has been applied. The *penalize* operator uses the *frequency* memory to penalize all moves that have been used more than 10 times up to the iteration when the operator is applied. The penalization is made by making the tabu status of these frequent moves twice as longer as the original tabu list size.

3.2.5 Tabu Search Implementation

The algorithm is provided with six basic parameters that enable the fine tuning of its performance. The parameters are presented in Table 3.1, where the first column presents parameter abbreviation, the second one its whole name, while the third one presents a short description for the parameter.

At the start, as presented in Algorithm 3.1, the procedure that creates a random initial solution is executed and tabu memories are initialized to an empty state. The evaluation of initial

Abbreviation	Name	Description
TLS	Tabu List Size	Specifies the number of iterations a certain move cannot be used.
MI	Maximum Iterations	Defines the number of iterations the algorithm shall run.
DOF	Delete Operator Frequency	Determines how often the delete operator needs to be applied.
PF	Perturbation Frequency	Sets the frequency of perturbation from actual best found solution.
RIF	Random Initialization Frequency	Indicates the frequency of re-initialization from random solution.
PEF	Penalization Frequency	Defines the frequency of penalization of frequently used moves.

Table 3.1: Algorithm parameters

solution and, latter, the generated candidate solutions, is done by using the objective function expressed by Equation 3.1. Next, the algorithm enters into a loop that runs *MI* iterations. The neighborhood exploration alternates between the above described operators in such way that in every second iteration, the algorithm uses the *Replace* operator, and then in between, in turn, it uses *Insert* and *Swap* operators. In each iteration, a best tabu and non-tabu solution is searched by trying all possible combinations available within the running operator. The best non tabu solution is considered first for adaption as current solution. If the non-tabu solution is not better than the current solution, then the best tabu solution is tested whether it fulfills the aspiration criteria. In this implementation, a tabu solution is considered to be fulfilling the aspiration criteria only if it is better than the best found solution so far. Normally, the adaption of a better solution is done whenever a new, better one, is found, otherwise the number of iterations without improvement is recorded. In occasional iterations, when the number of running iterations is equal to one of the values of controlling parameters (*DOF, PF, RIF* and *PEF*), the process of search diversification is applied by using the respective operator (*Delete, Perturbate, Restart* and *Penalise*) accordingly.

The number of consecutive iterations without any improvement is tracked continually and when it reaches *30%* of the maximum iterations allowed (as specified by *MI* parameter), the algorithm quits and returns the best found solution. If the number of iterations without improvement never surpasses *30%* of *MI* parameter value then the algorithm completes its full course of iterations.

Note that the source code of the algorithm could be downloaded on the following web page: https://sites.google.com/site/ushtrimet/tourist-trip-planning.

input : TLS, MI, DOF, PF, RIF, PEF
output: MCTOPTW solution

```
1  begin
2      Operator List = {Replace, Insert, Swap};
3      S_c = Create initial solution;
4      Evaluate S_c;
5      S_b = S_c;
6      IterationCounter=0;
7      IterationWithoutImprovement=0;
8      while MI not reached do
9          Select Operator from Operator List;
10         Apply Operator in S_c;
11         if S_c better then S_b then
12             S_b = S_c;
13             IterationWithoutImprovement=0;
14         else
15             IterationWithoutImprovement +1;
16         end
17         if IterationWithoutImprovement equals DOF then
18             Apply delete operator in S_c;
19         if IterationWithoutImprovement equals PF then
20             S_c = Perturbate S_b;
21         if IterationWithoutImprovement equals RIF then
22             Reset S_c to a random initial solution;
23         if IterationWithoutImprovement equals PEF then
24             Penalize frequently used moves;
25         if IterationWithoutImprovement equals 0.3*MI then
26             Quit search;
27         IterationCounter +1;
28     end
29 end
```

Algorithm 3.1: Algorithm for MCTOPTW based on Tabu Search

3.3 Computational Experiments

The algorithm is coded by using Java 1.7. All experiments are done by using an Intel i3 2.2 GHz processor with 2 GB of RAM memory. The experiments for fine tuning of algorithm parameters use 10 runs for each instance, while the experiment for testing the overall algorithm performance uses 30 runs for each instance. The best, average and worst scores are presented for each instance and parameter value. In the following we present a summary of experiments, while the complete results could be found in Appendix B and on-line through https://sites.google.com/site/ushtrimet/tourist-trip-planning.

3.3.1 Test instances and approach comparison

In order to compare the performance of our algorithm with the hybridized ILS-GRASP approach of Souffriau et al. [119], we use their instance set for Multi Constrained Team Orienteering Problem with Multiple Time Windows (MCTOPMTW). The test set consists of 148 instances and, based on the origin, they are divided into two subsets, namely Solomon [116] and Cordeau [25] based. The Solomon-based subset consists of 116 instances, while the Cordeau-based part consists of 32 instances. The number of tours varies from one to four. Note that even though the test instances allow modeling of multiple time windows, in our implementation we only consider single time windows by taking into account the opening time of first time window and closing time of last time window. Hence, we compare the results of our approach with the results obtained by ILS-GRASP approach [119] for MCTOPTW problem and with the best existing results in the literature for the relaxed problems of OP, TOP and TOPTW.

3.3.2 Parameter tuning

The range of values used for parameter tuning is based on preliminary experiments that included all test instances. Further, based on the same preliminary experimental results, we fixed the value of *MI* parameter to 46000 iterations. Increasing further the number of iterations only leads to longer execution time. In Tables 3.2 to 3.6, we present a range of values that are close to the optimal values of the parameters. In the first column the different parameter values are shown, while the next three columns represent the best, average and the worst score gaps from benchmark solution, respectively, averaged over all instances. In the last column, we represent the average execution time.

In our first experiment, we investigated the impact of the tabu list length. Table 3.2 shows that keeping a used move tabu for 10 iterations yields to lower score gap while it causes a slight increase in execution time compared to other considered values. It can be noticed that if the tabu list size is 5, the algorithm degrades its performance, both in terms of solution quality and computational time. For the other values, the average performance of the algorithm might differ only up to 0.1%.

	Gap from best (%)			
TLS	***Best***	***Average***	***Worst***	**Time (S)**
5	5.48	10.01	16.46	6.54
10	5.1	9.27	13.8	6.09
15	5.28	9.33	14.18	5.93
20	5.14	9.31	13.98	5.88
25	5.22	9.36	13.47	5.72

Table 3.2: Tabu list size

Next, we experimented with a range of preselected values for *DOF* parameter. Table 3.3 indicates that the delete operator should be applied once in 30 iterations. Further, if changing the values of *DOF* between 20 up to 90, the average solution quality changes for only 0.47% and the average computation time changes only for 0.3 Seconds.

	Gap from best (%)			
DOF	***Best***	***Average***	***Worst***	**Time (S)**
20	5.31	9.13	13.59	5.78
30	5.11	9.11	13.59	5.93
40	5.22	9.36	13.83	5.93
50	5.48	9.51	14.41	6.02
60	5.34	9.57	14.35	5.93
70	5.37	9.37	13.64	6.04
80	5.39	9.57	14.62	6.03
90	5.3	9.42	14.42	6.08
100	5.59	9.7	14.49	5.97

Table 3.3: Delete operator frequency

Experiments regarding the perturbation and random initialization frequency are given in Tables 3.4 and 3.5. It is indicated that these two operators should not be both applied in the same iteration, as in that case the solution quality worsens for about 3%. The best values for the *PF* and *RIF* pair are 190 and 160, respectively. In terms of value change, the *PF* parameter is less sensitive than *RIF* parameter, as the change of their respective values in the range from 120 to 200, deviates the solution quality for 0.32% in the earlier case and for 0.97% in the later one.

	Gap from best (%)			
PF	***Best***	***Average***	***Worst***	**Time (S)**
120	5.49	9.36	14.05	6.35
130	5.17	9.39	14.2	5.91
140	5.29	9.33	13.91	6.1
150	4.98	9.18	13.44	6
160	5.31	9.15	13.43	5.99
170	8.21	11.51	14.91	3.52
180	5.23	9.25	13.83	5.88
190	5.13	9.07	13.47	5.87
200	5.67	9.25	13.68	5.83

Table 3.4: Perturbation frequency

We also experimented with a range of selected values for the penalization frequency of frequently used moves. In Table 3.6, it can be noticed that the best frequency of penalization is 240. The sensitivity of the algorithm in changing the value of *PEF* inside its pre-defined domain, stands at a low rate of 0.33%, except for value 150 that has a difference of 1.82% compared to the best value of the parameter.

Analyzing the experiments for fine tuning of the parameters, we notice that in general the algorithm is not very sensitive to changes in the parameter values inside their defined domains. However, as claimed in [119], over fitting the parameter values to the current test set might cause

	Gap from best (%)			
RIF	***Best***	***Average***	***Worst***	**Time (S)**
120	5.92	10.01	14.61	5.13
130	5.8	9.56	13.63	5.64
140	5.56	9.55	14.26	5.73
150	8.31	11.79	15.11	3.49
160	5.17	9.06	13.67	6.08
170	5.23	9.27	13.94	5.91
180	5.14	9.11	13.69	5.93
190	5.22	9.04	13.78	6.08
200	5.32	9.19	13.75	5.73

Table 3.5: Re-initialization frequency

	Gap from best (%)			
PEF	***Best***	***Average***	***Worst***	**Time (S)**
150	6.12	11.18	16.6	4.68
180	5.32	9.69	14.58	5.92
210	5.47	9.49	14.23	6.09
240	5.27	9.36	14.3	6.01
270	5.38	9.43	14.33	6.06
300	5.48	9.62	14.36	5.64

Table 3.6: Penalization frequency

the algorithm no to perform well in other test sets or in any possible practical implementation.

3.3.3 Comparisons with the state of the art approach

In order to analyze the algorithm performance for the MCTOPMTW instances, we use the obtained optimal values for the parameters, which are as follows: *TLS=10, MI=46000, DOF=30, PF=190, RIF=160* and *PEF=240*. The experiments are conducted by executing the algorithm 30 times for each instance.

The comparison of the algorithm performance with the state of the art results and with the approach from [119] (in following tagged as ILS-GRASP) is presented in Table 7. To the best of our knowledge, ILS-GRASP approach [119] obtains the best results for the MCTOPTW and MCTOPMTW. Since, we use the set of test instances for MCTOPMTW problem, we compare our algorithm with results obtained by ILS-GRASP approach [119] for MCTOPTW. The results are averaged over specified subsets of test instances. First column shows the number of tours used, while the second column specifies whether Solomon-based or Cordeau-based instances were utilized. The gap from the state of the art results is presented in the next three columns, by showing the best, average and worst performance, respectively. Further, in the next three columns the gap from ILS-GRASP approach [119] is given. The second last column shows the

execution time of approach in ILS-GRASP approach [119], whereas the last column shows the average computation time of our algorithm.

		Gap from best (%)			**Gap from ILS-GRASP (%)**			**Time (S)**	
M	**Test set**	***Best***	***Average***	***Worst***	***Best***	***Average***	***Worst***	***Ten runs ILS-GRASP***	***One run***
1	Solomon	4.07	6.57	13.82	3.82	6.04	12.98	2.68	1.74
1	Cordeau	6.42	14.97	25.84	2.89	9.06	18.04	6.83	5.49
2	Solomon	2.59	6.35	11.8	0.69	2.14	5.82	7.69	3.34
2	Cordeau	5.41	14.5	23.51	-0.8	5.74	12.83	17.85	10.18
3	Solomon	3.3	7.85	13.09	-0.66	1.96	5.53	14.3	5.03
3	Cordeau	6.84	14.18	23.67	0.77	5.35	13.19	32.32	15.28
4	Solomon	3.06	7.85	12.56	-2.2	0.79	4	22.99	6.71
4	Cordeau	6.28	13.88	20.59	-0.62	5.51	10.86	52.13	18.32
1 - 4	Solomon	3.15	7.38	12.71	0.31	2.96	6.97	11.91	4.2
1 - 4	Cordeau	6.27	14.25	22.82	0.59	6.28	13.15	27.28	12.32
All		3.97	9.19	15.38	0.51	4	8.79	15.24	5.96

Table 3.7: Performance results

Authors in [119] compare their approach with the state of the art results that were obtained by various algorithms in the literature for the relaxed problems of OP, TOP and TOPTW [117]. Although the best existing results could not be reached by ILS-GRASP approach, according to the authors their results are very good because the problem at hand is more constrained as it is extended to multiple time windows and it has an extra budget constraint and a multiple category constraint. Comparing our results with those of ILS-GRASP approach, we can conclude that our algorithm is still outperformed by ILS-GRASP approach regarding the average performance in the set of instances. However, our algorithm finds better solutions then ILS-GRASP approach in 70 instances. Since we use different machines (the ILS-GRASP approach uses an Intel Xeon 2.5 GHz with 4 GB of RAM memory), we cannot give an accurate comparison regarding the computation time. However, also our algorithm gives good solutions within few seconds.

The results show that the algorithm performs well, especially with the Solomon-based instances. The average score gap from the best known solutions is 9.19% and it is only 4.00% from the results obtained by ILS-GRASP approach that is tested for the same instance set. For 7% of test instances new best solutions were found, while 18% of best known solutions were found.

3.4 Conclusions

In this chapter, we presented a solution approach for the MCTOPTW problem. The existing test instances were used to analyze the algorithm performance. The Tabu search metaheuristic was applied to solve the problem in conjunction with three basic operators *Insert, Replace* and *Swap*. Four additional operators (*Delete, Perturbate, Restart* and *Penalise*) were applied to escape from local optimum. Additionally, we proposed an efficient technique to speed up the evaluation of the time window constraint.

In a test set of 148 instances, the algorithm average performance, in terms of quality, has a gap of 4% from the state of the art approach [119] for MCTOPTW problem. It is evident that the Solomon-based instances are solved considerably better than the Cordeau-based instances with 2.96% and 6.48% average gap respectively. In 57.43% of test instances, the best solutions of [119] were either found or improved. In 70 instances, our algorithm obtains better best solutions than [119]. Note that in [119] also the waiting times between consecutive visits are allowed. Therefore it's a question whether the results of our algorithm eventually could be still improved if we would not enforce such a constraint as given in Equation 3.4.

The average time of execution of around 6 seconds, makes this algorithm suitable for use in tourism domain for planning personalized tourist trips, where real time response is expected. We can conclude that our proposed Tabu Search based algorithm can be used successfully for solving challenging instances from the literature. Although in average it shows worse performance compared to the state of the art solution, the new best solutions that are found (in the existing test set) indicate that our algorithm is competitive with the approach of Souffriau et al. in [119].

CHAPTER 4

Considering Preferences of Tourist Groups

Sightseeing tours are often done in groups, where tourists enjoy their trip in company with their relatives or friends. Hence, in this chapter, in order to model tourist group tours, we introduce a new problem, as an extension of the existing MCTOPTW problem. Since multiple tours need to be modeled for each tourist in the group, we call the new problem with name Multi Constraint ***Multiple*** Team Orienteering Problem with Time Windows (MC***M***TOPTW).

The new problem (MCMTOPTW) extends the existing problem in the literature (MCTOPTW) with two additional concepts. The first one is consideration of multiple tourists, where their individual preferences about POIs are taken into account, and the second one is introduction of mutual social relationship between the different tourists.

Figure 4.1 shows the modeling of mutual relationship between tourists, and the satisfaction factor of all tourists with all existing points of *Case 1* of the scenario. The social relationship between any two tourists is modeled to be asymmetric, since, in general it is considered that two persons might not be related towards each other with same relationship level. Preliminarily, we define a domain range of 0 to 5 for the values of social relationship (a detailed comparison between different domain ranges is given in the Section 4.3). In addition, the tourists themselves have their own specific interests (satisfaction factors) about POIs. The range of the satisfaction factor of tourists with POIs is define between values 0 and 50. The same domain range for satisfaction factors is used in the test instances in the literature [119]. As it can be seen in Figure 4.1, all tourists have a high social relationship between each other (in all cases the social relationship is 4 or 5), whereas also their preferences about POIs are mostly similar. In reference to *Case 1* of the scenario, in this chapter, we aim to investigate whether it will be better for all three tourists to travel in solo, organized in subgroups or all together.

In the following, we initially present the mathematical modeling of MCMTOPTW problem (Section 4.1), and then, we present a straightforward approach (Section 4.2) to solve this new problem by using the algorithm for solo trip planning that was introduced in the previous chapter. There are three modes of trip itinerary planning that are enabled by this approach, namely *Solo*

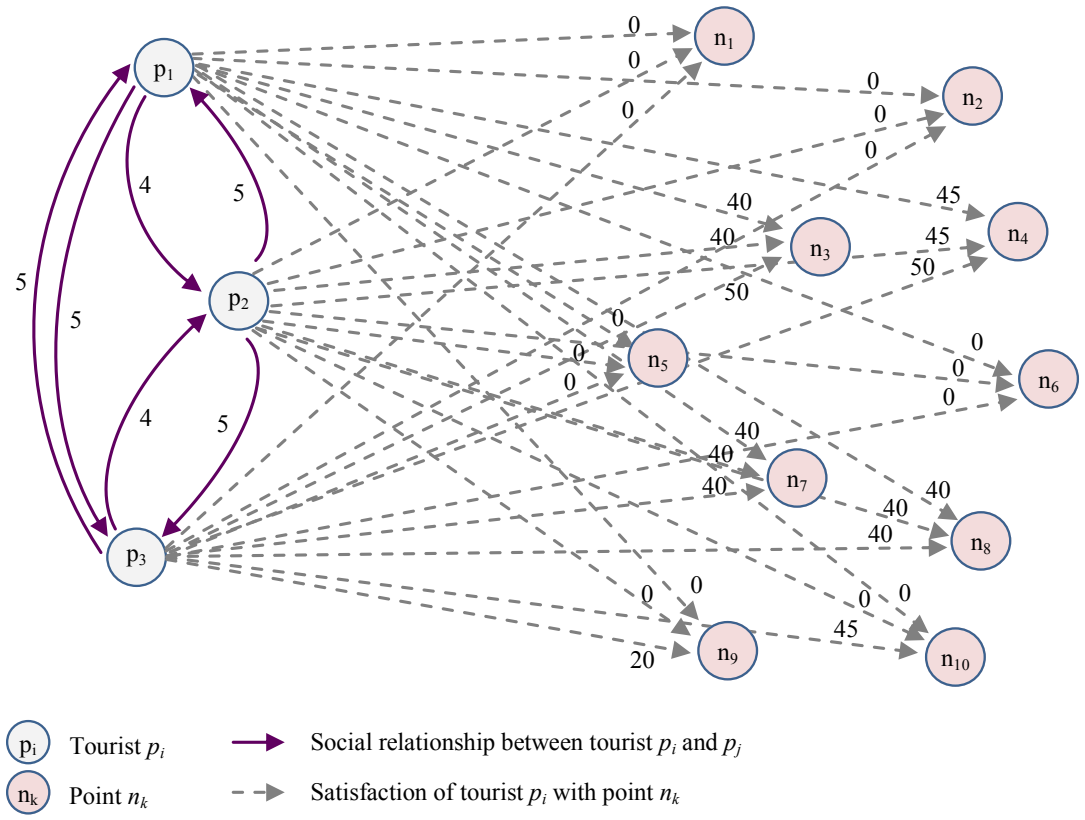


Figure 4.1: Model of social relationship and satisfaction factor with POIs for *Case 1*

(tourists travel alone), *subgroup* (related tourists travel together) and *Group*(all tourists travel together). In addition, we present the computation experiments (4.3) on a new test set that is created based on the instances in the literature. Finally, we present our conclusions (4.4) about application of solo trip planning algorithm in the context of tourist groups.

4.1 Mathematical Modelling of MCMTOPTW problem

A number of P persons make a multiple day trip to a geographic area. The visited area has N points, where each of them is characterized with its coordinates x_i and y_i, visiting duration T_i, opening O_i and closing time C_i, entry fee b_i and an array of Z category attributes e_{nz} that specify the different categories the point belongs to. The details known for each person p include: maximum budget B_p, satisfaction factor with each point S_{pn}, social relationship factor R_{pq} with each other person q in the group, and a range Z of E_{pz} values that specify the maximum number of points of each point category z to be visited. Each person p conducts exactly M tours that have the same duration T_{max}. A person can visit a point at most once. The starting point *1* and ending point N are fixed for each tour. The traveling times t_{ij} between points i and j are known for all points. The start of a visit to point i in tour m of person p is denoted as s_{imp}. The objective

is to prepare a multiple day trip itinerary for all persons so that their overall satisfaction, both with visited points and with each other's company, is maximized.

Next, we give a mathematical definition of the MCMTOPTW problem by extending the formulation made by Garcia et. al [43] for MCTOPTW problem. The Boolean variables x_{ijmp}, y_{pnm} and z_{pqnm} are defined with the following expressions:

$$x_{ijmp} = \begin{cases} 1, & \text{if a visit to point } i \text{ is followed with a visit to point } j \text{ in tour } m \text{ of person } p \\ 0, & \text{otherwise} \end{cases}$$

$$y_{pnm} = \begin{cases} 1, & \text{if person } p \text{ visits point } n \text{ in tour } m \\ 0, & \text{otherwise} \end{cases}$$

$$z_{pqnm} = \begin{cases} 1, & \text{if person } p \text{ and } q \text{ visit together point } n \text{ in tour } m \\ 0, & \text{otherwise} \end{cases}$$

The defined variables x_{ijmp}, y_{pnm} and z_{pqnm} are used in the following formulation equations of the new problem:

$$Max \sum_{p=1}^{P} \sum_{m=1}^{M} \left(\sum_{n=2}^{N-1} S_{pn} y_{pnm} + \sum_{\substack{q=1 \\ q \neq p}}^{P} R_{pq} z_{pqnm} \right), \tag{4.1}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{N} x_{1jmp} = \sum_{m=1}^{M} \sum_{i=1}^{N-1} x_{iNmp} = M;\ \forall p = 1, ..., P, \tag{4.2}$$

$$\sum_{i=1}^{N-1} x_{ikmp} = \sum_{j=2}^{N} x_{kjmp} = y_{pkm};\ \forall k = 2, ..., N-1;\ \forall m = 1, ..., M;\ \forall p = 1, ..., P, \tag{4.3}$$

$$(x_{ijmp} = 1) \implies (s_{jmp} = s_{imp} + T_i + t_{ij});\ \forall i, j = 1, ..., N;\ \forall m = 1, ..., M;\ \forall p = 1, ..., P, \tag{4.4}$$

$$\sum_{m=1}^{M} y_{pnm} \leq 1;\ \forall p = 1, ..., P;\ \forall n = 2, ..., N-1;\ \forall m = 1, ..., M, \tag{4.5}$$

$$\sum_{n=1}^{N} e_{nz} y_{pnm} \leq E_{pz};\ \forall p = 1, ..., P;\ \forall m = 1, ..., M;\ \forall z = 1, ..., Z, \tag{4.6}$$

$$O_i \leq s_{nmp} \leq C_i;\ \forall n = 1, ..., N;\ \forall m = 1, ..., M;\ \forall p = 1, ..., P, \tag{4.7}$$

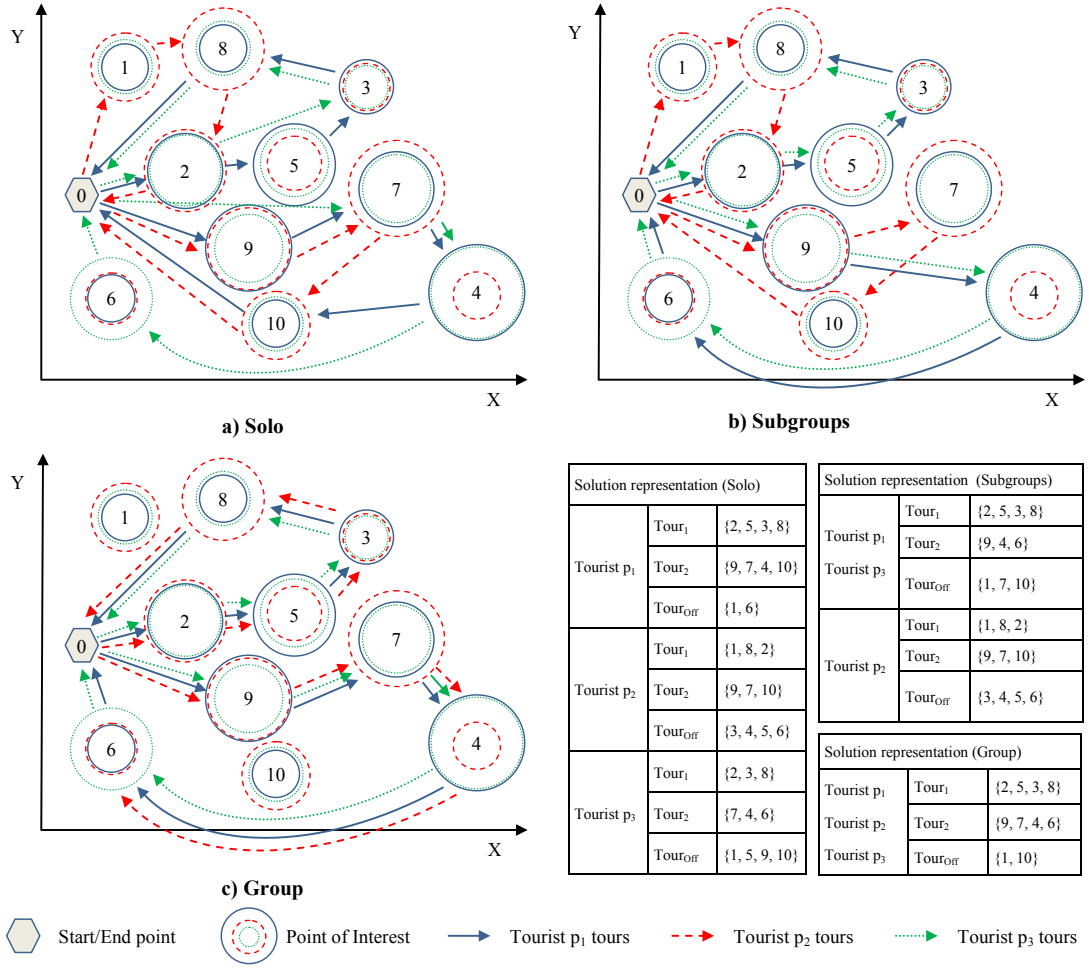
$$\sum_{i=1}^{N-1}\left(T_i y_{imp} + \sum_{j=2}^{N} t_{ij} x_{ijmp}\right) \leq T_{max};\ \forall m = 1, ..., M;\ \forall z = 1, ..., Z, \tag{4.8}$$

Equation 4.1 expresses the objective function of the problem, which is maximisation of the overall tourists' satisfaction, both in terms of satisfaction with points and with each other's company by considering their mutual social relationship factor. Constraint 4.2 enforces the start of each tour at point *1* and the end of it at point *N*. Constraint 4.3 enforces the continuity of a tour by making sure that any visit to a point is followed with another visit to some other point, so that no break up occurs in any of the tours. Constraint 4.4 allows no waiting times in between consecutive visits. Constraint 4.5 makes sure that any person could visit a particular point at most one time, whereas Constraint 4.6 limits the number of points of certain point category that could be visited by a person to the given maximal value E_{pz}. Constraint 4.7 forces the start of a visit to a point to occur only during its respective time windows. Constraint 4.8 limits the duration of any tour to the maximal specified value T_{max}.

4.2 Solution Approach

Given that the new problem of MCMTOPTW extends the existing problem of MCTOPTW, we present a straightforward method that uses the described algorithm from previous chapter to make an initial attempt for solving MCMTOPTW problem. The goal is to devise a method that can enable planning the tour itinerary in three different modes: *Solo* (each tourist makes the tour alone), *Subgroups* (tourists with common interests and constraints travel together) and *Group* (all tourists go together). The method consists of two steps, namely the *preprocessing* and *application of the planning algorithm*. Note that the *preprocessing* step is utilized only when the tour itinerary is planned in the modes of *Subgroups* or *Group* tour. In the mode of *Subgroups*, initially, a tourist subgrouping method is brought into action to subgroup tourists into subgroups based on tourists' preferences (about points and mutual social relationship) and constraints (available budget and maximum number of points for each point category), and then a procedure for merging the data of preferences and constraints of tourist's (belonging to the same subgroups) is applied. On the other hand, in the mode of *Group* tour only a procedure for merging the preferences and constraints of all group members needs to be applied. Further, in the second step, simply, the solo trip planning algorithm is applied for planning the itinerary for all tourists.

Figure 4.2 shows the solution representation for *Case 1* of the scenario for different trip modes. In the *Solo* mode (Figure 4.2 a), each from the three tourists has its own trip itinerary, whereas in the *Subgroups* mode (Figure 4.2 b), *Tourist 1* and *Tourist 3* have a joint itinerary throughout the trip, while *Tourist 2* has its own itinerary, and finally, in the *Group* mode (Figure 4.2 c), all three tourists have a joint itinerary throughout the trip. The score of a point is correlated to the size of the corresponding circle. The larger the circle the higher the score. The score of *Tourist 1* is illustrated with a solid line circle, whereas for *Tourist 2* the illustration is done with a dashed line circle, and lastly the score of *Tourist 3* is illustrated with a dotted line circle.



Solution representation (Solo)		
Tourist p_1	$Tour_1$	{2, 5, 3, 8}
	$Tour_2$	{9, 7, 4, 10}
	$Tour_{Off}$	{1, 6}
Tourist p_2	$Tour_1$	{1, 8, 2}
	$Tour_2$	{9, 7, 10}
	$Tour_{Off}$	{3, 4, 5, 6}
Tourist p_3	$Tour_1$	{2, 3, 8}
	$Tour_2$	{7, 4, 6}
	$Tour_{Off}$	{1, 5, 9, 10}

Solution representation (Subgroups)		
Tourist p_1 Tourist p_3	$Tour_1$	{2, 5, 3, 8}
	$Tour_2$	{9, 4, 6}
	$Tour_{Off}$	{1, 7, 10}
Tourist p_2	$Tour_1$	{1, 8, 2}
	$Tour_2$	{9, 7, 10}
	$Tour_{Off}$	{3, 4, 5, 6}

Solution representation (Group)		
Tourist p_1	$Tour_1$	{2, 5, 3, 8}
Tourist p_2	$Tour_2$	{9, 7, 4, 6}
Tourist p_3	$Tour_{Off}$	{1, 10}

Figure 4.2: Solution representation for different trip modes

In the the rest of this section, we initially describe the tourist subgrouping method and tourist's data merging process, and then elaborate the application of solo trip planning algorithm for the case of tourist groups.

4.2.1 Tourist Subgrouping

The *k-means algorithm* is a rather simple approach to subgroup a dataset into k subgroups. The algorithm has been introduced into several disciplines by many authors, most remarkably by Lloyd in [80] and [81].

The algorithm works on a dataset of d-dimensional vectors, $D = \{x_i \mid i=1\ldots N\}$, where x_i represents the i-th data point. It starts by picking k points in dataset as initial subgroup representatives also called "centroids". The methods used to select these initial centroids vary from random to more sophisticated ones. Then, the algorithm alternates between two steps until it

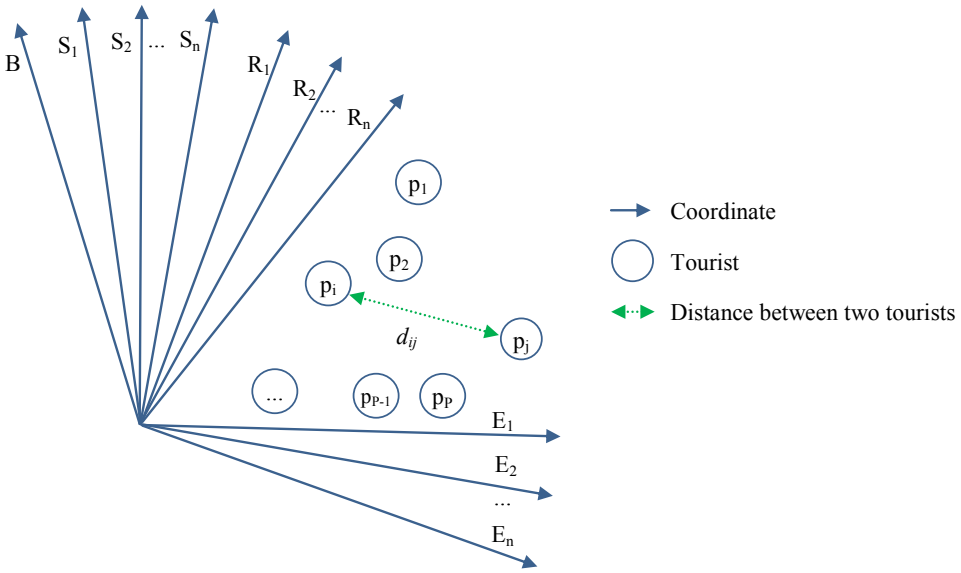


Figure 4.3: Representation of tourists in a n_v dimensional coordinate system

converges. The first step includes assignment of each data point to the closest centroid, which results into portioning of dataset. The second step, deals with relocation of the centroids, by calculation of the new central point of the data points assigned to individual subgroups. The algorithm converges when no more data points change their subgroups. The calculation of distance between data points is an issue to resolve and subject to different domains of implementations. The default distance measure is the Euclidean distance.

In our implementation, the data set consists of tourists' data, where each tourist p is represented by four types of attributes:

- Maximum budget (B_p),
- Satisfaction factor about points (S_{pn}; $n=1, ..., N$),
- Social relationship factor with other tourists (R_{pq}; $q=1, ..., P$; $p \neq q$),
- Maximum number of points for each point category z (E_{pz}; $z=1, ..., Z$)

Given these four types of attributes, the total number of different attributes is $n_v = (N) + (P-1)+(1)+(Z) = N+P+Z$. These attributes are used to represent the individual tourists into a n_v dimensional coordinate system (see Figure 4.3). The measurement of closeness between tourist i and tourist j is calculated by using Equation 4.9 that is based on the Euclidean distance.

$$d_{ij} = \sqrt{(B_i - B_j)^2 + \sum_{n=1}^{N}(S_{in} - S_{jn})^2 + \sum_{p=1}^{P}(R_{ip} - R_{jp})^2 + \sum_{z=1}^{Z}(E_{iz} - E_{jz})^2} \qquad (4.9)$$

In the initiation phase, the *k*-means algorithm (see Algorithm 4.1) randomly selects *k* centroids from the represented tourists in the n_v dimensional coordinate system. Then, the subgrouping is done by assigning the remaining tourists to their closest centroid. Afterwards, the algorithm iteratively alternates between adaption of new centroids for individual subgroups by choosing the most central point inside the subgroup, and then formation of new subgroups based on the distance from newly adapted centroids. The algorithm terminates and returns the result when no more tourists change their subgroups.

The *k*-means algorithm alone is associated with the disadvantage of keeping the number of *k* subgroups fixed. In order to overcome this disadvantage, we implemented an iterative approach (see Algorithm 4.2) that takes the advantage of *k*-means algorithm to create the subgroups and uses an evaluation framework to evaluate the subgrouping process for different number of subgroups.

input : $NumberOfSubgroups, TouristCoordinates$
output: Tourist subgroups

```
1 begin
2     Select random centroids for each subgroup;
3     Create subgroups around each centroid;
4     while some tourist changes the subgroup do
5         Select new centroids for each subgroup;
6         Create new subgroups around each centroid;
7     end
8 end
```

Algorithm 4.1: K-means algorithm for subgrouping tourists

The cluster analysis method of Caliński and Harabasz [18] is used to evaluate which is the best number of subgroups. The method is named *pseudo F-static* F_{ch} and it is based on statistical analysis of subgroups. In the following, we give details about the evaluation function and the comprising components, as well as for the variables used.

$$F_{ch} = (\frac{SST}{SSE} - 1)(\frac{P - n_c}{n_c - 1}), \tag{4.10}$$

$$SST = \sum_{i=1}^{n_c}\sum_{j=1}^{n_i}\sum_{k=1}^{n_v}(V_{ij}^k - \overline{V^k})^2, \tag{4.11}$$

$$SSE = \sum_{i=1}^{n_c}\sum_{j=1}^{n_i}\sum_{k=1}^{n_v}(V_{ij}^k - \overline{V_l^k})^2, \tag{4.12}$$

Where:
SST - total sum of squared distance to the overall mean,
SSE - sum of squared distance of the tourists to their own subgroup means,
P - Number of tourists,

n_c - number of subgroups,
n_i - number of members in subgroup i,
n_v - number of attributes used for representing tourists,
$V_{ij}{}^k$ - value of the k-th variable of the j-th tourist,
$\overline{V^k}$ - mean of the k-th variable,
$\overline{V_l^k}$ - mean over all observations of the k-th variable in subgroup i

In order to increase the probability of obtaining the best possible number of subgroups, the algorithm executes for number of iterations as specified by *MaximumIterations* parameter. In the course of a single iteration, the algorithm executes the *k-means* method for each possible number of subgroups, starting from minimum number of subgroups up to maximum number of subgroups. The best evaluating number of subgroups is found based on *pseudo F-static* F_{ch}. After a series of executions, the algorithm returns the best evaluated subgroups. Based on our experiments, for test instances with maximum 10 tourists, it is sufficient to run the algorithm for 20 iterations to obtain the best evaluating number of subgroups.

input : $MaximumIterations$
output: Tourist subgroups

```
1  begin
2      IterationCounter=1;
3      while IterationCounter <= MaximumIterations do
4          MinimumNumberOfSubgroups=2;
5          MaximumNumberOfSubgroups=NumberOfPersons-1;
6          CurrentNumberOfSubgroups=MinimumNumberOfSubgroups;
7          Sc = Get K-Means Subgrouping (CurrentNumberOfSubgroups);
8          Evaluate Sc;
9          Sb=Sc;
10         while CurrentNumberOfSubgroups <= MaximumNumberOfSubgroups do
11             Sc = Get K-Means Subgrouping (CurrentNumberOfSubgroups);
12             Evaluate Sc;
13             if Sc better then Sb then
14                 Sb=Sc;
15             end
16         end
17         IterationCounter +1;
18     end
19 end
```

Algorithm 4.2: An iterative algorithm based on k-means with automatic selection of the number of subgroups

4.2.2 Tourists' Data Merging

In the cases of subgroup and group trip itinerary planning, the data of tourists in individual subgroups or in the whole group, respectively, need to be merged into a single virtual tourist so that the algorithm for solo trip planning could be applied. In case there are *k* subgroups, then for each subgroup a virtual tourist needs to be created. The process of data merging should consider the hard constraints of each tourist in a particular subgroup or group.

If we suppose a subgroup/group comprised of *X* tourists, the corresponding virtual tourist's attributes for budget limitation (B_v) and maximum number of points for different point categories (E_{vz}), are set to the minimal values of the respective attributes by using Equations 4.13 and 4.14, respectively. On the other hand, the satisfaction factor of the virtual tourist with points (S_{pn}) is calculated as the average value of satisfaction factors of all tourists in the subgroup/group, respectively. Note that the solo trip planning algorithm is not designed to work with the attribute of social relationship between tourists, therefore such attributes are not considered when merging the tourist's data.

$$B_v = Min\{B_1, B_2, ..., B_X\}, \tag{4.13}$$

$$E_{vz} = Min\{E_{1z}, E_{2z}, ..., E_{Xz}\};\ \forall z = 1, ..., Z, \tag{4.14}$$

$$S_{vn} = Average\{S_{1n}, S_{2n}, ..., S_{Xn}\};\ \forall n = 1, ..., N, \tag{4.15}$$

4.2.3 Application of Solo Trip Planning algorithm

In Algorithm 4.3, we present the pseudo code of the method that applies solo trip planning algorithm for the case tourist groups. The algorithm takes an argument named *Tour Type* that determines the mode of algorithm execution among three different possibilities, namely *Solo, Subgroups* and *Group*. When the algorithm is run in the *Solo* mode, it prepares separate itineraries for each tourist so that they could travel alone and visit points of their own specific preferences. If the algorithm is run in *Subgroups* mode, then it will plan a separate itinerary for each tourist subgroup that is created by using the subgrouping method described earlier. The *Group* mode of the algorithm execution produces a joint itinerary for all tourists. The data merging process (described in previous subsection) is used to create virtual tourist when the algorithm is run in *Subgroups* or *Group* mode. A virtual tourist for each subgroup is created in *Subgroups* mode of execution, whereas only one virtual tourist is needed in *Group* mode of execution.

4.3 Computational Experiments

The algorithm is coded by using Java 1.7. All experiments are done by using an Intel i3 2.2 GHz processor with 2 GB of RAM memory. The experiments are done by using a new test set that is created based on existing test sets from the literature.The experiments use 10 runs for each instance. The best, average and worst scores are presented for each test instance. In the following we present a summary of experiments, while the complete results for

input : $TourType$
output: Tour itineraries

```
1  begin
2      if TourType is Solo then
3          NumberOfItineraries = NumberOfTourists;
4      else if TourType is Subgroups then
5          Subgroups=Apply Tourist Subgrouping Algorithm;
6          NumberOfItineraries = NumberOfSubgroups;
7          SubgroupCounter = 1;
8          while SubgroupCounter <= NumberOfSubgroups do
9              VirtualTourists[SubgroupCounter]=Create Virtual Tourist ;
10         end
11     else
12         NumberOfItineraries = 1;
13         VirtualTourist=Create Virtual Tourist ;
14     ItineraryCounter = 1;
15     while ItineraryCounter <= NumberOfItineraries do
16         Itineraries[ItineraryCounter]=Apply Algorithm for Solo Trip Planning ;
17     end
18 end
```

Algorithm 4.3: Application of singlet tourist trip planning algorithm for tourist groups

modes of *Solo*, *Subgroups* and *Group* could be found in Appendix C, Appendix D and Appendix E, respectively. In addition, the experimental results can be accessed on the web page: https://sites.google.com/site/ushtrimet/tourist-trip-planning.

4.3.1 Test set

The existing test set of 148 test instances for Multi Constrained Team Orienteering Problem with Multiple Time Windows (MCTOPMTW) created by Souffriau et al. [119] can only be used to model a solo trip planning problem. Therefore, we further extend this data to enable modeling of a group trip itinerary. In accordance to the test set origin, the instances are divided into two subsets, namely Solomon [116] and Cordeau [25] based. The Solomon-based subset consists of 116 instances, while the Cordeau-based ones consist of 32 instances. The number of tours differs from one to four. Note that even though the test instances allow modelling of multiple time windows, in our implementation we only utilize single time windows by considering the opening time of first time window and closing time of last time window.

In order to model ***multiple*** tourists to the problem of trip planning, an additional attribute *P* that specifies the number of tourists is added. The value of attribute *P* varies from 2 to 10. Further, the preferences of additional tourists are modeled by adding additional satisfaction factors (S_{pn}, *p={2,...,P}, n={1,...,N}*). The newly added satisfaction factor values get a random value in the range of 0 to 50 as in the original test instances. Moreover, the budget limit (B_p, *p={2,...,P}*)

and the maximum number of points for particular point categories (E_{pz}, $p=\{2,...,P\}$, $p=\{1,...,Z\}$) for the additional tourists, are modeled. The budget for the new added tourists is set as a random value in the range of the budget of the existing tourist in the instance, with a possible deviation of plus-minus 100. In addition, the values of the new tourists for the maximum number of points for particular points categories, are also random values in the range of the existing tourist, with a possible deviation of plus-minus 3. In order to model the "connectedness" between the tourists, we added a matrix of asymmetric social relationship values (R_{pq}, $p=\{1,...,P\}$, $q=\{1,...,P\}$, $p \neq q$) that determine how much individual tourist are related/friends to each other. We model this relationship as asymmetric, as in general it is considered that two persons might not have the same relationship/friendship level toward each other. In the following subsection, we make an experimentation for finding out which possible range of social relationship values (R_{pq}) is appropriate in a specific mode (*Solo, subgroup* and *Group*) of trip itinerary planning. Note that the format of instances of this new test set is presented in the web page presented above.

4.3.2 Mode of algorithm execution versus social relationship range

In this experiment, we focus in showing which trip itinerary planning mode is suitable to be applied in some preselected domain ranges of mutual social relationship (*SR*) factors of tourists. In Table 4.1, the first column represents the range of discrete values that are used to represent the social relationship factor between the tourists. The second column indicates the mode of execution of the algorithm. The next three columns represent the best, average and worst algorithm performance, respectively. The last column shows the average time of computation for different *SR* domain ranges and algorithm mode of execution.

SR domain	Mode	Best	Average	Worst	Average time (S)
0 to 1	Solo	6163.89	6068.50	5958.24	40.95
0 to 1	Subgroups	5485.49	5319.74	5141.78	12.69
0 to 1	Group	5301.91	5100.34	4818.48	5.64
0 to 3	Solo	6181.06	6077.46	5965.35	38.40
0 to 3	Subgroups	6062.74	5870.95	5654.93	12.77
0 to 3	Group	6415.11	6169.21	5871.83	5.28
0 to 5	Solo	6193.06	6091.78	5973.70	38.87
0 to 5	Subgroups	6598.26	6368.77	6126.08	13.03
0 to 5	Group	7578.79	7260.53	6838.80	5.15

Table 4.1: Mode of algorithm execution vs. SR range

It can be noticed that for *SR* domain range between 0 to 1, the *Solo* mode of the algorithm outperforms the other two modes by a clear distance, where the average advantage for all instances compared to *Subgroups* and *Group* modes is 12.34% and 15.95%, respectively. This could also be intuitively explained that for poor relationship factors between different tourists, it would be better for them to conduct the trip separately so that they visit the POIs that are more related to their specific preferences.

When the *SR* domain range varies between values 0 to 3, the *Group* mode shows to be a better approach than the other two modes. The results, averaged over the complete test set, indicate that while the advantage in comparison to *Solo* mode is only 1.49%, the advantage from *Subgroups* mode reaches the value of 4.83%.

If the upper bound of *SR* domain range is increased to 5, the *Group* mode becomes superior to the other modes. The dominance, for the complete set of test instances, from *Solo* and *Subgroups* mode stands at 16.10% and 12.28%, respectively. This could be understood by the fact that if the mutual social relationship factor between the tourists increases, their overall satisfaction depends more and more in the component of mutual social relationship (see Equation 4.1). In addition, it is expected that further increase of upper bound of *SR* domain range would intuitively favor utilization of *Group* mode for trip itinerary planning, therefor we do not find it important, from research point of view, to make additional experimentations with *SR* domain ranges higher than 0 to 5.

In general, it can be concluded that dividing tourists into subgroups does not yield to any improvement in their overall satisfaction. This could be explained by considering two different issues. The first, if they get divided into subgroups, they get a decrease in overall satisfaction factor, since a "compromise" in preferences of different subgroup members is made. The second, they also lose points in their overall satisfaction due to the component of mutual social relationship, since by getting divided from some member of the whole group, the respective gain from mutual social relationship to them is lost. Nevertheless, the *Subgroups* mode proves to be a better choice than the *Solo* mode when the SR domain range of 0 to 5 is used. In that case, the difference if favor of *Subgroups* mode is 6.14%.

It is obvious that the computation time is always in favor of *Group* mode, since it only requires a single run of the algorithm for solo trip planning to produce a joint itinerary for all tourists. The average execution time, taken form the complete test set, across different *SR* domain ranges is 39.41 seconds for *Solo* mode, 12.83 seconds for *Subgroups* mode and 5.35 seconds for *Group* mode.

4.3.3 Comparisons between different modes of algorithm execution

In this subsection, we examine the performance of different modes of the algorithm execution for different number of tours (1 to 4 tours) and different number of tourists (2 to 10 tourists). We also present the performance of the the tree modes of algorithm execution when considering different subsets of *Solomon* and *Cordeau* based instances. Further, in the remaining part of this thesis, we adapt the domain range of 0 to 5 for the social relationship factor of tourists.

Initially, we present the performance of three modes of algorithm execution for itineraries with number of tourists ranging from 3 to 10. In this experiment, for comparison purposes, we skip instances with two tourists, since *Subgroups* mode would always divide the two tourists into two subgroups each having one tourist, and this would be the same solution that is obtained by the *Solo* mode of algorithm. Figure 4.4 shows that for tours with three tourists the *Solo* and *Group* mode have equal number of instances, in which they are better, whereas for tours with more than four tourists, the group mode performs better. Additionally, this experiment shows that *Subgroups* mode of execution is outperformed by the two other modes, since, in total, it is better only for eight instances (with 3 to 6 tourists).

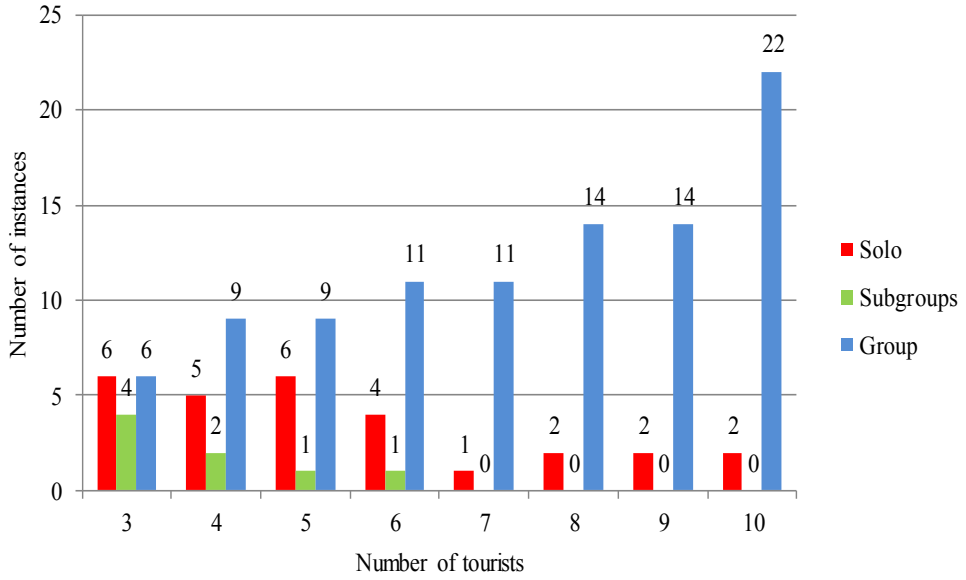


Figure 4.4: Algorithm mode vs. number of tourists

Further, in Figure 4.5 it can be noticed that, in overall, for itineraries consisting of one tour, the *Solo* mode of the algorithm execution performs slightly better, while for higher number of tours the *Group* mode is superior. Further, even though the *Subgroups* mode is outperformed by the *Group* mode for itineraries with all possible number of tours, it proves to be a better choice then the *Solo* mode for itineraries with 3 and 4 tours.

In addition, in Table 4.2, we present the performance of the three modes of algorithm execution for different number of tours (M) and different test subsets (*Solomon* and *Cordeau* based). An in depth analysis of the results in the column of average performance, shows that the *Group* mode performs better than the other two modes in all presented subsets, except for the *Solomon* subset with 1 tour, in which *Solo* mode is better. Further, the *Subgroups* mode is also better in more occasions than the *Solo* mode, especially when dealing with *Cordeau* based test subsets, where for all four of them, the *Subgroups* modes performs better.

4.4 Conclusions

In this chapter, we presented a new problem called "MC***M***TOPTW" that represents an extension to the existing problem of "MCTOPTW". Further, we elaborated a straightforward approach for tackling the new MCMTOPTW problem by using the existing algorithm for solving MCTOPTW problem. A *k-means* based approach was introduced for subgrouping tourists into subgroups.

Mode	M	Test set	Best	Avgerage	Worst	Time (S)
Solo	1	Solomon	2431.55	2392.96	2325.00	11.08
	1	Cordeau	3853.00	3741.51	3529.88	35.22
	2	Solomon	4763.72	4701.33	4623.45	20.53
	2	Cordeau	6991.50	6791.65	6602.25	68.19
	3	Solomon	6841.00	6752.60	6661.59	31.28
	3	Cordeau	9567.38	9379.75	9125.38	103.82
	4	Solomon	8704.07	8587.91	8465.86	42.57
	4	Cordeau	11726.00	11458.88	11230.88	129.60
	1-4	Solomon	5685.09	5608.70	5518.97	21.09
	1-4	Cordeau	8034.47	7842.95	7622.09	84.21
	All		**6193.06**	**6091.78**	**5973.70**	**38.87**
Subgroups	1	Solomon	2242.03	2171.72	2065.00	3.58
	1	Cordeau	4044.50	3877.14	3553.75	11.45
	2	Solomon	4797.52	4668.46	4516.14	7.07
	2	Cordeau	7675.50	7268.95	6941.88	24.04
	3	Solomon	7203.52	6989.93	6808.21	10.78
	3	Cordeau	10534.88	10060.25	9482.88	32.14
	4	Solomon	9584.17	9280.74	8998.76	14.75
	4	Cordeau	13439.25	12839.09	12197.13	42.30
	1-4	Solomon	5956.81	5777.71	5597.03	9.05
	1-4	Cordeau	8923.53	8511.36	8043.91	27.48
	All		**6598.26**	**6368.77**	**6126.08**	**13.03**
Group	1	Solomon	2370.38	2295.70	2029.83	1.35
	1	Cordeau	4682.50	4185.80	3147.13	4.59
	2	Solomon	5309.59	5152.63	4948.00	2.53
	2	Cordeau	8898.25	8314.29	7662.75	8.45
	3	Solomon	8245.03	7973.32	7677.38	4.27
	3	Cordeau	12329.75	11618.37	10795.88	13.36
	4	Solomon	11195.28	10849.44	10492.62	5.79
	4	Cordeau	15986.13	14968.56	13751.25	18.36
	1-4	Solomon	6780.07	6567.77	6286.96	3.49
	1-4	Cordeau	10474.16	9771.76	8839.25	11.19
	All		**7578.79**	**7260.53**	**6838.80**	**5.15**

Table 4.2: Mode of algorithm execution for different instance subsets

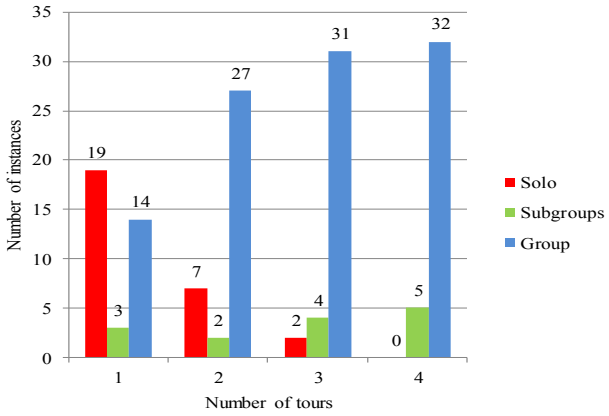


Figure 4.5: Algorithm mode vs. number of tours

The process of subgrouping is evaluated by using the *pseudo F-static* F_{ch} function of Caliński and Harabasz [18]. In addition, we singled out three different modes of algorithm execution so that the tour itinerary could be planned in three different modes, namely *Solo, Subgroups* and *Group*. Further, we introduced new test instances based on the existing instances that are used for solo trip planning problem.

The computational results made by using the newly generated test instances indicate that the *Solo* mode of itinerary planning is more suitable in the cases when the mutual social relationship factor between the tourists is lower (with a domain rang from 0 to 1), whereas for higher mutual social relationship factors, the *Group* mode of itinerary planning becomes superior. The relative short time (in average) of algorithm execution of a little more than 5 seconds for *Group* mode, indicates that the algorithm for solo trip planning could be adapted for use in group trip planning.

Further, the computational results with *SR* domain range of 0 to 5, indicate that subgrouping tourists into subgroups does not prove to be a better choice than the *Group* mode, although it is evident that it is a better choice in comparison to *Solo* mode. However, it would be worth investigating whether using other functions for measuring the distance between tourists, would make the *Subgroups* mode more competitive to the *Group* mode.

In addition, this chapter answers our first hypothesis, where we hypothesized that solo trip planning method could be used for planing the trip for a group of tourists. We can conclude that the first hypothesis is met, since the solo trip planing algorithm can also be used to plan the itinerary for multiple tourists regardless whether they are in subgroups or groups.

Finally, in this chapter the second hypothesis is also answered, which hypothesized that by clustering tourists into subgroup based on their preferences and social relationship the results could be improved. As result, based on the computational experiments that were performed on the generated test set and on the distance measuring function, despite our expectations, we can conclude that in overall the process of clustering tourists into subgroups does not improve the results to an extent that would be better then the results obtained by group approach.

CHAPTER 5

Planning Group Trip Itinerary

All three modes of itinerary planning presented in previous chapter, concentrate in making a solution that predetermines the company of tourists' throughout the trip. They are either all separated, in subgroups or all together. In this chapter, we aim to tackle this drawback by allowing tourists to change their company during the trip, so that they can visit specific POIs of their own interest in the company of their closely related relatives or friends. This is needed when considering frequent situations where some tourists have their own specific preferences about POIs and they are not very much related to some group members.

Revisiting Case 1, it can be noticed that all three tourists are very much related to each other, both in terms of mutual social relationship and in the similarity of their interest about points. Their social relationship values are high (either 4 or 5), whereas the interest values for the corresponding points are always either the same or with value difference of maximum 20, except for point n_{10}, in which the value difference of tourist *p3* from both tourists, *p1* and *p2*, is 45. In this particular case, in order to get a maximal score for tourists, it is obvious that all tourists should be traveling together (according to *Group* mode of the approach from previous chapter). Nevertheless, if we would be able to somehow separate tourist *p3* from the other tourists, so that he can visit point n_{10}, while the other two tourists visit some point of their common interest, we would be able to get an itinerary that would increase the satisfaction of tourist p_3, and with that the satisfaction of the overall group members.

In general, we assume that in situations similar to Case 1, a method that would allow separating or joining tourists during the trip, might make it possible to find better tours for some of the them in the group. Therefore, our approach in this chapter is devising an algorithm that uses the existing algorithm for solo trip planning to create a good joint starting solution, and then applying some effective operators to somehow personalize the trip itineraries for tourists based on their specific preferences. In this chapter, we first present a slight mathematical reformulation of MCMTOPTW problem (Section 5.1), and then show a detailed description of the solution approach (Section 5.2). The computational experiments (Section 5.3) are followed with our conclusions (Section 5.4) about the proposed approach.

5.1 Mathematical Modelling

The mathematical formulation of the MCMTOPTW problem presented in the previous chapter does not allow any waiting time between consecutive visits in a given tour (Constraint 4.4). In this chapter, we relax this constraint by allowing waiting times of tourists before starting a visit to a certain point. The Constraint 4.4, in the modeling of Chapter 4, is replaced with Constraint 5.1, which, instead of not allowing waiting times, makes sure that consecutive visits are aligned in timely manner so that no overlapping occurs between consecutive visits and traveling times between them. Coefficient Q in Constraint 5.1 represents a constant.

$$s_{imp} + t_{ij} - s_{jmp} \leq Q(1 - x_{ijmp});\ \forall i, j = 1, ..., N;\ \forall m = 1, ..., M;\ \forall p = 1, ..., P, \quad (5.1)$$

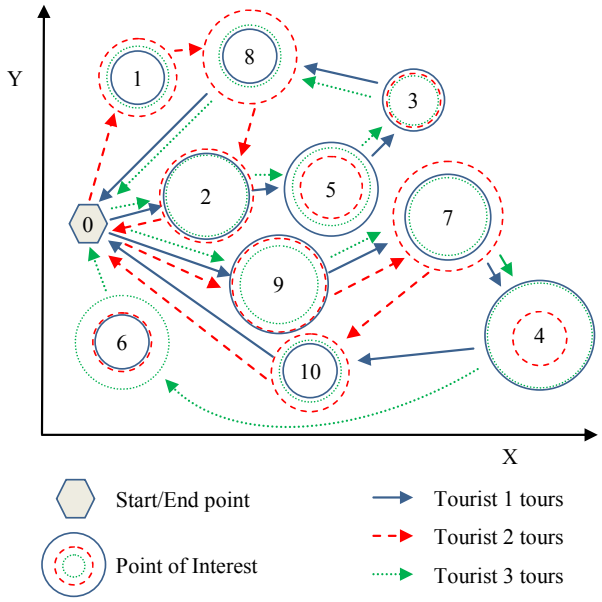
Note that the objective function and all the other constraints remain the same as defined in Section 4.1 of Chapter 4.

5.2 Solution Approach

In Chapter 3, we presented a tabu search implementation that is competitive to state of the art algorithms for solving solo trip itinerary planning problem that is regarded as MCMTOPTW problem. Then, in Chapter 4, we showed that the algorithm for solo trip itinerary planning problem could be applied to plan tour itinerary for tourist groups, which we denoted as MCMTOPTW problem. Further, the *Group* mode of itinerary planning proved to be a better choice when the range of social relationship values is between 0 to 5. In this regard, on top of solo itinerary planning algorithm, we employ again an approach that is based on tabu search metaheuristic to optimize the group tour itinerary planning problem. The approach initially uses the solo itinerary planning algorithm for creating the initial solution that is optimized based on joint preferences of all tourists, and then it iteratively applies a neighborhood exploration mechanism that aims in optimizing the itinerary based on specific preferences of tourists. In the following subsections, we initially show the solution representation, then discuss the operators we use as part of the neighborhood exploration mechanism, and at the end we describe the general structure of our meta tabu search implementation.

5.2.1 Solution Representation

The representation of the solution is done by using an array of P objects, where each object consists of $M+1$ lists. For each tourist p, a single list contains the sequence of points that are part of a tour. The first M lists contain the points that are scheduled to be visited in respective tours, whereas the last list keeps record of points that are not part of the solution of the corresponding tourist. Considering Example 1, which is a problem consisting of $P=3$ tourists, $N=10$ points and $M=2$ tours, a candidate solution might be represented as in following: *Tourist1{Tour1=[2, 5, 3, 8], Tour2=[9, 7, 4, 10], TourOff [1, 6]}*, *Tourist2{Tour1=[1, 8, 2], Tour2=[9, 7, 10], TourOff [3, 4, 5, 6]}* and *Tourist3{Tour1=[2, 5, 3, 8], Tour2=[9, 7, 4, 6], TourOff [1, 10]}* (see Figure 5.1). The score of a point is in direct proportion to the size of the corresponding circle. The larger the



Solution representation		
Tourist1	Tour1	{2, 5, 3, 8}
	Tour2	{9, 7, 4, 10}
	TourOff	{1, 6}
Tourist2	Tour1	{1, 8, 2}
	Tour2	{9, 7, 10}
	TourOff	{3, 4, 5, 6}
Tourist3	Tour1	{2, 5, 3, 8}
	Tour2	{9, 7, 4, 6}
	TourOff	{1, 10}

Figure 5.1: Sample representation of a solution with two tourists and two tours

circle the higher the score. The score of *Tourist 1* is illustrated with a solid line circle, whereas for *Tourist 2* the illustration is done with a dashed line circle, and lastly the score of *Tourist 3* is illustrated with a dotted line circle.

5.2.2 Neighborhood Exploration

The exploration of the neighborhood of a given current solution S_c is performed by using three operators, namely *Separate, Join* and *Insert*. The first two operators are used in local search techniques, whereas the the last one is based on greedy algorithms. In the following part of this subsection, we describe these operators in more details.

5.2.2.1 Separate operator

The *separate* operator tries to separate a tourist p from his current subgroup at point i in tour m (see Figure 5.2). The number of combinations for the *separate* operator is determined by considering every tourist (belonging to a subgroup of at least two members) for separation from his own subgroup at every point i of every tour m.

The operator procedure (see Algorithm 5.1) after initialization of best non tabu and tabu points to a null value, enters into a triple loop that iterates for every tour m, every point i in tour m and every tourist p visiting point i. Then, a list of points (LP_p) that are located close to point i is created based on current consumption time $T_s = t_{(i-1)i} + W_i + T_i + t_{i(i+1)} + W_{(i+1)}$ of tourist p. All available points that can replace point i in the time limit specified by T_s are inserted into LP_p list. In the following, every point j from LP_p list is evaluated against the constraints, such

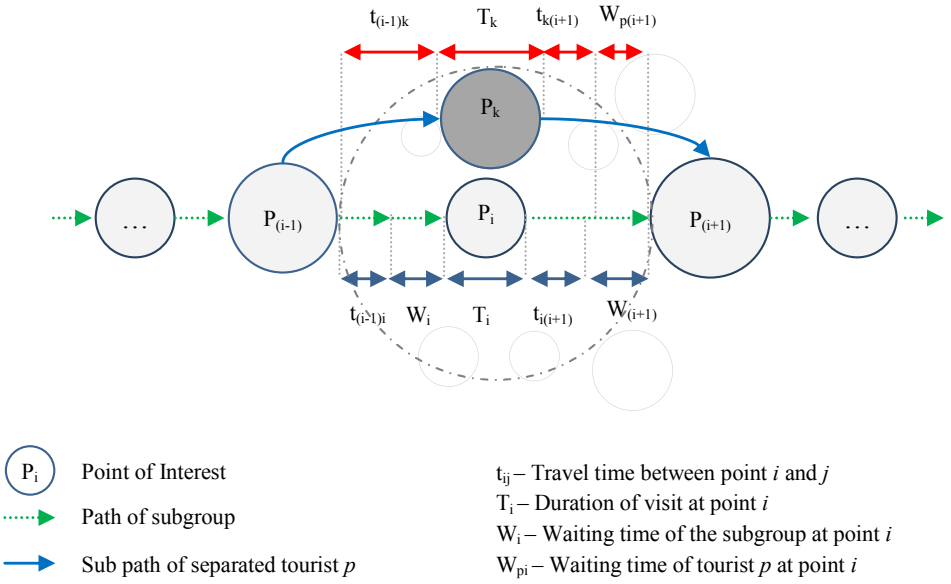


Figure 5.2: Separating tourist p at point P_k

tourist budget, tour duration and maximum point category constraint. Afterwards, every feasible point is evaluated by using the objective function of the problem (expressed by Equation 4.1 in Chapter 4). Based on the tabu status of the evaluated point, a comparison to the best found non tabu or tabu point is made. The operator always records a best non tabu or tabu point when a such point is encountered.

The *separate* operator returns two best separating points, one of them being non tabu and the other one tabu. A returned point is associated with informations such as tourist being separated, point being replaced and position of separation in tour. At the main algorithm level, it will be decided whether the non tabu or tabu point would be adopted as the separating point of tourist.

5.2.2.2 Join operator

The *join* operator considers joining any two tourists belonging to subgroups that visit different points in the same tour. The *join* operator returns two best joining points, one of them being non tabu and the other one tabu (see Algorithm 5.2). The breadth of neighborhood exploration of the *join* operator is determined by the product of the triplet comprising number tourists, number of tours and number of points in each tour.

The operator procedure starts by initializing the best non tabu and tabu points to a null value. Then, for each existing point i in each tour m of each tourist p, a list of points (LP_p) that contains points located close to point i of tourist p, is created. The closeness of other points to point i is calculated by using the respective consumption time $T_{sp} = t_{(i-1)i} + W_i + T_i + t_{i(i+1)} + W_{(i+1)}$ of tourist p (see Figure 5.3). All points that could replace point i in the time specified by T_{sp} are included into LP_p list. Next, a list of other tourists (LOT) that can visit one ore more points

input : $ListOfTourists$, $ListOfPoints$, $NumberOfTours$
output: $BestNonTabuPoint$, $BestTabuPoint$

```
1  begin
2      Initialize BestNonTabuPoint and BestTabuPoint;
3      for m = 1 to NumberOfTours do
4          foreach point i in tour m do
5              foreach tourist p visiting point i in tour m do
6                  Create list of points LP_p that are located close to point i;
7                  foreach point j in LP_p do
8                      if point j satisfies the constraints then
9                          Evaluate point j;
10                         if point j is not tabu then
11                             if point j is better then BestNonTabuPoint then
12                                 BestNonTabuPoint=j;
13
14                         else
15                             if point j is better then BestTabuPoint then
16                                 BestTabuPoint=j;
17
18                         end
19
20                 end
21             end
22         end
23     end
24 end
```

Algorithm 5.1: Pseudo code of *Separate* operator

from LP_p list, is created. The LOT list is populated by selecting tourists that visit at least a point that is in LP_p list. The consumption time $T_{sq} = t_{(k-1)i} + W_k + T_k + t_{k(k+1)} + W_{(k+1)}$ of tourist *q* is used as a time limit within which one ore more points from LP_p list have to be in position to replace the current point of tourist *q*, in order for tourist *q* to be considered as closely located to tourist *p*. In the following, for each identified closely located tourist *q*, a list of points (LP_{pq}) that could be visited by both tourists *p* and *q*, is created. Then, for each candidate point in LP_{pq} list, the algorithm evaluates the feasibility of constraints, such as point time window, budget, maximum point category, tour duration and at most one point constraint. Further, a feasible point is evaluated by using the objective function expressed by Equation 4.1 in Chapter 4. Afterwards, same as in the case of *separate* operator, based on the tabu status of the evaluated point, a comparison to the best found non tabu or tabu point is made. The operator always saves a better non tabu or tabu point when it finds one.

After all tourist have undergone the above procedure, the best non tabu and best tabu points are returned. A returned point is associated with information about: tourists being joined, points

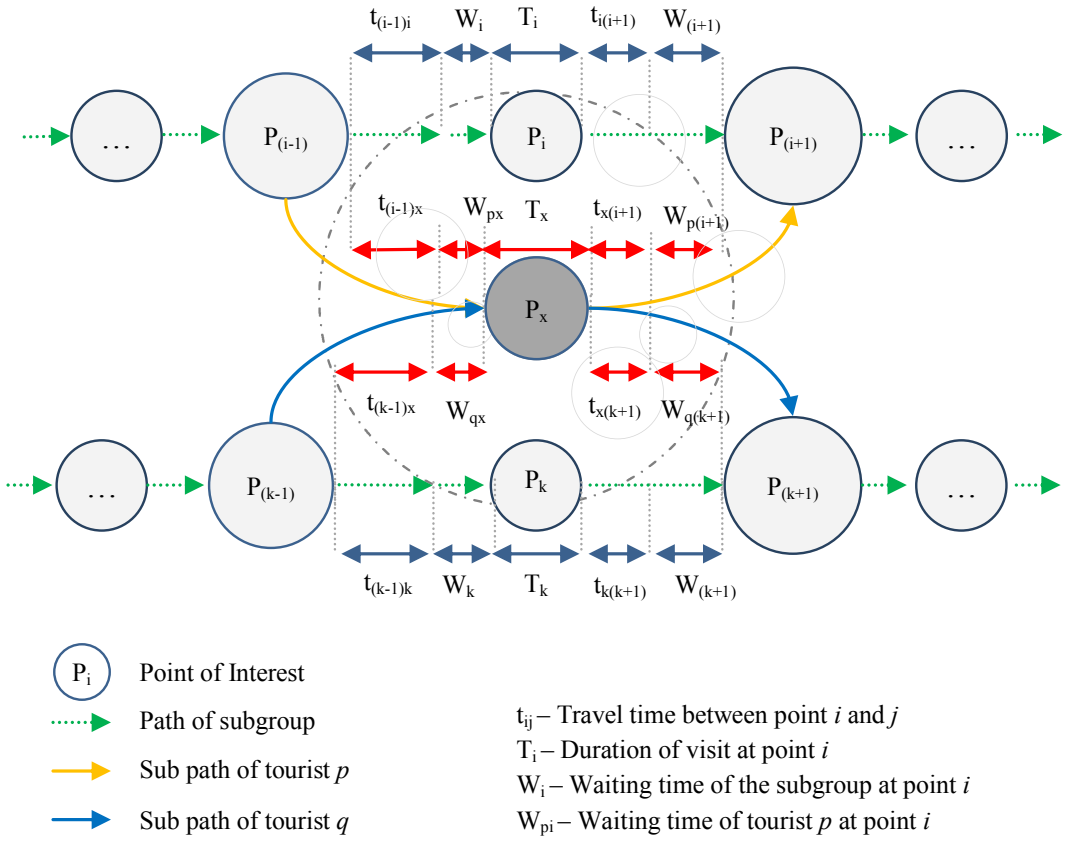


Figure 5.3: Joining tourist p and q at point P_x

being replaced and position of join in the tour. Then, at the main algorithm level, either best non tabu or best tabu solution is selected as the joining point (denoted as P_x in Figure 5.3).

5.2.2.3 Insert operator

The *insert* operator tries to insert some new points for tourists that are scheduled to wait at some points in the tour or have some spare time at the end of the tour. The operator takes a greedy algorithms approach by trying to insert new non included points that have the highest satisfaction factor for the respective tourists.

The procedure of *insert* operator (see Algorithm 5.3) consists of three main loops, which, in turn, enable consideration of each tourist p visiting each point point i in each tour m. In addition, at the end of each tour m, the operator tries to insert new points in the spare time that might be available. During the course of an iteration for tourist p visiting point i in tour m, a list of points (LP_p) that are closely located to point i is created. All points that could be inserted in between point *i-1* and point i, in the time defined by $T_{sp} = t_{(i-1)i} + W_i$, are included into LP_p list (see Figure 5.4). Afterwards, LP_p list is filtered by removing points that do not meet problem

input : $ListOfTourists$, $ListOfPoints$, $NumberOfTours$
output: $BestNonTabuPoint$, $BestTabuPoint$

```
1  begin
2      Initialize BestNonTabuPoint and BestTabuPoint;
3      foreach tourist p in the ListOfTourists do
4          for m = 1 to NumberOfTours do
5              foreach point i in tour m of tourist p do
6                  Create list of points LP_p that are located close to point i of tourist p;
7                  Create list of other tourists LOT that can visit points in LP_p;
8                  foreach tourist q in LOT do
9                      Create list of points LP_pq from LP_p that could be visited by tourist p
                       and q;
10                     foreach point j in LP_pq do
11                         if point j satisfies the constraints then
12                             Evaluate point j;
13                             if point j is not tabu then
14                                 if point j is better then BestNonTabuPoint then
15                                     BestNonTabuPoint=j;
16                                 end
17                             else
18                                 if point j is better then BestTabuPoint then
19                                     BestTabuPoint=j;
20
21                             end
22
23                     end
24                 end
25             end
26         end
27     end
28 end
```

Algorithm 5.2: Pseudo code of *Join* operator

constraints. Then, if LP_p list does not become empty, the point with the highest score for tourist *p* from LP_p list is added before point *i* in tour *m*. If a new point is added into current tour *m*, then the next iteration reconsiders again point *i* to check whether there is still room left before it for insertion of new points.

5.2.3 Tabu Memories

For experimentation purposes, we design three different tabu memories, which differ in terms of restrictiveness they pose to the search process. There are two two-dimensional memories,

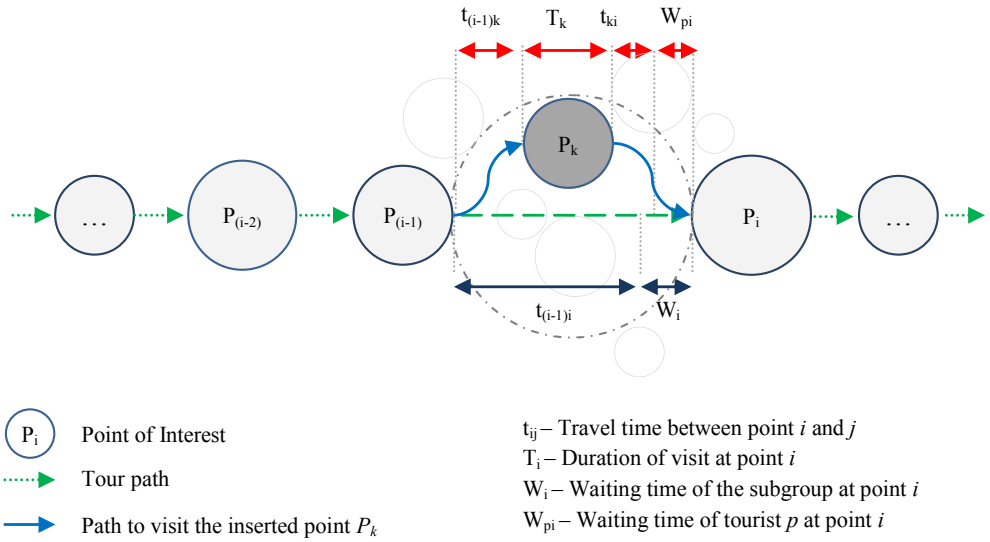


Figure 5.4: Inserting point P_k in the tour path of tourist p

input : $ListOfTourists$, $ListOfPoints$, $NumberOfTours$
output: Solution with new inserted points

```
1  begin
2      for m = 1 to NumberOfTours do
3          foreach point i in tour m do
4              foreach tourist p visiting point i in tour m do
5                  Create list of points LP_p that are located close to point i;
6                  Remove points that do not satisfy the constraints from LP_p;
7                  Select best scoring point k from LP_p;
8                  Add point k before point i in tour m of tourist p;
9              end
10         end
11     end
12 end
```

Algorithm 5.3: Pseudo code of *Insert* operator

namely the *tourist based memory* (see Figure 5.5 a) that is used to record operator information about two different tourists and the *point based memory* (see Figure 5.5 b) that keeps record of operator information whenever an operation between two points occurs, and a three-dimensional one, called *tourist and point based memory* (see Figure 5.5 c) that saves operator information about two tourists being involved in an operation at a certain point. In tourist trip planning situations that we consider in this PhD thesis, the number of tourists is usually much less than the number of points, therefore the *tourist based memory* is more restrictive than the *point based*

memory, since there is more probability that more tourists than points would be included into respective memories. On the other hand, the *tourist and point based memory* can be classified as the least restrictive one, since there is a low probability that two tourists would be very often subject of an operator at the same point. In Figure 5.5, a cell, denoted as TBM_{ij}, PBM_{ij} or $PTBM_{ijk}$, shows the iteration when a certain move was last time applied between tourists p_i and p_j, points n_i and n_j or tourists p_i and p_j at point n_k, respectively.

In our encoding, we save the value of current algorithm iteration into respective cells/cell of memory, whenever the *Separate* or *Join* operator is applied. In the case of *Separate* operator, all the cells that match the separated tourist with subgroup members from whom she/he has been separated, are updated. Whereas, in the case of *Join* operator, only the cell that matches the two tourists being joined is updated. Further, if the *tourist and point based memory* is utilized, then the cells/cell of respective matrix that represents the point where the separation or joining occurs, are/is updated. If the value of a cell in a memory equal to value zero that indicates that the respective tourists (points, or tourists and point) are not used at all by the operators during the search process, therefor eventual application of an operator between them is allowed (not tabu). Conversely, when the value of a cell is grater than zero, then, that indicates that the respective tourists (points, or tourists and point) are used at least once either by *Separate* ore *Join* operator, and the corresponding value of the cell shows the latest iteration when an operator between the tourists (points, or tourists and point) was utilized. In this case, in order to find out whether a certain operator between two tourists (points, or tourists and point) is tabu, we initially define a delta function $\triangle D$ (Equation 5.2) that finds the difference between the current iteration and the iteration when the operator was applied last time for them. Then, we define a function with the name *Tabu Status (TS)* as expressed by Equation 5.3, which uses the $\triangle D$ function to calculate the tabu status. This function returns a *true* value in case the operator between tourists (points, or tourists and point) is tabu, otherwise it returns a *false* value.

$$\triangle D = \begin{cases} I - TM[p_i][p_j], & \text{tourist based memory} \\ I - TM[n_i][n_j], & \text{point based memory} \\ I - TM[p_i][p_j][n_k], & \text{tourist and point based memory} \end{cases} \tag{5.2}$$

$$TS = \triangle D < TLS \tag{5.3}$$

Where:
p_i - Tourist *i* number,
p_j - Tourist *j* number,
n_i - Point *i* number,
n_j - Point *j* number,
n_k - Point *k* number,
I - Current iteration number,
TM - Tabu memory (two or three dimensional array),
TLS - Tabu List Size (algorithm parameter described in next subsection)

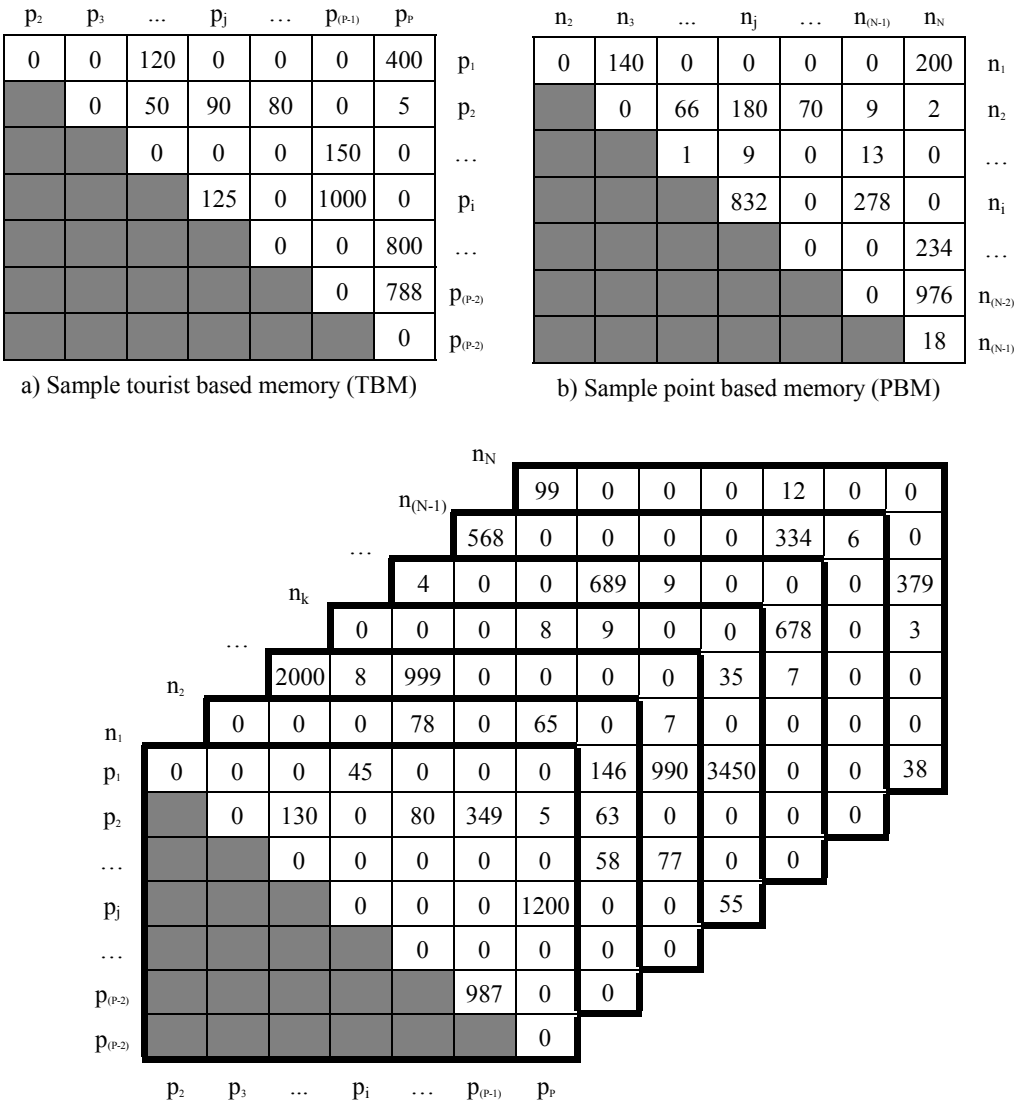


Figure 5.5: Sample representation of tabu memories

5.2.4 Tabu Search Implementation

The algorithm can be fine tuned by using four different parameters. Table 5.1 gives more details about the utilized parameters, where first column shows the abbreviation, second column represent the full name, whereas the third column describes the purpose of using a particular parameter.

Abbreviation	Name	Description
TLS	Tabu List Size	Specifies the number of iterations a certain move cannot be used.
MI	Maximum Iterations	Defines the number of iterations the algorithm shall run.
RIF	Re-Initialization Frequency	Indicates the frequency of re-initialization from a new solution.
OSF	Operator Switching Frequency	Sets the frequency of switching between *Separate* and *Join* operators.

Table 5.1: Algorithm parameters

The procedure of the algorithm (see Algorithm 5.4) starts by setting the tabu memory at an empty state and defining a list of possible operators. Then, the data of all group members are merged into a single virtual tourist with the tourist's data merging procedure described in subsection 4.2.2 of Chapter 4, and then the *Group* mode of the algorithm for solo trip planning is applied, in order to create a joined initial solution, which is denoted as current solution S_c. The evaluation of the quality of a group solution is done by using the objective function expressed by Equation 4.1 (presented in Chapter 4). Next, the algorithm starts its iterative procedure by entering into a loop that runs as far as the maximum number of iterations allowed (specified by *MI* parameter) is not surpassed. During the course of an iteration, the algorithm applies either *Separate* or *Join* operator. Since, the starting solution is a group solution, where all tourists are joint together, it becomes obvious that the *Separate* operator should be applied in the first couple of iterations as specified by *OSF* operator. Then, the algorithm alternates, every *OSF* iterations, between applying *Separate* or *Join* operator. Same as with the algorithm for solo trip planning, in each iteration, a best tabu and non-tabu solution is returned by the respective operator (*Separate* or *Join*). The best non tabu solution is considered first for adaption as current solution. If the non-tabu solution is not better than the current solution, then the best tabu solution is tested whether it fulfills the aspiration criteria. Also, in this implementation, a tabu solution is considered to be fulfilling the aspiration criteria only if it is better than the best found solution so far.

Whenever *Separate* or *Join* operator is applied, it might happen that some tourists get scheduled to wait for some time for the other tourists so they can start the next visit together. In order to try to reduce or eliminate this waiting time, we apply the *Insert* operator to try and insert new points in the margins of the waiting time.

Further, in order increase the probability of not getting stack in some are of search space, a random re-initiation procedure foresees restarting the search process from a new initial solution by re-using the solo trip planning algorithm. This procedure is repeated every *RIF* iterations. The solution that is returned by the solo trip planing algorithm is a group solution that is optimized based on preferences and constraints of all tourists. Since the returned solution represents only a starting solution that is used by group trip planning algorithm, we employ the solo trip planing algorithm with its reduced value of *MI* parameter. As shown in Subsection 3.3.2 of Chapter 3, the best value of *MI* parameter of the solo trip planning algorithm was 46000 iterations, but

when the algorithm is called to generate the starting solution for group trip planner, the value of its respective *MI* parameter is set to 10000 iterations. This is reasoned based on two different aspects. The first, it shortens the overall computation time, and the second, since the returned solution is used as a starting solution, it does not fully optimize the starting solution based on the objective function of solo trip planner, thus retaining a degree of diversity in the initial solution for different algorithm callings.

The algorithm quits either if it reaches 30% of iterations without any improvement or if it makes the foreseen maximum iterations by *MI* parameter.

Note that the source code of the algorithm could be downloaded on the following web page: https://sites.google.com/site/ushtrimet/tourist-trip-planning.

5.3 Computational Experiments

In order to compare the approach presented in this chapter with the approaches from Chapter 4, we conduct the computational experiments by using the same test of 148 instances that is described in Section 3.3 of Chapter 4. The algorithm is coded by using Java 1.7. All experiments are done by using an Intel i3 2.2 GHz processor with 2 GB of RAM memory. A single experiment is conducted by making 10 runs for each instance in the test set. The best, average and worst scores are presented for each instance and parameter value. In the following we present a summary of experiments, while the complete results could be found in Appendix F and Appendix G and on-line through https://sites.google.com/site/ushtrimet/tourist-trip-planning.

5.3.1 Parameter tuning

Based on some preliminary experiments that included a subset of test instances, we selected a range of values for tuning the values for each of the parameters of the algorithm. The value of *MI* parameter is fixed to 10000 iterations. Increasing further the number of iterations only leads to longer execution time.

The *RIF* parameter, as described in previous section, determines the frequency of reinitialization of search process from a new solution that is generated by the algorithm for solo trip planning. Hence, the *RIF* parameter will determine how often the solo trip planning algorithm will be called. Even though, the solo trip planning algorithm is called with a reduced value of *MI=10000*, it still requires an average time of execution of about 1.2 seconds. Thus, calling the algorithm for solo trip planning very often makes the algorithm for group trip planning execute more slowly. Therefore, based on the experiments with the complete data set and in accordance to the selected value for the *RIF* parameter, we identify two different modes of group algorithm execution, namely slow and fast mode. The algorithm operates in the slow mode when the value of *RIF* parameter equals 5, whereas if the value of *RIF* parameter equals 40 the algorithm executes in the fast mode. In the next subsection, we give a detailed comparison of the complete data set for the two selected values of *RIF* parameter.

In the following, the approach presented in this chapter is tagged with the name “Combined” by considering that the produced trip itineraries are often combined between the tourists in the

input : TLS, MI, RIF, OSF
output: MCMTOPTW solution

begin
  $OperatorList = \{Separate, Join, Insert\}$;
  S_c = Create initial solution;
  Evaluate S_c;
  S_b = S_c;
  $IterationCounter = 0$;
  $IterationWithoutImprovement = 0$;
  $IterationWithoutChangeOfCurrentSolution = 0$;
  while $IterationCounter <= MI$ **do**
    Select $CurrentOperator$ from $OperatorList$;
    Apply $CurrentOperator$ in S_c;
    Apply $Insert$ operator in S_c;
    if S_c *better then* S_b **then**
      $S_b = S_c$;
      $IterationWithoutImprovement = 0$;
    else
      $IterationWithoutImprovement + 1$;
    end
    if S_c *has changed* **then**
      $IterationWithoutChangeOfCurrentSolution = 0$;
    else
      $IterationWithoutChangeOfCurrentSolution + 1$;
    end
    if $IterationWithoutChangeOfCurrentSolution$ *equals* RIF **then**
      Reset S_c to a random initial solution;
    if *IterationWithoutImprovement equals 0.3*MI* **then**
      Quit search;
    $IterationCounter + 1$;
  end
end

Algorithm 5.4: Algorithm for MCMTOPTW based on Tabu Search

group. Further, the presented results are averaged over 10 executions for each of the instances in the data set. The algorithm is executed in the fast mode by setting *RIF* parameter value to 40.

In Table 5.2, we present the experimental results that are done by using the three different types of tabu memories and five different tabu list sizes. The first column shows the type of tabu memory, whereas the second column displays the size of tabu list. In the next three columns, we present the gap of the *Combined* approach from the three approaches from Chapter 4, namely *Solo*, *Subgroups* and *Group*. A positive value of the gap means that the *Combined* approach has the advantage in comparison to the other compared approach. In the last column, we represent

the average execution time of the *Combined* approach.

		Gap			
Type of tabu memory	**TLS**	***Solo***	***Subgroups***	***Group***	**Average time (S)**
Tourist based	5	18.36%	14.65%	2.70%	10.59
	10	18.37%	14.66%	2.71%	10.28
	15	18.29%	14.58%	2.62%	9.48
	20	18.39%	14.68%	2.73%	9.00
	25	18.50%	14.80%	2.87%	9.50
Point based	**5**	**18.58%**	**14.87%**	**2.96%**	**9.56**
	10	18.56%	14.86%	2.94%	9.27
	15	18.55%	14.85%	2.93%	9.10
	20	18.45%	14.74%	2.81%	8.76
	25	18.47%	14.76%	2.83%	8.83
Tourist and Point based	5	18.26%	14.54%	2.58%	9.43
	10	18.46%	14.75%	2.81%	9.39
	15	18.56%	14.86%	2.94%	9.23
	20	**18.58%**	**14.88%**	**2.96%**	**9.03**
	25	18.52%	14.81%	2.88%	8.94

Table 5.2: Tabu list size

Based on current experiments, the results in Table 5.2 indicate that *TLS=5* of point based memory and *TLS=20* of tourist and point based memory produce the best results in comparison to the other considered values. In those two cases, the gap of *Combined* approach from approaches such as *Solo*, *Subgroups* and *Group* is 18.58%, 14.87% and 2.96%, respectively, except for *TLS=20* of tourist and point based memory, which is slightly better (14.88%) when *Combined* and *Subgroups* approaches are compared. Further, since for *TLS=20* of tourist and point based memory, the average execution time of algorithm is 9.03 seconds, which is for 0.53 seconds faster than in the case of *TLS=5* of point based memory, we conclude that utilizing a tourist and point based memory with a TLS value of 20 is the most favorable combination out of all other considered combination of tabu memories and tabu list sizes. In addition, the results show that a tourist based memory, besides producing worse solutions than the other two memories, it also causes the algorithm to execute a little slower, especially when the size of tabu list is short (*TLS=5* and *TLS=10*).

Further, Table 5.3 shows the advantage of *Combined* mode over the three modes of Chapter 4 in respect to a range of different values of *OSF* parameter. The selection of values for the experiment is done based on some preliminary experiments with a subset of test instances. In this experiment, the first column presents the value of *OSF* parameter, while the other columns are denoted same in the previous experiment.

In addition, based on current experiments, the experimental data in Table 5.3 indicate that, in terms of quality of solutions, the range of higher selected values (from 50 to 110) yields considerably better results than the lover range of selected values (from 3 to 40), but the execution time in those cases is much slower (at times, it is more than three times slower than for the

	Gap			
OSF	***Solo***	***Subgroups***	***Group***	**Average time (S)**
3	18.50%	14.80%	2.87%	9.35
5	**18.59%**	**14.89%**	**2.97%**	**9.26**
10	18.53%	14.82%	2.89%	9.07
20	18.53%	14.82%	2.90%	8.94
30	18.37%	14.66%	2.71%	8.79
40	18.26%	14.55%	2.58%	8.65
50	19.12%	15.44%	3.61%	19.84
60	19.18%	15.50%	3.67%	22.86
70	19.32%	15.65%	3.84%	24.95
80	19.45%	15.79%	3.99%	23.68
90	**19.49%**	**15.83%**	**4.04%**	**27.60**
100	19.22%	15.54%	3.72%	28.94
110	19.34%	15.68%	3.87%	31.64

Table 5.3: Operator Switching Frequency

lower range of values). Such results might be correlated to the value of *RIF* parameter, which, for this experiment, is set to 40. There is an indication that as soon the *OSF* parameter values surpasses the value of *RIF* parameter, the quality of results increases in expense of a noticeable increase of the computation time. This result suggests that in order to get solutions that are better in terms of quality, the value of *OSF* parameter should be higher than value of *RIF* parameter, although in that case the computation time increases too. According to the obtained results, the most favorable value of *OSF* parameter is 90, which is a little more than the double value of *RIF* parameter.

On the other hand, the lower range of selected values of *OSF* parameter has the advantage in terms of computation time, which is 2.85 times faster then in the case of the higher range of values. As result, in order to enforce a compromise between the quality of solutions and computation time, we alternatively select two best values for the *OSF* operator, one being the best value from the lower range (*OSF=5*) and the other one being the best value from the higher range (*OSF=90*). We set the alternation frequency of *OSF* parameter to 5% of the maximum number of iterations ($F = (5/100) * MI$). Thus, the alternation between the selected values of *OSF* occurs every *F* iterations. In the first *F* iterations, *OSF* takes value 5, and then, in the next *F* iterations, it uses value 90. This alternation process continues as far as the algorithm runs.

5.3.2 Comparison with the previous approaches

In this subsection, we present the performance of the group trip algorithm (tagged as "Combined") in comparison to the three approaches from the previous chapter, namely *Solo*, *Subgroups* and *Group*. Further, we present a comparison between the slow and fast mode of the execution of the *Combined* approach.

Tables 5.4 and 5.5 present the comparison results of the *Combined* approach with the previous approaches, in terms of quality of solutions and time of execution, respectively. The results are aggregated based on the instance types (originated from Solomon or Cordeau) and on the number of tours in the instances. The test instances that belong to a certain type and have certain number of tours are aggregated into a single value by calculating the average value over 10 executions for each instance in the subset.

In table 5.4, the first column depicts the names of the methods being compared, whereas the second and third column show the number of tours and the type of the aggregated test instances, respectively. The last two columns show the average advantage of slow (*RIF=5*) and fast (*RIF=40*) mode from the three previous modes, over the instances in a particular subset.

In terms of quality of solutions, the obtained results indicate that both modes (slow and fast) of "Combined" approach yield to better results than any of the approaches from Chapter 4, namely *Solo*, *Subgroups* and *Group*. The comparison results of *Combined* approach with the *Solo* approach show that the difference is quite large (20.68% for slow mode and 19.48% for the fast mode). This can be explained by the fact that *Solo* approach does not consider social relationship between tourists in the group, and as result, the overall satisfaction of tourists is calculated only based on the satisfaction of tourists with POIs. The difference margin of the *Combined* approach with the *Subgroups* approach is narrower (17.08% for slow mode and 15.82% for the fast mode), since *Subgroups* approach takes into account the social relationship between the tourists into the subgroups, but as we discussed in the previous chapter, in most of the test instances, it is not a favorable method. Lastly, the *Combined* approach shows that it can even obtain better results (5.47% for slow mode and 4.03% for the fast mode) than the *Group* approach, which in overall proved to be better than the *Solo* and *Subgroups* approach.

Further, it is obvious that in comparison to all three previous approaches, the *Combined* approach (in both modes) makes more improvement in the Cordeau based instances than on the Solomon based instances. Such results, highlight the strength of the *Combined* approach in solving difficult instances such as Cordeau based instances.

In reference to the time of execution, Table 5.5 shows the comparison results of all four approaches. First and second column of Table 5.5 present the number of tours and instance type of the test subset, respectively. The next three columns show the average execution time of the previous approaches (*Solo*, *Subgroups* and *Group*), whereas the last two columns present the execution time of *Combined* approach in both modes (Slow and Fast).

The experimental data in Table 5.5 show that the slow mode of *Combined* approach (as the best performing approach in terms of quality), is the most intensive it terms of computation time by requiring in average a little more than 150 seconds per execution, which is by multiple times higher than of all the other presented approaches. On the other hand, in average, the fast mode of the *Combined* approach is eight time faster then the slow mode and it is also two times faster then the *Solo* approach, while being slower than the *Subgroups* approach (less than two times) and the *Group* approach (around four times). In addition, for both modes of the *Combined* approach, the computation time for the subset of Coredeau based instances is much higher than for the subset of Solomon based instances. The ratio of computation time between Coredeau and Solomon based instances is around 5 for slow mode and around 2 the for fast mode.

In terms of quality of solutions, in Table 5.6, we give a more detailed description between

			Gap	
Comparison	**M**	**Test set**	***RIF=5***	***RIF=40***
Combined vs. Solo	1	Solomon	5.30%	4.28%
	1	Cordeau	21.14%	19.81%
	2	Solomon	13.36%	12.29%
	2	Cordeau	25.86%	23.40%
	3	Solomon	18.73%	17.64%
	3	Cordeau	25.58%	24.20%
	4	Solomon	23.49%	22.28%
	4	Cordeau	28.10%	27.40%
	1-4	Solomon	18.38%	17.24%
	1-4	Cordeau	26.09%	24.75%
	All		**20.68%**	**19.48%**
Combined vs. Subgroups	1	Solomon	14.05%	13.13%
	1	Cordeau	18.28%	16.90%
	2	Solomon	13.97%	12.90%
	2	Cordeau	20.65%	18.02%
	3	Solomon	15.87%	14.75%
	3	Cordeau	20.18%	18.70%
	4	Solomon	17.32%	16.01%
	4	Cordeau	19.44%	18.66%
	1-4	Solomon	15.92%	14.75%
	1-4	Cordeau	19.79%	18.34%
	All		**17.08%**	**15.82%**
Combined vs. Group	1	Solomon	9.15%	8.17%
	1	Cordeau	11.78%	10.29%
	2	Solomon	5.05%	3.87%
	2	Cordeau	9.24%	6.23%
	3	Solomon	4.04%	2.75%
	3	Cordeau	7.81%	6.11%
	4	Solomon	3.34%	1.82%
	4	Cordeau	6.08%	5.17%
	1-4	Solomon	4.42%	3.09%
	1-4	Cordeau	7.91%	6.24%
	All		**5.47%**	**4.03%**

Table 5.4: Quality of solutions for different modes of algorithm execution

		Time (S)				
					Combined	
M	**Test set**	***Solo***	***Subgroups***	***Group***	***Slow***	***Fast***
1	Solomon	11.08	3.58	1.35	37.27	8.48
1	Cordeau	35.22	11.45	4.59	119.84	18.61
2	Solomon	20.53	7.07	2.53	67.25	12.54
2	Cordeau	68.19	24.04	8.45	230.22	31.85
3	Solomon	31.28	10.78	4.27	101.46	18.17
3	Cordeau	103.82	32.14	13.36	264.17	42.53
4	Solomon	42.57	14.75	5.79	127.51	23.86
4	Cordeau	129.60	42.30	18.36	1028.61	59.50
1-4	Solomon	21.09	9.05	3.49	83.37	15.76
1-4	Cordeau	84.21	27.48	11.19	410.71	38.12
All		**38.87**	**13.03**	**5.15**	**154.15**	**20.60**

Table 5.5: Time of algorithm execution for different modes

the two modes of the *Combined* approach. While the first two columns are the same as in the previous table, the last three columns represent the gap between the results of the slow and the fast mode of the *Combined* approach. A positive gap means that the slow mode obtains better results, otherwise the fast mode performs better.

		Gap		
M	**Test set**	***Best***	***Avgerage***	***Worst***
1	Solomon	0.86%	1.07%	1.02%
1	Cordeau	1.13%	1.66%	3.63%
2	Solomon	0.65%	1.22%	2.71%
2	Cordeau	2.64%	3.22%	6.02%
3	Solomon	0.87%	1.32%	2.18%
3	Cordeau	0.91%	1.82%	4.39%
4	Solomon	0.41%	1.55%	2.83%
4	Cordeau	-0.09%	0.96%	1.91%
1-4	Solomon	0.64%	1.37%	2.45%
1-4	Cordeau	0.94%	1.78%	3.73%
All		**0.73%**	**1.49%**	**2.83%**

Table 5.6: Comparison of performance between slow and fast mode

In average, the slow mode obtains better results than the fast mode with a gap of 1.49%. An other advantage of slow mode is that it improves the worst case solutions by 2.83% in average, whereas the best case solutions are improved as well, but with gap of only 0.73%. In addition, the slow mode is better than the fast mode with a higher margin for Cordeau based instances (average gap is 1.78%) than for Solomon based instances (average gap is 1.37%). The fast

mode is slightly better only in the case of Cordeau subset of instances where number of tours is four. In that case the gap, in favor of fast mode, is 0.09%. In overall, the slow mode has the advantage of obtaining solutions with a higher quality than the fast mode, but it requires much more computation efforts.

5.4 Conclusions

In this chapter, we presented an algorithm for planning the trip itinerary for tourist groups that is built on top of the solo trip planning algorithm, which is used for construction of the starting solution. The proposed algorithm was implemented under the framework of tabu search meta heuristic with a neighborhood structure that relays on three operators, namely *Separate*, *Join* and *Insert*. The solution that is returned by the algorithm creates personalized itineraries for each tourist in the group, by allowing them to switch the company with different tourists during the trip, in accordance to their preferences.

In the experimental results that were conducted in the test set of Solomon and Coredeau based instance, based on the required computation efforts, we outlined two different modes of algorithm execution, which were tagged as slow and fast mode. The slow mode of algorithm execution proved to obtain in average better results for 1.49% than the fast mode, while being around eight times slower than fast mode. In addition, both modes of algorithm execution outperform the previous approaches, namely *Solo*, *Subgroups* and *Group*, although their computation time is in general increased. The fast mode of algorithm execution requires around 20 seconds of computation time for planning a combined trip itinerary for a group of tourist, which is four time more than the time needed by the *Group* approach, which would otherwise enable planning a joint itinerary throughout the trip.

When comparing the *Combined* mode to the *Group* mode, as the two most closest approaches in terms of quality of solutions, we can conclude that even if the *Group* mode is executed for the same amount of time as the *Combined* mode, it can not obtain better results than *Combine* mode, because the solution of *Group* mode is obtained by applying the solo trip planning algorithm, which is primarily not intended to plan the trip for tourist groups. The disadvantages in applying the solo trip planning algorithm for a tourist group are as follows:

- The satisfaction factors of all tourists are averaged for each individual points (as explained in Subsection 4.2.2 of Chapter 4). This makes the *Combined* approach unable to consider personal preferences of individual tourists.

- When tourists are kept into a single group, the personal constraints of individual tourist must be enforced. Therefore the process of planing becomes more constraint and with that the number of possible candidate solutions is decreased.

- The solo tour planning algorithm, when applied to plan a group trip itinerary, always (throughout the trip) keeps the group members together, since it has no operators for joining or separating tourists during the trip. Therefore, the solo tour planning algorithm can not consider personal preferences of individual tourists.

In general, the new proposed approach can obtain better itineraries then the previous approaches by enabling tourists to get divided during the trip, so that sometimes they travel together (or in subgroups) and visit POIs of their common interest, and at other times they travel alone so that they can visit POIs of their own specific interest.

We conclude that the slow mode of algorithm execution could be suitable for application of planning a combined trip itinerary during the pre-trip phase, where tourists could afford to wait few minutes in order to get a itinerary with a higher quality, whereas the fast mode could be applicable in the on-trip phase where tourists might select the option of planning a trip itinerary in a shorter time (in few seconds), although with a slight decrease in quality.

In reference to our first hypothesis, which was given a positive answer in Chapter 5, the approach presented in this chapter supports again a positive answer, since the solo trip planing algorithm is used as a sub algorithm for the group trip planing algorithm.

Finally, this chapter answers our third hypothesis, which hypothesized that new specialized operators in local search techniques may further improve the solutions for tourist groups. We can conclude that the third hypothesis is met, since by applying the two new operators for separating and joining tourists we were able to further improve itineraries for tourists groups.

CHAPTER 6

Scenario Evaluation

In this chapter, we present the results of the evaluation of the three cases of the scenario (described in Section 1.4 of Chapter 1) with all four modes of itinerary planning. The scenarios are executed against ten arbitrary selected POIs (belonging to five categories) in city of Prishtina. In table 6.1, the categories of the POIs are presented, whereas the detailed description and the corresponding data of POIs is given in Appendix A. In addition, with aim of clarifying the process of calculation of solution evaluation, we show the steps of calculation of the solution evaluation for Case 1 in the *Solo* and *Subgroups* mode of itinerary planning. Note that, since the scenario cases are very simple examples, individual methods manage to solve them easily by returning always the same results for different executions.

Point	**Category**
POI 1	Archeology
POI 2	Archeology
POI 3	Architecture
POI 4	Architecture
POI 5	Shopping center

Point	**Category**
POI 6	Shopping center
POI 7	Nature
POI 8	Nature
POI 9	Religious art
POI 10	Religious art

Table 6.1: Categories of ten POIs of an arbitrary tourism destination

Tables 6.2, 6.4 and 6.5 show the itinerary of the trip for Case 1 of the scenario for *Solo*, *Subgroups*, *Group* and *Combined* mode, whereas Tables 6.6 and 6.7 show the itinerary of Case 2 and Case 3 for the *Combined* mode of itinerary planning. Further, in all mantioned tables, the first column shows the tourists that are planned to pursue a particular itinerary, whereas the second column presents the respective tours and POIs planned in the itinerary. The next three columns show the wait time, start time, end time of a visit to a POI, whereas the second last column presents the amount of time that is left as unused at the end of a trip. Finally, the last column shows the cost of an itinerary for each respective tourist.

6.1 Case 1 - A group of three friends

Tourist	Tour/POI	Wait time	Start time	End time	Left time	Cost
Woman 1 Woman 2	Tour 1		9:00	11:30	0	33
	POI 3	0	9:18	9:43		
	POI 7	0	9:58	10:47		
	Tour 2		9:00	10:55	35	
	POI 4	0	9:25	9:45		
	POI 8	0	10:04	10:23		
Man	Tour 1		9:00	11:30	0	32
	POI 3	0	9:18	9:43		
	POI 7	0	9:58	10:47		
	Tour 2		9:00	11:11	19	
	POI 10	0	9:35	9:55		
	POI 8	0	10:19	10:39		

Table 6.2: Trip itinerary for tourists of Case 1 in *Solo* and *Subgroups* mode

As it can be seen in Table 6.2, the itinerary of Case 1 for *Solo* and *Subgroups* modes is the same, where the women are scheduled to visit *POI 3* and *POI 7* in the first tour and *POI 4* and *POI 8* in the second tour, while the man is also scheduled to visit *POI 3* and *POI 7* in the first tour, but in the second tour he is initially scheduled to visit *POI 10* and then *POI 8*. Even though the *Solo* mode plans the trip separately for each tourist, the resulting itineraries are totally the same for both woman, whereas the man is also scheduled with same itinerary as the woman in the first tour, while in the second tour he is scheduled to follow a different path. On the other hand, the *Subgroups* mode puts the two woman in a subgroup, while keeping the man separately. However, also the *Subgroups* mode produces the same itinerary as the *Solo* mode. The similarity in the itineraries of three tourists (especially in the case of woman) can be explained due to their similar preferences about POIs categories of nature and architecture. Nevertheless, the man in contrast to the woman is scheduled to visit *POI 10* in the second tour, as he is also interested in visiting POIs that belong to category of religious art and due to his specification that he does not want to visit more than one POI that belongs to category of architecture (see Table A.11 in Appendix A). It can be noticed that in the first tour, non of the tourists has any left (unused) time at the end of the tour, while in the second tour the woman have 35 minutes left, whereas the man has only 19 minutes left.

The details for calculation of the overall satisfaction of all tourists with the itineraries of *Solo* and *Subgroups* mode are given in Table 6.3. The first column shows all POIs that are in the itinerary, whereas the next four columns present the social relationship values (see Table A.9 in Appendix A) between the tourists that visit a certain POI together. The last two columns show the satisfaction of tourists when visiting a certain POI (satisfaction from POIs) in company of certain group members (satisfaction from company of other tourists).

The calculation of the overall tourists satisfaction (evaluation of solution) with the proposed itinerary is done by using the objective function (Equation 4.1) that is defined in Section 4.1 of

		Social relationship				
POI	**Tourist**	Woman 1	Woman 2	Man	**Satisfaction with POIs**	**Satisfaction with company**
POI 3	Woman 1		4	5	40	9
	Woman 2	5		5	40	10
	Man	5	4		50	9
POI 4	Woman 1		4		45	4
	Woman 2	5			45	5
POI 7	Woman 1		4	5	40	9
	Woman 2	5		5	40	10
	Man	5	4		40	9
POI 8	Woman 1		4		40	4
	Woman 2	5			40	5
	Man				40	0
POI 10	Man				45	0
Subotal		***191***	***195***	***193***	***505***	***74***
					Total	**579**

Table 6.3: Calculation of solution evaluation for Case 1 in *Combined* mode

Chapter 4. The satisfaction of a single tourist consists of two parts, which is the satisfaction for visiting POIs and the satisfaction for being accompanied by the other tourists. Each time a tourist visits a certain POI with some particular group members, the score of that POI for that tourist and the sum of social relationship values of the tourist with the particular group members are added into her/his overall satisfaction. Similarly, the process is repeated for all tourists, where the sum of their satisfaction constitutes the overall group satisfaction with the tour. In following, we present three functions, namely S_{woman1}, S_{woman2} and S_{man} that reflect the overall satisfaction of woman and man, respectively. Further, the S_{total} function reflects the overall satisfaction of all tourists.

S_{woman1} = [40+ (4+5)] +[45+(4)]+[40+(4+5)]+[40+(4)] = 191

S_{woman2}= *[40+ (5+5)] +[45+(5)]+[40+(5+5)]+[40+(5)] = 195*

S_{man}= *[50+ (5+4)] +[40+(5+4)]+[40+(0)]+[45+(0)] = 193*

S_{total} = *191 + 195 +193 = 579*

Note that the procedure for calculation of the overall evaluation of the itinerary is the same also for the cases of the *Group* and the *Combined* mode of trip planing, therefore, we do not present it in such details in the remaining part of this section.

Further, the *Group* mode makes a joint itinerary for all tourists (see Table 6.4). In this case, based on the procedure for merging tourists' data (see Subsection 4.2.2 of Chapter 4), the score

of a POI is calculated as the average value of the score of all tourists with that POI. In addition, in order to meet the constraints set by each tourist regarding the budget limit and the maximum number of POIs, the minimal values of the data of the respective tourists are taken. Considering such constraints, the *Group* mode of trip planning prepares the itinerary that contains *POI 3* and *POI 7* in the first tour and *POI 10* and *POI 8* in the second tour. The negative side of the itinerary returned by *Group* mode is evident in the case of woman, who although they are not interested in visiting POIs of category of religious art, they are still scheduled to visit *POI 10*, due to the presence of man in the group, who has expressed interest in visiting POIs of such category. The overall satisfaction of all tourists for the *Group* mode is 527, while the left time for the first tour is zero and for the second tour it is 19 minutes.

Tourist	Tour/POI	Wait time	Start time	End time	Left time	Cost
Woman 1 Woman 2 Man	Tour 1		9:00	11:30	0	32
	POI 3	0	9:18	9:43		
	POI 7	0	9:58	10:47		
	Tour 2		9:00	11:11	19	
	POI 10	0	9:35	9:55		
	POI 8	0	10:19	10:39		

Table 6.4: Trip itinerary for tourists of Case 1 in *Group* mode

Finally, Table 6.5 presents the itinerary that is returned by the *Combined* mode of algorithm execution. It can be noticed that the returned itinerary is a combination of previous itineraries returned by *Solo* and *Subgroups* mode on one side, and the itinerary returned by *Group* mode on the other side. The woman visit their most preferred POIs (*3, 7, 4* and *8*), whereas also the man visits his most preferred POIs (*3, 7, 10* and *8*). In this way, the woman are not scheduled to visit *POI 10* (which is out of their interest) as in the case of *Group* mode, whereas the man is not scheduled to travel alone in the second tour as in the cases of *Solo* and *Subgroups* mode. The *Combined* approach takes advantage of the left time at the end of the second tour, by scheduling the woman to wait (15 minutes) for the man when they arrive at *POI 8*, so that they can make the visit together, hence their overall satisfaction will be increased. In this way, the *Combined* mode of itinerary planning increases the overall satisfaction of three tourists by assuming that none of them finds it inconvenient to wait for the other group members in order to make a joint visit. The overall satisfaction of all tourists with itinerary is 598, while the left time for both tours is 19 minutes.

Note that Case 2 and Case 3 of the scenario are also executed against all proposed modes of itinerary planning. Nevertheless, in the following we present only the itinerary that is returned by the *Combined* mode, since, in both cases, it returns the best evaluating itinerary amongst the others.

Tourist	Tour/POI	Wait time	Start time	End time	Left time	Cost
Woman 1 Woman 2	Tour 1		9:00	11:30	0	33
	POI 3	0	9:18	9:43		
	POI 7	0	9:58	10:47		
	Tour 2		9:00	11:11	19	
	POI 4	0	9:25	9:45		
	POI 8	15	10:19	10:39		
Man	Tour 1		9:00	11:30	0	32
	POI 3	0	9:18	9:43		
	POI 7	0	9:58	10:47		
	Tour 2		9:00	11:11	19	
	POI 10	0	9:35	9:55		
	POI 8	0	10:19	10:39		

Table 6.5: Trip itinerary for tourists of Case 1 in *Combined* mode

6.2 Case 2 - A group of two couples

Table 6.6 shows that the two couples are scheduled with a combined (mixed) itinerary. They all start the trip together by visiting *POI 1*, and then they get separated, since the women are scheduled to go to *POI 5* and *POI 6* (both shopping centers), while the men shall join them at *POI 6* after making a visit to *POI 4*. In addition, the women are assigned to wait (8 minutes) for their men before starting the visit at *POI 6*. The overall evaluation for the *Combined* mode is $S_{Combined}$=*622*, whilst the other modes are evaluated as in following: S_{Solo}=*590*, $S_{Subgroups}$=*590* and S_{Group}=*594*.

Tourist	Tour/POI	Wait time	Start time	End time	Left time	Cost
Man 1 Man 2	Tour 1		10:00	12:47	13	24
	POI 1	0	10:10	10:40		
	POI 4	0	10:52	11:22		
	POI 6	8	11:43	12:23		
Woman 1 Woman 2	Tour 1		10:00	12:47	13	30
	POI 1	0	10:10	10:40		
	POI 5	0	11:00	11:30		
	POI 6	0	11:43	12:23		

Table 6.6: Trip itinerary for tourists of Case 2 in *Combined* mode

6.3 Case 3 - A group of seven student of arts

Table 6.7 shows that the students of the Case 3 of the scenario are scheduled to be divided into two subgroups, where the first subgroup consists of the first four students, whereas the other

subgroup has the remaining students. Both subgroups start the trip separately, where the first one visits in sequence *POI 10* and *POI 8*, while the second subgroup visits consecutively *POI 1* and *POI 2*. Afterwards, before the end of the trip, the second subgroup waits (5 minutes) for the first subgroup so that they can get together before starting the visit to *POI 9*. The overall satisfaction of tourists of Case 3 for the *Combined* mode is $S_{Combined}$=*1191*, whereas for the other modes, the results areas follows: S_{Solo}=*1138*, $S_{Subgroups}$=*1023* and S_{Group}=*1147*.

Tourist	Tour/POI	Wait time	Start time	End time	Left time	Cost
Student 1	Tour 1		8:00	11:23	37	
Student 2	POI 10	0	8:35	9:15		
Student 3	POI 8	0	9:39	9:59		19
Student 4	POI 9	0	10:08	10:38		
Student 5	Tour 1		8:00	11:23	37	
Student 6	POI 1	0	8:10	8:40		
Student 7	POI 2	0	8:55	9:45		21
	POI 9	5	10:08	10:38		

Table 6.7: Trip itinerary for tourists of Case 3 in *Combined* mode

6.4 Summary

In Table 6.8, we present a summary of the results for the three cases. It is evident that in all three cases of the scenario, the results of the *Combined* mode are better then the results of any other mode. Although, the selected cases are primarily introduced for easing to the reader of this thesis the understanding of the goal of the proposed methods, the obtained results are in compliance with the results obtained from the computational experiments presented in Section 5.3 of Chapter 5, where it was indicated that in general it could be possible to improve the itinerary by applying the *Combined* mode of trip planning. Further, the results of the cases of the scenario indicate that there might often exist situations in real life where tourist could have their itinerary prepared into a combined (mixed) form, where at some POI they could be together, and then at some other POI they could get separated or they could change the group.

Case	Solo	Subgroups	Group	Combined
Case 1	579	579	527	598
Case 2	590	590	594	622
Case 3	1138	1023	1147	1191

Table 6.8: Summary of results for the three cases

CHAPTER 7

Conclusion

There is a trend to ever more sophisticated recommendation services in the domain of electronic tourism. The traditionally supported services, such as recommendation of accommodations, restaurants, organized tours, etc., are now being complemented with more specialized services such as for example the case of recommendation of trip itineraries for tourists. In addition, group trips are already, either directly or indirectly, supported by various online communities, which take advantage of user relations to enable them to share tourism information that might be of mutual relevance. In this regard, the research work in this PhD thesis was motivated by the idea of by making it possible to plan the trip itinerary for a group of tourists, where individual preferences of tourists and their mutual social relationship is considered. The goal was to plan a trip itinerary that maximizes the satisfaction of all groups members, both in terms of scheduling POIs that meet their individual preferences and in arranging the visits in groups (or subgroups) of tourists, who are in close social relationship. In order to accomplish this task, we took an evolutionary approach of research and development by first developing an algorithm that can be used to plan the trip itinerary for a single tourist, and then, we used it as a starting point for introducing three different approaches for planning the trip itinerary for a group of tourists.

This chapter aims to answer the research question (Section 7.1) that was initially formulated within this thesis. In order to do this, the main contributions that are achieved in this thesis are revisited and discussed in the context of their contribution in answering the posed research question. Finally, we present our views for future research work (Section 7.2) in the envisioned field.

7.1 Answer to Research Question

The work in this thesis was guided by the following research question:

> *Can we improve trip planning systems for tourist groups by considering individual preferences and social relationship?*

We answer the research question by developing and comparing four approaches for planning the trip itinerary for a group of tourists. These approaches relay on three algorithms (artifacts [62]) that are built based on a predefined sequence. The first algorithm (solo trip planning) is developed with a twofold goal, where the first is to create an algorithm that is comparable with existing approaches in the literature, and the second is to use it as a building block for developing the two later algorithms. The second algorithm (tourist subgrouping) is aimed at finding out how much could trip planning systems (for tourists groups) be improvement if tourists are grouped into smaller subgroups based on their individual preferences and social relationship. Finally, the third algorithm (group trip planning) is aimed to investigate further improvement of trip planning systems (for tourist groups) by enabling tourists to change their group (company) while they are on the trip, so that a more personalized trip itinerary is planned by accounting individual preferences and social relationship.

In Chapter 3, we presented the solo trip planning algorithm that was developed based on tabu search metaheuristic. The algorithm enables planning the trip itinerary by considering multiple period trips, time windows of POIs and maximum number of POIs of certain category (e.g. archeology, architecture, nature, etc.). In the literature, such a problem is known with the name Multi Constraint Team Orienteering Problem with Time Windows (MCTOPTW) [43]. The algorithm can be attributed with two main components, namely the *neighborhood exploration* and *search diversification*. The *neighborhood exploration* component explores the neighbors of a given solution based on three operators that includes *Insert* operator (for inserting new non-included points), *Swap* operator (for swapping two included points) and *Replace* operator (for replacing two included points with two non-included points). Meanwhile, the *search diversification* component employs a diversification mechanism based on four operators, namely *Delete* (that removes two points from solution), *Perturbate* (that restarts the process from best solution), *Restart* (that restarts the search process from a random initial solution) and *Penalize* (that penalizes frequently used moves).

Based on the experimental computations on the instances from literature, we can conclude that our proposed algorithm obtains good results, although in average they are worse than the results obtained by the state of the art approach of Souffriau et al. [119] for about 4%. Nevertheless, when comparing the best returned solutions, our approach finds new better solutions for 70 (out of 148) instances in comparison to the approach of Souffriau et al. [119] for the MCTOPTW problem. In terms of computation time, we can not draw a conclusion in comparing our approach with the approach of Souffriau et al. [119], since we conduct the computation experiments by using machines with different processing capabilities. However, our approach gives good solutions within few seconds of computation time.

Despite its good performance for solo trip planning, this algorithm can not be appropriately applied in the case of tourists groups, because it does not take into account the preferences of multiple tourists and the social relationship between them. For instance, in Case 1 of the scenario, when the algorithm was applied in the *Solo* and *Subgroups* approaches, the women and man were scheduled to travel separately in their second tour, although they visit the same place (*POI 8*). Further, also in Case 1 of the scenario, when the algorithm was applied in *Group* approach, the women and the man were scheduled to visit *POI 10* (religious art), since the man is interested in such POI categories, although the women are not interested in that POI category.

However, based on the performance of the algorithm (in the test set existing in the literature), both in terms of quality of solution and computation time, we can conclude that this algorithm can be used to create a good starting solution when planning the itinerary for tourists groups, where all tourists would either be initially scheduled to travel alone or together.

In order to theoretical model the group trip planning problem, we defined a new problem that we called Multi Constraint Multiple Team Orienteering Problem with Time Windows (MCMTOPTW), which has two extensions in comparison to the existing MCTOPTW problem [43]. The first extension is about modeling preferences and constraints of multiple tourists, whereas the second one is about modeling interpersonal social relations of the tourists in the group. Further, we generated a new test set based on the existing tests for the MCTOPTW problem [119].

In Chapter 4, we presented a method for clustering tourists into subgroups based on their preferences about POIs and on their interpersonal relationships. The subgrouping method is based on *k-means* algorithm and clusters tourists into subgroups by measuring the distance between any two of them using a function based on euclidean distance. The measurement of the distance between any two tourists includes their: budget limit, interests about all POIs, limits about the maximum number of POIs of different categories they want to visit, mutual social relationship and social relationship with other tourists. In order to keep the number of groups (value of k) dynamic, we implemented an iterative procedure that executes the *k-means* algorithm for a number of iterations, where the best evaluating subgrouping solution is returned. The evaluation of a subgrouping solution is done by using the method of Caliński and Harabasz [18]. The subgrouping method is used to identify tourist subgroups, for which a separate itinerary is planned by applying the solo trip planning algorithm (the *Subgroups* approach).

Based on computational experiments on the instances of new MCMTOPTW problem and by using the above presented distance function between tourists, despite our second hypothesis, in overall, we can conclude that there is no indication that the process of clustering tourists into subgroups yields to better trip itineraries for tourist groups. Nevertheless, the experimental results indicate that *Subgroups* approach performs slightly better then *Solo* approach for test instances with three and four number of tours.

A drawback that it obvious when planning the itinerary in the *Subgroups* and *Group* approach is related to the constraints (e.g. budget limit of 50 Euro, maximum 2 POIs of category religious art, etc.) that individual tourists in subgroup/group impose, since when they are in subgroup/group, the constraints of each of the members need to be respected, and as result the process of planning the itinerary for the whole subgroup/group is much more constrained. In addition, the *Subgroups* approach partially inherits non favorable features of *Solo* approach in reference to the component of social relationship, since the relations between tourists that belong to different subgroups are always omitted, although such relations sometimes might be week. On the other hand, the *Group* approach always considers the social relationship between all tourists, since they are always together. Further, it remains an open question to investigate utilization of other distance functions (between tourists) that could hypothetically improve the subgrouping algorithm and with that increase its impact in planning better itineraries for tourist groups.

In Chapter 5, we presented a group trip planning algorithm that combines the itineraries of different group members with the goal of enabling them to change the group during the trip so

that they can visit most preferred PIOs with the closest related group members. Therefore, the algorithm can consider trips with multiple tourists and the social relationship between them. The algorithm is based on the general framework of tabu search meta heuristic and it is implemented on top of the solo trip planning algorithm, since its initial solution is created by employing solo trip planning algorithm. The neighborhood structure of the algorithm consists of three operators, namely *Separate* (two tourists at a given POI), *Join* (two tourists at a given POI) and *Insert* (new POIs in vacant time slots of tours). In addition, at some iterations (as determined by *re-initialization frequency - RIF* parameter) the algorithm employees a diversification mechanism by restarting the search process from a new initial solution.

The experimental results that were conducted on the instances of the new MCMTOPTW enabled two identify two modes of group algorithm execution (the *Combined* approach), which differentiate in terms of computation time and quality of solutions they produce. The slow mode requires about 150 seconds per execution, while the fast mode needs about 20 seconds. The quality of solutions obtained by the slow mode is in average for 1.5% higher then in the case of fast mode.

In general, it can be concluded that the *Combined* approach (in both modes) of itinerary planning obtains better itineraries then all three previous approaches. The difference in terms of quality of solutions of the *Combined* approach (in its slow mode) from the *Solo*, the *Subgroups* and the *Group* approaches is 20.68%, 17.08% and 5.47%, respectively. Thus, the *Combined* approach is able to make a combined itinerary for tourists by enabling them to change their groups during the trip based on their preferences about POIs and the social relationship. For instance, if Case 1 of the scenario is considered again, it can be noticed that the *Combined* approach meets the preferences of the women by scheduling them to visit *POI 4* (architecture) instead of visiting *POI 10* (religious art) as it was previously proposed by *Group* approach. Further, also in Case 1, the *Solo* and *Subgroups* approaches scheduled the man to travel separately from the women for the whole duration of the second tour, whereas the *Combined* approach schedules them separately only when visiting POIs of their specific interest (*POI 4* for the women and *POI 10* for the man).

As result, given the obtained computational results, we can finally answer our research question with a "*yes*", the trip planning systems for tourist groups obtain improved itineraries when they consider separating and joining tourists for some part of the trip based on individual preferences and social relationship.

7.2 Future Work

In this section, we discuss some opportunities for future research work that are divided into four different paths that include test set extension, algorithmic perspective, feature extensions and possible applications in other domains.

7.2.1 Test Set Extension

In reference to the generated test set (for the new MCMTOPTW problem), which is used to evaluate the group trip planning algorithm, in Subsection 4.3.1 of Chapter 4, we presented the

details about modelling the test data for multiple tourists and the social relationship between them. The data about preferences (score for POIs) and constraints (budget limit, and limits for maximum number of POIs of certain category) of additional tourists very generated randomly by following the same patterns as the existing data in the test instances. In addition, the data about social relationship between tourists were also generated randomly in a domain range from 0 to 5 by assuming that the relationship between different tourists in the group has a random distribution structure. Nevertheless, based on claims by many authors [32] [140] [88] in the literature that the social relations between people in different networks, tend to have a power of low distribution (scale free networks, where some persons have stronger relationship than the others) [7], it might be worth to investigate creating test data that model social relationship between tourists based on a power of law distribution rather than in a random way.

Another extension to the problem could be enforcing a constraint that defines a minimum number of tourists that are at any moment in the group. This constraint is typical in practice, as often people do not want to remain alone or in a group of very few members.

7.2.2 Algorithmic Perspective

As described in Chapter 2, the simplest theoretical modelling for the solo trip planning problem is the Orienteering Problem, which was proved by Golden et al. [57] to belong to the class of NP-hard problems. Further, the other extensions of OP such as TOP, TOPTW, MCTOPTW are at least as hard as OP. In addition, our extension of MCTOPTW problem to MCMTOPTW in order to consider multiple tourists produces even larger search space.

As results, within the group trip planning algorithm we applied two new operators, where one of them separates two tourist at a given POI, while the other one joins two of them at given POI. In principal, the two new operators consider separating/joining any pair of tourists at any given point in the itinerary, provided that they satisfy the constraints of the problem. Therefore, one could assume that these two operators are atomic enough so that they can theoretically reach any possible combination in the search space.

Nevertheless, investigating neighbourhood mechanisms that apply new operators for solving the problem at hand might lead to improvements in efficiency or/and effectiveness of exploration of search space, given that the problem is also highly constraint. For instance, a new possible operator for investigation in future could be swapping one or more tourists between two subgroups that visit two different closely located POIs. Another more general operator could rotate three or more tourists between their own subgroups so that each tourist goes in the subgroup of another tourist based on some order that might be generated randomly or by using a specific heuristic function.

Besides trying out new operators, it would also be interesting to apply other meta heuristic techniques such as for instance memetic algorithms, or hybridization of meta heuristic techniques with other techniques from constraint programming and operations research.

Gavalas et al. [45] identify application of parallel algorithms for computation of trip itinerary as one of the future promising paths of research in the envisioned field. The search process of the metaheuristics could be divided into separate parts so that they could be executed in different computation threads. For instance, a possible separation of search process could include parallel execution of different operators when exploring the neighbourhood of a given

current solution. This might enable more efficient utilization of large neighbourhood search operators [114], which in a sequential oriented implementation might be very computation intensive.

7.2.3 Feature Extensions and Applications in Other Domains

Besides the inclusion of features for automatic selection of the accommodation (hotel) at the end of the trip [30], it would be an added value for the tourist if the trip itinerary could also include stays at restaurants so that tourists can eat and rest after visiting a couple of POIs. The process of selection should be based on the preferences of tourists (e.g. time of eating, budget, cousin type, indoor or outdoor, fast food, etc.) and on the geographical position of the restaurants.

In the approaches presented within this thesis and on the related approaches from the literature, the visit duration to a certain POI is considered to be the same (static) for any tourist. In practice, it might happen that individual tourists (in accordance to their preferences) might prefer to stay longer or shorter to a given POI. Therefor it would be interesting to investigate new methods that determine the length of a visit (variable visit duration) to a POI based on tourist preferences.

Many tourists might prefer walking through a "scenic route" [63] (e.g. down town of a city) that has a number of POIs alongside. Nevertheless, consideration of scenic routes makes the tourists trip planning problem further more complex, since the score of such route must be taken into account.

A feature of added value for tourist trip planning systems could also be consideration accessibility features to POIs, such as for example the POIs that are physically located at places (e.g. hilly area, staircases, etc.) that are not reachable by people with disabilities.

During the on-trip phase, the itinerary might become infeasible due to different situations that might arise, such as for instance closure of certain POI, transportation difficulties, whether change, etc. Therefore investigating methods that automatically detect the infeasibility in the itinerary might be an important path of future research in the field of tourist trip planning. In addition, during the on-trip phase, it might also happen that new opportunities (POIs) might become available for tourists (e.g. a football match is played in the city where tourist are doing their trip). In such cases, it would be interesting to have a system that would automatically react in updating the trip itinerary by including a visit to a such POI.

A possible application of the approach for group trip planning could be the problem of assigning staff members to projects, where different staff members are engaged in different projects during a working day. In that case, each project would have a score for each staff member, whereas also the staff members have their preferences in reference to the co-workers they want to work with. Therefore, the objective would be to find a plan for assigning staff members to projects, so that they are assigned to the projects where they fit the most and the groups consists of staff members that are highly related. In between the day, the staff members could change their group. Although our approach covers the concept of assigning staff members based on their score to projects and social relations between staff members, it is a question of further investigation to determine a correlation between the constrains of this new problem and the constraints that are supported by our approach (e.g. maximum budget, maximum POIs of certain category, etc.).

List of Figures

List of Tables

APPENDIX A

Scenario data

We suppose a geographic location that contains *N=10* POIs, which are characterized with attributes as presented in Table A.1. Further, there are five different types and categories of POIs that are utilized in the scenario (see Table A.2 and A.3). In addition, the same starting and ending point is used for all three examples of the scenario.

A single case of the scenario is characterized with basic trip details and data of tourists, as described in tables A.4 and A.5, respectively. Table A.6 presents the actual values used for the basic trip details of the three cases in the scenario.

The data about the points and the tourists of the corresponding case could be found in the respective tables. The data about points and the distances between points (including starting/ending point) are presented in tables A.7 and A.8. The data about tourists of a single case are divided into three separate tables and are organized in parts as follows: tourists budget and tourists' relationship, satisfaction factors with POIs, and maximum allowed point categories. Tables from A.9 up to A.17 show the data of tourists for all three envisioned cases of the scenario.

Symbol/Abbreviation	**Description**
b_i	Entrée fee
O_i	Open time
C_i	Closing time
PC_{ij} *(i=1, ... N; j=1, ..., 5)*	Contains a logical value that indicates whether point *i* belongs to category *j*. Not that a single point might belong to different categories.
d_{ij} *(i=1, ..., N; j=1, ..., N,* $i \neq j$*)*	Distance (in unit of minutes) between point *i* and *j*

Table A.1: Details of POIs

#	Type name
1	Castle
2	Park
3	Mosque
4	Church
5	Miscellaneous

Table A.2: List of types of POIs

#	Category name
1	Archeology
2	Architecture
3	Religious art
4	Nature
5	Shopping center

Table A.3: List of categories of POIs

Symbol/Abbreviation	Description
P	Number of tourists
M	Number of tours (days) for the trip
ST	Start time of the tour
ET	End time of the tour

Table A.4: General trip details

Symbol/Abbreviation	Description
B	Budget limitation
S_{ij} $(i=1,..., P; j=1, \ldots, N)$	Satisfaction factor of tourist i with point j
SR_{ij} $(i=1, \ldots, P; j=1, \ldots, P, i \neq j)$	Social relationship level of tourist i with tourist j
MC_{ij} $(i=1,..., P; j=1, \ldots, N)$	Maximum allowed number of points of category j that are allowed by tourist i

Table A.5: Tourist details

Example	P	M	ST	ET
Case 1	3	2	09:00	11:30
Case 2	4	1	10:00	13:00
Case 3	7	2	08:00	12:00

Table A.6: List of examples

				Category				
Point	**Entry fee**	**Open time**	**Close time**	***PC1***	***PC2***	***PC3***	***PC4***	***PC5***
P1	5	0	90	1	0	0	0	0
P2	10	0	145	1	0	0	0	0
P3	7	30	120	0	1	0	0	0
P4	9	0	145	0	1	0	0	0
P5	15	0	120	0	0	1	0	0
P6	10	0	145	0	0	1	0	0
P7	12	0	120	0	0	0	1	0
P8	5	0	130	0	0	0	1	0
P9	6	40	200	0	0	0	0	1
P10	8	0	80	0	0	0	0	1

Table A.7: Data of points

		Distance									
Point	**Start/end point**	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
P1	10	-	15	18	25	30	25	42	32	45	35
P2	30	15	-	8	12	20	24	18	30	24	26
P3	18	18	8	-	16	23	10	28	30	18	28
P4	30	25	12	16	-	30	19	15	35	15	44
P5	25	30	20	23	14	-	13	28	19	15	14
P6	12	25	24	10	10	13	-	33	16	26	12
P7	16	42	18	28	27	38	33	-	17	5	45
P8	22	32	30	30	25	19	16	17	-	9	20
P9	10	34	24	18	27	15	13	5	9	-	24
P10	30	18	26	28	22	14	12	45	20	24	-

Table A.8: Distances between points

Tourist	**Budget**	**SR1**	**SR2**	**SR3**
T1	50	-	4	5
T2	40	5	-	5
T3	45	5	4	-

Table A.9: Case 1: Budget and tourists' relationship

Tourist	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**
T1	0	0	40	45	0	0	40	40	0	0
T2	0	0	40	45	0	0	40	40	0	0
T3	0	0	50	50	0	0	40	40	20	45

Table A.10: Case 1: Satisfaction factors with POIs

Tourist	MC1	MC2	MC3	MC4	MC5
T1	3	2	3	2	1
T2	3	2	3	2	1
T3	3	1	0	3	2

Table A.11: Case 1: Maximum allowed point categories

Tourist	Budget	SR1	SR2	SR3	SR4
T1	50	-	5	4	3
T2	100	5	-	3	4
T3	50	4	3	-	5
T4	100	3	4	5	-

Table A.12: Case 2: Budget and tourists' relationship

Tourist	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	50	50	50	40	10	20	10	5	5	10
T2	50	50	0	10	45	50	10	5	10	10
T3	50	50	50	40	10	20	5	5	5	10
T4	50	50	10	10	45	50	5	10	10	10

Table A.13: Case 2: Satisfaction factors with POIs

Tourist	MC1	MC2	MC3	MC4	MC5
T1	3	2	3	2	3
T2	3	2	3	3	4
T3	3	4	3	2	2
T4	3	2	3	3	2

Table A.14: Case 2: Maximum allowed point categories

Tourist	Budget	SR1	SR2	SR3	SR4	SR5	SR6	SR7
T1	70	-	2	2	3	1	3	1
T2	60	3	-	1	2	1	4	2
T3	80	1	1	-	2	1	1	2
T4	90	2	1	2	-	1	3	2
T5	70	2	3	1	1	-	5	1
T6	70	2	3	4	3	3	-	2
T7	75	3	2	4	3	1	2	-

Table A.15: Case 3: Budget and tourists' relationship

Tourist	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	0	0	0	0	40	10	10	45	45	50
T2	0	0	0	0	20	5	5	45	45	50
T3	0	0	0	0	30	10	10	45	45	50
T4	0	0	0	0	20	5	5	40	45	50
T5	50	50	0	0	30	5	5	0	0	50
T6	50	50	0	0	20	5	5	0	0	50
T7	50	50	0	0	30	3	5	0	0	50

Table A.16: Case 3: Satisfaction factors with POIs

Tourist	MC1	MC2	MC3	MC4	MC5
T1	3	3	4	4	3
T2	4	3	5	4	4
T3	4	3	5	4	4
T4	3	3	5	3	4
T5	3	3	5	3	4
T6	3	4	5	3	4
T7	3	4	5	3	4

Table A.17: Case 3: Maximum allowed point categories

APPENDIX B

Results for MCTOPTW problem

Instance	ILS-GRASP	Best	Average	Worst	Average time (ms)
MCTOPMTW-1-c101.txt	320	260	240.00	220	1402.50
MCTOPMTW-1-c102.txt	360	320	316.67	250	1375.87
MCTOPMTW-1-c103.txt	400	390	377.33	370	1900.17
MCTOPMTW-1-c104.txt	420	400	400.00	400	1987.70
MCTOPMTW-1-c105.txt	340	340	334.00	220	1739.00
MCTOPMTW-1-c106.txt	340	340	337.67	300	2214.27
MCTOPMTW-1-c107.txt	370	350	349.33	330	2593.03
MCTOPMTW-1-c108.txt	370	370	370.00	370	1646.23
MCTOPMTW-1-c109.txt	380	380	380.00	380	2411.13
MCTOPMTW-1-r101.txt	198	170	122.50	71	523.07
MCTOPMTW-1-r102.txt	286	275	275.00	275	1456.47
MCTOPMTW-1-r103.txt	293	282	278.33	275	1576.13
MCTOPMTW-1-r104.txt	303	295	290.07	269	1369.90
MCTOPMTW-1-r105.txt	247	226	200.80	152	2155.73
MCTOPMTW-1-r106.txt	293	293	283.67	277	1578.20
MCTOPMTW-1-r107.txt	299	293	285.63	271	1911.67
MCTOPMTW-1-r108.txt	286	295	292.27	286	1981.33
MCTOPMTW-1-r109.txt	277	277	275.37	264	1872.47
MCTOPMTW-1-r110.txt	284	284	282.57	271	1606.00
MCTOPMTW-1-r111.txt	297	285	282.63	281	1861.67
MCTOPMTW-1-r112.txt	298	298	286.30	285	1453.53
MCTOPMTW-1-rc101.txt	219	203	191.20	163	1857.80
MCTOPMTW-1-rc102.txt	266	239	215.67	157	1811.37
MCTOPMTW-1-rc103.txt	266	252	250.30	242	1751.13
MCTOPMTW-1-rc104.txt	301	301	299.77	264	1286.20
MCTOPMTW-1-rc105.txt	244	233	217.03	183	2013.53
MCTOPMTW-1-rc106.txt	252	252	250.70	231	1534.30
MCTOPMTW-1-rc107.txt	277	267	264.93	254	1493.47
MCTOPMTW-1-rc108.txt	298	278	278.00	278	1974.13
MCTOPMTW-1-pr01.txt	308	301	271.40	253	1358.57
MCTOPMTW-1-pr02.txt	379	377	363.03	335	2875.57
MCTOPMTW-1-pr03.txt	388	381	349.97	318	5040.60
MCTOPMTW-1-pr04.txt	475	443	402.73	353	7617.83
MCTOPMTW-1-pr05.txt	526	535	464.27	392	10433.83

Instance	ILS-GRASP	Best	Average	Worst	Average time (ms)
MCTOPMTW-1-pr07.txt	298	298	272.53	188	2220.97
MCTOPMTW-1-pr08.txt	463	435	406.10	357	4907.57
MCTOPMTW-1-pr09.txt	463	453	398.47	358	9453.97
MCTOPMTW-2-c101.txt	580	520	460.33	420	3073.10
MCTOPMTW-2-c102.txt	640	630	613.00	590	3487.53
MCTOPMTW-2-c103.txt	670	710	692.67	670	3774.13
MCTOPMTW-2-c104.txt	710	750	736.33	720	3969.60
MCTOPMTW-2-c105.txt	640	620	616.67	600	3794.30
MCTOPMTW-2-c106.txt	620	600	594.67	580	3475.03
MCTOPMTW-2-c107.txt	660	670	668.00	640	2872.57
MCTOPMTW-2-c108.txt	640	680	674.33	670	3492.33
MCTOPMTW-2-c109.txt	680	710	708.67	690	3814.30
MCTOPMTW-2-r101.txt	330	262	235.07	210	1758.67
MCTOPMTW-2-r102.txt	508	486	455.73	423	3474.10
MCTOPMTW-2-r103.txt	513	513	490.30	463	3581.03
MCTOPMTW-2-r104.txt	503	539	519.57	497	3798.83
MCTOPMTW-2-r105.txt	430	393	351.43	330	2743.47
MCTOPMTW-2-r106.txt	529	529	493.50	447	3690.33
MCTOPMTW-2-r107.txt	517	529	508.73	479	3287.40
MCTOPMTW-2-r108.txt	525	550	530.73	507	3703.40
MCTOPMTW-2-r109.txt	466	493	475.17	442	3788.67
MCTOPMTW-2-r110.txt	502	510	489.93	459	3167.50
MCTOPMTW-2-r111.txt	517	529	511.97	492	4043.93
MCTOPMTW-2-r112.txt	515	506	494.80	479	3735.70
MCTOPMTW-2-rc101.txt	427	382	339.03	301	2878.40
MCTOPMTW-2-rc102.txt	495	466	447.80	416	3245.50
MCTOPMTW-2-rc103.txt	509	513	492.20	463	3254.37
MCTOPMTW-2-rc104.txt	565	564	540.43	509	3969.27
MCTOPMTW-2-rc105.txt	440	434	403.53	348	2820.27
MCTOPMTW-2-rc106.txt	455	442	419.73	363	2743.40
MCTOPMTW-2-rc107.txt	515	505	485.43	467	2400.00
MCTOPMTW-2-rc108.txt	546	526	509.87	414	3159.87
MCTOPMTW-2-pr01.txt	471	480	435.07	413	2019.97
MCTOPMTW-2-pr02.txt	611	652	622.53	573	5590.17
MCTOPMTW-2-pr03.txt	695	689	624.50	569	9745.37
MCTOPMTW-2-pr04.txt	795	804	727.03	635	15006.67
MCTOPMTW-2-pr05.txt	923	937	816.10	689	20262.50
MCTOPMTW-2-pr07.txt	544	546	499.83	469	3761.57
MCTOPMTW-2-pr08.txt	727	740	694.03	611	9226.60
MCTOPMTW-2-pr09.txt	755	765	654.33	580	15804.50

Instance	ILS-GRASP	Best	Average	Worst	Average time (ms)
MCTOPMTW-3-c101.txt	770	660	628.33	590	4387.13
MCTOPMTW-3-c102.txt	860	840	808.33	770	4844.73
MCTOPMTW-3-c103.txt	910	960	933.67	900	5934.47
MCTOPMTW-3-c104.txt	940	1010	981.33	960	4723.87
MCTOPMTW-3-c105.txt	810	840	816.00	780	4522.53
MCTOPMTW-3-c106.txt	820	840	814.67	780	5000.93
MCTOPMTW-3-c107.txt	830	870	856.33	840	5112.50
MCTOPMTW-3-c108.txt	850	900	893.33	870	4195.03
MCTOPMTW-3-c109.txt	910	960	946.00	930	5476.60
MCTOPMTW-3-r101.txt	475	373	328.47	292	2588.13
MCTOPMTW-3-r102.txt	666	640	592.63	548	5348.83
MCTOPMTW-3-r103.txt	669	707	669.33	610	6132.87
MCTOPMTW-3-r104.txt	737	749	721.63	678	6393.93
MCTOPMTW-3-r105.txt	609	532	491.40	453	4293.57
MCTOPMTW-3-r106.txt	695	710	658.83	614	5270.20
MCTOPMTW-3-r107.txt	710	740	707.50	673	5916.93
MCTOPMTW-3-r108.txt	731	770	742.93	719	5195.03
MCTOPMTW-3-r109.txt	668	692	651.50	585	5536.00
MCTOPMTW-3-r110.txt	717	717	671.20	623	5496.00
MCTOPMTW-3-r111.txt	705	748	702.13	660	5434.07
MCTOPMTW-3-r112.txt	729	738	709.17	667	5029.33
MCTOPMTW-3-rc101.txt	574	515	481.30	443	4248.33
MCTOPMTW-3-rc102.txt	694	665	621.47	579	4579.30
MCTOPMTW-3-rc103.txt	703	728	684.07	639	5463.33
MCTOPMTW-3-rc104.txt	786	818	760.73	700	5741.63
MCTOPMTW-3-rc105.txt	629	624	555.27	514	4340.77
MCTOPMTW-3-rc106.txt	676	676	644.67	572	4895.53
MCTOPMTW-3-rc107.txt	715	737	700.17	654	5433.77
MCTOPMTW-3-rc108.txt	729	747	721.13	684	4250.77
MCTOPMTW-3-pr01.txt	572	590	562.37	529	2196.33
MCTOPMTW-3-pr02.txt	830	864	814.87	752	8026.63
MCTOPMTW-3-pr03.txt	868	918	830.77	686	14493.83
MCTOPMTW-3-pr04.txt	1112	1081	972.40	836	21039.07
MCTOPMTW-3-pr05.txt	1284	1215	1131.23	990	31996.87
MCTOPMTW-3-pr07.txt	688	687	647.63	613	5038.87
MCTOPMTW-3-pr08.txt	1030	1003	923.53	862	13785.40
MCTOPMTW-3-pr09.txt	1056	1032	925.07	787	25647.03
MCTOPMTW-4-c101.txt	960	860	800.00	750	6425.97
MCTOPMTW-4-c102.txt	1070	1100	1026.67	960	7610.97
MCTOPMTW-4-c103.txt	1090	1160	1134.00	1100	6204.87

Instance	ILS-GRASP	Best	Average	Worst	Average time (ms)
MCTOPMTW-4-c104.txt	1160	1210	1195.67	1170	6324.57
MCTOPMTW-4-c105.txt	1000	1030	996.33	950	6493.70
MCTOPMTW-4-c106.txt	1000	1030	1006.00	980	6479.77
MCTOPMTW-4-c107.txt	1060	1100	1077.33	1050	6778.70
MCTOPMTW-4-c108.txt	1030	1110	1098.67	1070	6921.90
MCTOPMTW-4-c109.txt	1090	1160	1147.33	1130	6782.67
MCTOPMTW-4-r101.txt	584	464	419.87	392	3386.23
MCTOPMTW-4-r102.txt	767	735	698.80	659	7010.73
MCTOPMTW-4-r103.txt	843	830	796.63	760	6984.63
MCTOPMTW-4-r104.txt	869	933	882.13	834	7755.63
MCTOPMTW-4-r105.txt	740	692	623.60	582	6105.97
MCTOPMTW-4-r106.txt	817	867	813.50	749	8412.57
MCTOPMTW-4-r107.txt	857	897	861.23	809	7196.23
MCTOPMTW-4-r108.txt	900	948	910.30	856	6840.47
MCTOPMTW-4-r109.txt	792	847	797.17	750	7357.33
MCTOPMTW-4-r110.txt	826	854	822.53	779	7314.50
MCTOPMTW-4-r111.txt	852	925	867.97	810	7685.13
MCTOPMTW-4-r112.txt	902	932	893.03	855	7606.40
MCTOPMTW-4-rc101.txt	745	671	614.97	559	5088.20
MCTOPMTW-4-rc102.txt	801	847	774.27	720	6695.87
MCTOPMTW-4-rc103.txt	881	911	862.03	819	6835.57
MCTOPMTW-4-rc104.txt	991	1035	966.13	920	6746.83
MCTOPMTW-4-rc105.txt	795	794	713.90	654	5684.80
MCTOPMTW-4-rc106.txt	837	875	807.27	753	6302.87
MCTOPMTW-4-rc107.txt	902	937	882.27	822	6561.70
MCTOPMTW-4-rc108.txt	945	974	916.03	866	6962.57
MCTOPMTW-4-pr01.txt	630	644	635.90	616	2149.13
MCTOPMTW-4-pr02.txt	974	1012	955.67	903	9314.53
MCTOPMTW-4-pr03.txt	1071	1061	985.53	878	17576.07
MCTOPMTW-4-pr04.txt	1329	1324	1197.83	1059	26326.70
MCTOPMTW-4-pr05.txt	1536	1585	1379.97	1266	36949.77
MCTOPMTW-4-pr07.txt	793	802	769.90	733	5657.03
MCTOPMTW-4-pr08.txt	1141	1209	1106.10	1041	17247.53
MCTOPMTW-4-pr09.txt	1321	1270	1153.67	1051	31344.63

APPENDIX C

Results for MCMTOPTW problem in *Solo* mode

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-c101.txt	690	632	450	3.60
MCMTOPMTW-1-c102.txt	1250	1245	1220	5.20
MCMTOPMTW-1-c103.txt	1810	1785.7	1747	7.00
MCMTOPMTW-1-c104.txt	2408	2375.5	2350	7.80
MCMTOPMTW-1-c105.txt	2261	2186.8	2010	11.80
MCMTOPMTW-1-c106.txt	2949	2857.3	2740	15.40
MCMTOPMTW-1-c107.txt	3602	3565.2	3520	17.70
MCMTOPMTW-1-c108.txt	4039	4020.8	3929	12.10
MCMTOPMTW-1-c109.txt	4706	4676.5	4659	12.40
MCMTOPMTW-1-r101.txt	308	308	308	0.90
MCMTOPMTW-1-r102.txt	1055	1019.2	910	5.80
MCMTOPMTW-1-r103.txt	1487	1457.4	1432	7.10
MCMTOPMTW-1-r104.txt	1797	1781.9	1773	8.40
MCMTOPMTW-1-r105.txt	1716	1637.6	1514	12.30
MCMTOPMTW-1-r106.txt	2738	2713	2667	15.00
MCMTOPMTW-1-r107.txt	3393	3365.3	3297	15.40
MCMTOPMTW-1-r108.txt	3687	3661.3	3642	14.90
MCMTOPMTW-1-r109.txt	3612	3572.4	3522	17.40
MCMTOPMTW-1-r110.txt	3961	3939.2	3907	17.70
MCMTOPMTW-1-r111.txt	3949	3895.7	3862	19.40
MCMTOPMTW-1-r112.txt	4370	4341.7	4255	17.80
MCMTOPMTW-1-rc101.txt	508	434.5	304	3.90
MCMTOPMTW-1-rc102.txt	899	843.9	762	5.60
MCMTOPMTW-1-rc103.txt	1374	1354.1	1302	6.50
MCMTOPMTW-1-rc104.txt	1759	1746.7	1702	6.10
MCMTOPMTW-1-rc105.txt	1951	1847.5	1668	11.60
MCMTOPMTW-1-rc106.txt	2265	2210.4	2107	13.00
MCMTOPMTW-1-rc107.txt	3011	2982.5	2951	13.10
MCMTOPMTW-1-rc108.txt	2960	2938.7	2915	16.30
MCMTOPMTW-1-pr01.txt	889	863.5	842	2.60
MCMTOPMTW-1-pr02.txt	1605	1555.2	1479	8.70
MCMTOPMTW-1-pr03.txt	2316	2235.5	2046	17.40
MCMTOPMTW-1-pr04.txt	3539	3407	3136	32.50
MCMTOPMTW-1-pr05.txt	5130	4979.4	4638	61.40

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-pr07.txt	3665	3581.6	3457	14.60
MCMTOPMTW-1-pr08.txt	6072	5920	5777	47.00
MCMTOPMTW-1-pr09.txt	7608	7389.9	6864	97.60
MCMTOPMTW-2-c101.txt	1360	1268	1040	6.70
MCMTOPMTW-2-c102.txt	2429	2409.1	2380	9.20
MCMTOPMTW-2-c103.txt	3500	3474.7	3420	14.20
MCMTOPMTW-2-c104.txt	4680	4657.4	4640	15.80
MCMTOPMTW-2-c105.txt	5095	5054.4	5035	20.80
MCMTOPMTW-2-c106.txt	5747	5695.7	5630	22.60
MCMTOPMTW-2-c107.txt	7407	7378.9	7343	28.40
MCMTOPMTW-2-c108.txt	8338	8305.2	8262	30.90
MCMTOPMTW-2-c109.txt	9472	9443.7	9418	32.80
MCMTOPMTW-2-r101.txt	607	571.9	522	3.10
MCMTOPMTW-2-r102.txt	1849	1768.1	1677	11.00
MCMTOPMTW-2-r103.txt	2616	2564.6	2508	13.50
MCMTOPMTW-2-r104.txt	3949	3920.3	3886	19.50
MCMTOPMTW-2-r105.txt	2973	2888.3	2746	14.90
MCMTOPMTW-2-r106.txt	4970	4890.9	4798	24.00
MCMTOPMTW-2-r107.txt	6162	6084.5	6046	26.30
MCMTOPMTW-2-r108.txt	7992	7899.6	7619	37.00
MCMTOPMTW-2-r109.txt	7345	7163.8	6924	33.50
MCMTOPMTW-2-r110.txt	7670	7524.3	7426	35.10
MCMTOPMTW-2-r111.txt	7701	7627.5	7556	34.30
MCMTOPMTW-2-r112.txt	7893	7829.9	7747	33.00
MCMTOPMTW-2-rc101.txt	973	920.8	865	5.60
MCMTOPMTW-2-rc102.txt	1772	1743.8	1707	8.80
MCMTOPMTW-2-rc103.txt	2573	2534.7	2445	14.30
MCMTOPMTW-2-rc104.txt	3626	3554.6	3486	17.40
MCMTOPMTW-2-rc105.txt	3597	3516.8	3475	17.60
MCMTOPMTW-2-rc106.txt	4373	4293.6	4218	20.00
MCMTOPMTW-2-rc107.txt	5320	5271.6	5219	21.30
MCMTOPMTW-2-rc108.txt	6159	6082	6042	23.90
MCMTOPMTW-2-pr01.txt	1094	1047.1	998	3.80
MCMTOPMTW-2-pr02.txt	2929	2847.3	2791	15.80
MCMTOPMTW-2-pr03.txt	4285	4140.3	4033	37.60
MCMTOPMTW-2-pr04.txt	6828	6528.1	6234	76.80
MCMTOPMTW-2-pr05.txt	9161	8952.1	8688	124.50
MCMTOPMTW-2-pr07.txt	7321	7220.1	7046	30.10
MCMTOPMTW-2-pr08.txt	10588	10322	10115	86.50
MCMTOPMTW-2-pr09.txt	13726	13276.2	12913	170.40

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-3-c101.txt	1740	1676.1	1620	10.80
MCMTOPMTW-3-c102.txt	3570	3516.6	3466	15.60
MCMTOPMTW-3-c103.txt	5205	5140.9	5098	20.30
MCMTOPMTW-3-c104.txt	6354	6298	6260	23.70
MCMTOPMTW-3-c105.txt	7236	7184.6	7121	28.80
MCMTOPMTW-3-c106.txt	8395	8308	8262	31.00
MCMTOPMTW-3-c107.txt	9972	9898.7	9802	41.60
MCMTOPMTW-3-c108.txt	11874	11828	11771	42.00
MCMTOPMTW-3-c109.txt	13566	13492.8	13435	46.90
MCMTOPMTW-3-r101.txt	947	875.4	804	5.40
MCMTOPMTW-3-r102.txt	2558	2437	2351	13.80
MCMTOPMTW-3-r103.txt	4211	4141	4091	25.60
MCMTOPMTW-3-r104.txt	5510	5399	5301	26.00
MCMTOPMTW-3-r105.txt	4587	4536.3	4482	24.20
MCMTOPMTW-3-r106.txt	6854	6801	6715	35.70
MCMTOPMTW-3-r107.txt	9256	9122.6	8985	45.50
MCMTOPMTW-3-r108.txt	10734	10640.9	10584	49.80
MCMTOPMTW-3-r109.txt	9720	9606	9352	45.60
MCMTOPMTW-3-r110.txt	10663	10552.2	10466	50.10
MCMTOPMTW-3-r111.txt	11591	11391.8	11169	55.60
MCMTOPMTW-3-r112.txt	11896	11813.2	11665	57.10
MCMTOPMTW-3-rc101.txt	1398	1302.7	1223	9.20
MCMTOPMTW-3-rc102.txt	2568	2484.8	2391	16.00
MCMTOPMTW-3-rc103.txt	3675	3621.6	3527	17.30
MCMTOPMTW-3-rc104.txt	5549	5460.8	5391	25.50
MCMTOPMTW-3-rc105.txt	4963	4849.9	4760	25.70
MCMTOPMTW-3-rc106.txt	6532	6447.4	6361	35.60
MCMTOPMTW-3-rc107.txt	7772	7642.6	7515	38.60
MCMTOPMTW-3-rc108.txt	9493	9355.5	9218	44.00
MCMTOPMTW-3-pr01.txt	1735	1703.5	1678	4.60
MCMTOPMTW-3-pr02.txt	4081	3920.3	3807	26.50
MCMTOPMTW-3-pr03.txt	5799	5667.2	5511	54.00
MCMTOPMTW-3-pr04.txt	8821	8594.3	8279	108.00
MCMTOPMTW-3-pr05.txt	12179	11954.6	11645	197.80
MCMTOPMTW-3-pr07.txt	9913	9741.1	9518	37.00
MCMTOPMTW-3-pr08.txt	14758	14501.9	14120	125.70
MCMTOPMTW-3-pr09.txt	19253	18955.1	18445	277.00
MCMTOPMTW-4-c101.txt	2240	2180.6	2130	14.70
MCMTOPMTW-4-c102.txt	4488	4445.6	4392	25.20
MCMTOPMTW-4-c103.txt	6420	6375	6327	29.20

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-4-c104.txt	8590	8546.5	8515	37.50
MCMTOPMTW-4-c105.txt	8896	8795.4	8681	37.30
MCMTOPMTW-4-c106.txt	10608	10558.5	10478	43.00
MCMTOPMTW-4-c107.txt	13566	13470.5	13359	60.40
MCMTOPMTW-4-c108.txt	14846	14757.6	14707	54.40
MCMTOPMTW-4-c109.txt	17345	17296	17245	68.00
MCMTOPMTW-4-r101.txt	1318	1113.7	1042	6.60
MCMTOPMTW-4-r102.txt	3108	3055.9	2957	18.90
MCMTOPMTW-4-r103.txt	4633	4527.3	4290	30.20
MCMTOPMTW-4-r104.txt	6968	6844.7	6706	42.90
MCMTOPMTW-4-r105.txt	6161	5991.3	5890	33.20
MCMTOPMTW-4-r106.txt	9040	8924.8	8829	55.00
MCMTOPMTW-4-r107.txt	10733	10548.7	10303	54.70
MCMTOPMTW-4-r108.txt	12397	12302.6	12153	62.10
MCMTOPMTW-4-r109.txt	13453	13263.9	13119	72.00
MCMTOPMTW-4-r110.txt	14139	14014	13894	67.70
MCMTOPMTW-4-r111.txt	14323	14113.2	13844	71.00
MCMTOPMTW-4-r112.txt	14299	14165.6	14026	73.90
MCMTOPMTW-4-rc101.txt	1612	1542.8	1471	11.30
MCMTOPMTW-4-rc102.txt	3146	3030.3	2807	17.50
MCMTOPMTW-4-rc103.txt	5187	5060.5	4959	28.20
MCMTOPMTW-4-rc104.txt	6857	6793.2	6708	35.80
MCMTOPMTW-4-rc105.txt	6419	6255.2	6089	30.00
MCMTOPMTW-4-rc106.txt	8164	8010.9	7850	40.90
MCMTOPMTW-4-rc107.txt	10679	10534.7	10399	52.20
MCMTOPMTW-4-rc108.txt	12783	12530.5	12340	60.70
MCMTOPMTW-4-pr01.txt	1955	1935.5	1919	4.30
MCMTOPMTW-4-pr02.txt	4684	4504.1	4369	29.20
MCMTOPMTW-4-pr03.txt	7507	7159.1	6962	71.70
MCMTOPMTW-4-pr04.txt	10978	10777.7	10494	136.50
MCMTOPMTW-4-pr05.txt	15246	14905.3	14475	264.60
MCMTOPMTW-4-pr07.txt	11472	11331.3	11218	45.20
MCMTOPMTW-4-pr08.txt	18332	17980.2	17706	161.50
MCMTOPMTW-4-pr09.txt	23634	23077.8	22704	323.80

APPENDIX D

Results for MCMTOPTW problem in *Subgroups* mode

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-c101.txt	700	643	430	3.10
MCMTOPMTW-1-c102.txt	1063	1051.3	1050	3.00
MCMTOPMTW-1-c103.txt	1682	1656.6	1616	2.80
MCMTOPMTW-1-c104.txt	2230	2226	2210	2.50
MCMTOPMTW-1-c105.txt	1849	1836.5	1744	2.50
MCMTOPMTW-1-c106.txt	2822	2766.7	2615	4.20
MCMTOPMTW-1-c107.txt	3234	3202.3	3145	3.30
MCMTOPMTW-1-c108.txt	3568	3556.1	3514	3.60
MCMTOPMTW-1-c109.txt	4312	4250.4	4158	2.10
MCMTOPMTW-1-r101.txt	308	308	308	0.90
MCMTOPMTW-1-r102.txt	1041	971.5	740	4.20
MCMTOPMTW-1-r103.txt	1373	1346.8	1182	3.50
MCMTOPMTW-1-r104.txt	1604	1589.9	1554	3.90
MCMTOPMTW-1-r105.txt	1643	1483.1	1267	3.60
MCMTOPMTW-1-r106.txt	2592	2537.9	2417	7.40
MCMTOPMTW-1-r107.txt	3224	3182.5	3124	2.80
MCMTOPMTW-1-r108.txt	3485	3423.5	3321	6.80
MCMTOPMTW-1-r109.txt	3397	3137.9	2783	3.20
MCMTOPMTW-1-r110.txt	3813	3699.1	3606	5.40
MCMTOPMTW-1-r111.txt	3490	3358.8	3317	2.80
MCMTOPMTW-1-r112.txt	4139	3983.3	3876	3.40
MCMTOPMTW-1-rc101.txt	519	453.6	364	4.40
MCMTOPMTW-1-rc102.txt	776	739.4	659	2.50
MCMTOPMTW-1-rc103.txt	1313	1300.8	1274	2.90
MCMTOPMTW-1-rc104.txt	1632	1627.2	1626	2.90
MCMTOPMTW-1-rc105.txt	1879	1704	1612	3.20
MCMTOPMTW-1-rc106.txt	2062	1921	1627	5.10
MCMTOPMTW-1-rc107.txt	2525	2437.7	2233	3.00
MCMTOPMTW-1-rc108.txt	2744	2585.1	2513	4.80
MCMTOPMTW-1-pr01.txt	889	859.5	840	3.10
MCMTOPMTW-1-pr02.txt	1622	1574.9	1499	6.10
MCMTOPMTW-1-pr03.txt	2422	2326.7	1948	8.50
MCMTOPMTW-1-pr04.txt	3469	3304.1	3173	18.40
MCMTOPMTW-1-pr05.txt	5419	5119.2	4658	21.00

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-pr07.txt	3746	3672.1	3417	3.60
MCMTOPMTW-1-pr08.txt	6090	5903.6	5074	14.20
MCMTOPMTW-1-pr09.txt	8699	8257	7821	16.70
MCMTOPMTW-2-c101.txt	1350	1289.7	1239	6.30
MCMTOPMTW-2-c102.txt	2438	2404.8	2370	6.50
MCMTOPMTW-2-c103.txt	3278	3253.2	3230	6.00
MCMTOPMTW-2-c104.txt	4460	4413.9	4345	10.00
MCMTOPMTW-2-c105.txt	5162	4992.1	4875	6.60
MCMTOPMTW-2-c106.txt	5652	5615.3	5570	11.10
MCMTOPMTW-2-c107.txt	7527	7211.5	6941	8.20
MCMTOPMTW-2-c108.txt	8165	8014.8	7941	15.70
MCMTOPMTW-2-c109.txt	9804	9536.4	9343	5.30
MCMTOPMTW-2-r101.txt	607	571.9	518	3.30
MCMTOPMTW-2-r102.txt	1786	1707.3	1632	7.10
MCMTOPMTW-2-r103.txt	2621	2554.5	2516	5.90
MCMTOPMTW-2-r104.txt	3924	3864.2	3795	11.20
MCMTOPMTW-2-r105.txt	2744	2637.3	2424	3.80
MCMTOPMTW-2-r106.txt	4972	4850.6	4706	5.20
MCMTOPMTW-2-r107.txt	6011	5905.4	5838	5.00
MCMTOPMTW-2-r108.txt	8729	8545.7	8185	6.70
MCMTOPMTW-2-r109.txt	7711	7458.6	6553	6.70
MCMTOPMTW-2-r110.txt	8425	8224.5	7990	6.40
MCMTOPMTW-2-r111.txt	7160	7070.3	6942	6.30
MCMTOPMTW-2-r112.txt	8546	8407.7	8222	5.70
MCMTOPMTW-2-rc101.txt	985	930.5	810	7.20
MCMTOPMTW-2-rc102.txt	1702	1667	1626	5.30
MCMTOPMTW-2-rc103.txt	2525	2474.7	2415	5.60
MCMTOPMTW-2-rc104.txt	3284	3252.4	3219	9.90
MCMTOPMTW-2-rc105.txt	3797	3521.7	3200	5.60
MCMTOPMTW-2-rc106.txt	4209	4138.8	4032	10.10
MCMTOPMTW-2-rc107.txt	5750	5238.8	5081	5.40
MCMTOPMTW-2-rc108.txt	5804	5631.7	5410	6.90
MCMTOPMTW-2-pr01.txt	1082	1047.4	1021	3.80
MCMTOPMTW-2-pr02.txt	3078	2985.4	2892	10.40
MCMTOPMTW-2-pr03.txt	4660	4474.8	4246	14.50
MCMTOPMTW-2-pr04.txt	6612	6264	5925	47.10
MCMTOPMTW-2-pr05.txt	9877	9239.9	8773	40.00
MCMTOPMTW-2-pr07.txt	8506	8308.8	8085	6.80
MCMTOPMTW-2-pr08.txt	12708	11873.5	11169	17.90
MCMTOPMTW-2-pr09.txt	14881	13957.8	13424	51.80

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-3-c101.txt	1734	1681.5	1537	9.50
MCMTOPMTW-3-c102.txt	3498	3460.8	3400	10.60
MCMTOPMTW-3-c103.txt	4940	4860.8	4808	9.70
MCMTOPMTW-3-c104.txt	6151	6084.7	6010	13.60
MCMTOPMTW-3-c105.txt	7160	7023.2	6933	7.50
MCMTOPMTW-3-c106.txt	8543	8366.5	8242	6.80
MCMTOPMTW-3-c107.txt	10178	10082.3	9980	8.00
MCMTOPMTW-3-c108.txt	12586	12435.2	12260	8.90
MCMTOPMTW-3-c109.txt	14332	14237.7	14123	7.50
MCMTOPMTW-3-r101.txt	902	880.7	838	5.70
MCMTOPMTW-3-r102.txt	2646	2533.1	2479	9.20
MCMTOPMTW-3-r103.txt	4452	4234	4016	11.60
MCMTOPMTW-3-r104.txt	5775	5644.5	5483	10.70
MCMTOPMTW-3-r105.txt	4836	4635.1	4298	8.20
MCMTOPMTW-3-r106.txt	6820	6654.4	6571	18.30
MCMTOPMTW-3-r107.txt	10403	9824.7	9594	9.70
MCMTOPMTW-3-r108.txt	11640	11050.6	10709	13.80
MCMTOPMTW-3-r109.txt	10565	10299.4	10068	8.10
MCMTOPMTW-3-r110.txt	12152	11658.8	11100	11.50
MCMTOPMTW-3-r111.txt	13287	13013.6	12499	8.50
MCMTOPMTW-3-r112.txt	13293	12182.6	11769	19.80
MCMTOPMTW-3-rc101.txt	1381	1303.1	1247	9.20
MCMTOPMTW-3-rc102.txt	2671	2563.4	2449	9.70
MCMTOPMTW-3-rc103.txt	3726	3670.4	3622	9.10
MCMTOPMTW-3-rc104.txt	5414	5331.4	5140	15.90
MCMTOPMTW-3-rc105.txt	5244	5046.5	4910	7.20
MCMTOPMTW-3-rc106.txt	6676	6501.3	6244	13.90
MCMTOPMTW-3-rc107.txt	8276	8082.1	7916	8.00
MCMTOPMTW-3-rc108.txt	9621	9365.5	9193	22.50
MCMTOPMTW-3-pr01.txt	1738	1700.2	1662	5.10
MCMTOPMTW-3-pr02.txt	4234	4068.3	3941	16.10
MCMTOPMTW-3-pr03.txt	5801	5574.4	5243	24.50
MCMTOPMTW-3-pr04.txt	9838	9084	8581	43.30
MCMTOPMTW-3-pr05.txt	12751	12068.7	11183	64.70
MCMTOPMTW-3-pr07.txt	11517	11216.4	10974	8.50
MCMTOPMTW-3-pr08.txt	15463	15063.3	14505	48.30
MCMTOPMTW-3-pr09.txt	22937	21706.7	19774	46.60
MCMTOPMTW-4-c101.txt	2320	2204.2	2140	17.10
MCMTOPMTW-4-c102.txt	4684	4592.2	4494	15.80
MCMTOPMTW-4-c103.txt	6457	6411.9	6347	10.60

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-4-c104.txt	8706	8571.5	8457	15.70
MCMTOPMTW-4-c105.txt	9263	9065.6	8818	10.60
MCMTOPMTW-4-c106.txt	11239	11099.9	11034	9.80
MCMTOPMTW-4-c107.txt	14857	14674.4	14433	13.90
MCMTOPMTW-4-c108.txt	15541	15401.2	15286	15.10
MCMTOPMTW-4-c109.txt	20398	20018.4	19730	11.90
MCMTOPMTW-4-r101.txt	1176	1130.7	1096	6.90
MCMTOPMTW-4-r102.txt	3303	3245.3	3189	12.20
MCMTOPMTW-4-r103.txt	4596	4453.8	4233	13.90
MCMTOPMTW-4-r104.txt	7014	6795.5	6560	22.90
MCMTOPMTW-4-r105.txt	6361	6144.6	5626	10.10
MCMTOPMTW-4-r106.txt	10464	9936.2	9484	13.90
MCMTOPMTW-4-r107.txt	11163	10947	10742	24.60
MCMTOPMTW-4-r108.txt	14319	13848	13275	11.00
MCMTOPMTW-4-r109.txt	16316	14979	13852	19.90
MCMTOPMTW-4-r110.txt	15117	14499.7	14146	25.40
MCMTOPMTW-4-r111.txt	16755	16559.5	16277	10.50
MCMTOPMTW-4-r112.txt	17415	16840.5	16263	13.60
MCMTOPMTW-4-rc101.txt	1614	1549.3	1495	11.00
MCMTOPMTW-4-rc102.txt	3201	3111.4	3034	13.10
MCMTOPMTW-4-rc103.txt	5532	5301.9	5023	14.60
MCMTOPMTW-4-rc104.txt	7504	7221.1	6854	15.90
MCMTOPMTW-4-rc105.txt	7550	7309.5	7027	9.60
MCMTOPMTW-4-rc106.txt	8153	7959.1	7786	22.30
MCMTOPMTW-4-rc107.txt	11944	11436.8	11041	12.90
MCMTOPMTW-4-rc108.txt	14979	13833.4	13222	23.00
MCMTOPMTW-4-pr01.txt	1950	1926.6	1905	4.40
MCMTOPMTW-4-pr02.txt	4802	4683	4571	19.50
MCMTOPMTW-4-pr03.txt	7988	7569.5	7162	36.80
MCMTOPMTW-4-pr04.txt	11761	10993.9	10471	53.00
MCMTOPMTW-4-pr05.txt	16980	16043.5	15157	93.10
MCMTOPMTW-4-pr07.txt	14569	14256.5	13736	12.60
MCMTOPMTW-4-pr08.txt	22676	22037.2	21163	30.50
MCMTOPMTW-4-pr09.txt	26788	25202.5	23412	88.50

APPENDIX E

Results for MCMTOPTW problem in *Group* mode

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-c101.txt	567	543.6	514	1.60
MCMTOPMTW-1-c102.txt	940	925	870	1.70
MCMTOPMTW-1-c103.txt	1560	1558	1540	1.00
MCMTOPMTW-1-c104.txt	2100	2082.6	1926	1.20
MCMTOPMTW-1-c105.txt	1834	1834	1834	1.60
MCMTOPMTW-1-c106.txt	3210	3147.6	2923	2.20
MCMTOPMTW-1-c107.txt	3548	3340	2834	1.20
MCMTOPMTW-1-c108.txt	3925	3925	3925	1.00
MCMTOPMTW-1-c109.txt	4514	4504	4494	0.80
MCMTOPMTW-1-r101.txt	216	216	216	0.40
MCMTOPMTW-1-r102.txt	911	869.1	714	1.80
MCMTOPMTW-1-r103.txt	1592	1524.5	1492	1.60
MCMTOPMTW-1-r104.txt	1477	1477	1477	1.40
MCMTOPMTW-1-r105.txt	1707	1459.3	1347	1.40
MCMTOPMTW-1-r106.txt	2832	2379.8	1170	1.60
MCMTOPMTW-1-r107.txt	3671	3639.3	3598	1.10
MCMTOPMTW-1-r108.txt	3696	3679.1	3527	0.90
MCMTOPMTW-1-r109.txt	3628	3423.4	2402	1.60
MCMTOPMTW-1-r110.txt	4693	4620.9	3972	1.50
MCMTOPMTW-1-r111.txt	4293	4031.5	1919	1.20
MCMTOPMTW-1-r112.txt	4990	4988.6	4988	1.50
MCMTOPMTW-1-rc101.txt	473	449.4	387	1.80
MCMTOPMTW-1-rc102.txt	720	686.3	606	1.60
MCMTOPMTW-1-rc103.txt	1246	1246	1246	1.40
MCMTOPMTW-1-rc104.txt	1227	1218.2	1139	1.20
MCMTOPMTW-1-rc105.txt	1935	1710.2	1083	1.40
MCMTOPMTW-1-rc106.txt	2381	2336.8	2161	1.50
MCMTOPMTW-1-rc107.txt	2699	2626.6	2518	0.70
MCMTOPMTW-1-rc108.txt	2156	2133.4	2043	1.20
MCMTOPMTW-1-pr01.txt	984	937.5	874	1.30
MCMTOPMTW-1-pr02.txt	1675	1621.9	1536	3.00
MCMTOPMTW-1-pr03.txt	2327	2150.8	1486	4.30
MCMTOPMTW-1-pr04.txt	3564	3209.7	2223	4.70
MCMTOPMTW-1-pr05.txt	6050	5502.1	4226	9.20

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-pr07.txt	3979	3868.6	3680	1.30
MCMTOPMTW-1-pr08.txt	7945	6827.3	4519	4.50
MCMTOPMTW-1-pr09.txt	10936	9368.5	6633	8.40
MCMTOPMTW-2-c101.txt	1440	1357.1	1291	2.50
MCMTOPMTW-2-c102.txt	2428	2388.9	2339	3.00
MCMTOPMTW-2-c103.txt	3380	3329.5	3275	3.30
MCMTOPMTW-2-c104.txt	4524	4449.5	4387	2.20
MCMTOPMTW-2-c105.txt	5472	5360.4	5230	2.50
MCMTOPMTW-2-c106.txt	6050	5765.9	5267	3.20
MCMTOPMTW-2-c107.txt	7780	7767	7740	3.00
MCMTOPMTW-2-c108.txt	9847	9638.2	9420	2.50
MCMTOPMTW-2-c109.txt	11598	11317.2	11204	2.60
MCMTOPMTW-2-r101.txt	531	514.5	468	1.50
MCMTOPMTW-2-r102.txt	1734	1673.7	1579	2.50
MCMTOPMTW-2-r103.txt	2801	2734.4	2563	3.10
MCMTOPMTW-2-r104.txt	4283	4079.5	3941	2.70
MCMTOPMTW-2-r105.txt	2771	2727.6	2576	2.10
MCMTOPMTW-2-r106.txt	5508	5238.1	4884	2.60
MCMTOPMTW-2-r107.txt	6528	6439.1	6100	2.10
MCMTOPMTW-2-r108.txt	10423	10156.5	10031	3.10
MCMTOPMTW-2-r109.txt	9268	8771.9	7590	3.10
MCMTOPMTW-2-r110.txt	9628	9194.4	8736	2.20
MCMTOPMTW-2-r111.txt	8795	8231	7678	2.40
MCMTOPMTW-2-r112.txt	9984	9608.6	9268	2.30
MCMTOPMTW-2-rc101.txt	1018	926.8	863	2.60
MCMTOPMTW-2-rc102.txt	1699	1604	1539	3.10
MCMTOPMTW-2-rc103.txt	2669	2609.5	2495	2.20
MCMTOPMTW-2-rc104.txt	2965	2946.6	2903	2.90
MCMTOPMTW-2-rc105.txt	3737	3728.9	3696	2.00
MCMTOPMTW-2-rc106.txt	4625	4541.3	4381	2.20
MCMTOPMTW-2-rc107.txt	6202	6156.3	6078	2.00
MCMTOPMTW-2-rc108.txt	6290	6169.9	5970	1.90
MCMTOPMTW-2-pr01.txt	1240	1189.6	1156	1.40
MCMTOPMTW-2-pr02.txt	2985	2898.8	2727	4.70
MCMTOPMTW-2-pr03.txt	4762	4521.4	4232	6.30
MCMTOPMTW-2-pr04.txt	7412	6954.9	6704	13.70
MCMTOPMTW-2-pr05.txt	10847	10218.9	9599	15.90
MCMTOPMTW-2-pr07.txt	9732	9304.4	8856	3.10
MCMTOPMTW-2-pr08.txt	14360	13642	12828	9.50
MCMTOPMTW-2-pr09.txt	19848	17784.3	15200	13.00

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-3-c101.txt	1753	1695.2	1652	3.80
MCMTOPMTW-3-c102.txt	3646	3589.8	3538	5.00
MCMTOPMTW-3-c103.txt	4968	4894.4	4832	5.20
MCMTOPMTW-3-c104.txt	6466	6435.7	6353	4.40
MCMTOPMTW-3-c105.txt	7843	7674.4	7477	3.40
MCMTOPMTW-3-c106.txt	9574	9136.2	8902	3.70
MCMTOPMTW-3-c107.txt	11480	11275	11052	4.50
MCMTOPMTW-3-c108.txt	14704	14264.2	13990	3.20
MCMTOPMTW-3-c109.txt	16682	16388.3	16201	3.50
MCMTOPMTW-3-r101.txt	1031	957.3	881	3.00
MCMTOPMTW-3-r102.txt	2531	2380	2244	3.60
MCMTOPMTW-3-r103.txt	4594	4480.7	4357	6.00
MCMTOPMTW-3-r104.txt	6203	6080.9	5757	5.10
MCMTOPMTW-3-r105.txt	5342	5025.5	4724	3.70
MCMTOPMTW-3-r106.txt	7275	7093.7	6799	3.90
MCMTOPMTW-3-r107.txt	11685	11178.3	10384	4.70
MCMTOPMTW-3-r108.txt	13583	13137	12822	4.80
MCMTOPMTW-3-r109.txt	12251	11811.9	11521	3.40
MCMTOPMTW-3-r110.txt	15443	14951.5	14423	4.40
MCMTOPMTW-3-r111.txt	15796	15412.8	15186	3.90
MCMTOPMTW-3-r112.txt	16340	15792.7	15279	4.20
MCMTOPMTW-3-rc101.txt	1385	1256.3	1166	4.00
MCMTOPMTW-3-rc102.txt	2609	2517.6	2394	5.00
MCMTOPMTW-3-rc103.txt	3717	3632.4	3565	4.30
MCMTOPMTW-3-rc104.txt	6323	6170.9	5942	4.90
MCMTOPMTW-3-rc105.txt	5918	5467.6	4798	5.00
MCMTOPMTW-3-rc106.txt	8417	7538.4	5990	4.60
MCMTOPMTW-3-rc107.txt	9432	9195.7	8897	4.50
MCMTOPMTW-3-rc108.txt	12115	11791.8	11518	4.10
MCMTOPMTW-3-pr01.txt	1928	1882	1834	2.00
MCMTOPMTW-3-pr02.txt	4278	4107.5	3959	8.50
MCMTOPMTW-3-pr03.txt	5618	5306.5	4982	11.50
MCMTOPMTW-3-pr04.txt	10901	9753.6	8897	22.30
MCMTOPMTW-3-pr05.txt	14847	14215.7	13138	27.70
MCMTOPMTW-3-pr07.txt	13732	13268.1	13014	4.00
MCMTOPMTW-3-pr08.txt	20058	18869.2	17349	13.60
MCMTOPMTW-3-pr09.txt	27276	25544.4	23194	17.30
MCMTOPMTW-4-c101.txt	2292	2216.8	2148	6.40
MCMTOPMTW-4-c102.txt	4678	4608	4522	8.30
MCMTOPMTW-4-c103.txt	6800	6671	6550	4.50

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-4-c104.txt	8624	8514.4	8470	6.40
MCMTOPMTW-4-c105.txt	9811	9731.1	9642	5.10
MCMTOPMTW-4-c106.txt	12880	12469	12178	5.30
MCMTOPMTW-4-c107.txt	17457	17178.5	16629	6.40
MCMTOPMTW-4-c108.txt	18864	18677.8	18420	5.10
MCMTOPMTW-4-c109.txt	24075	23735.6	23209	4.90
MCMTOPMTW-4-r101.txt	1235	1114.7	1075	3.00
MCMTOPMTW-4-r102.txt	3445	3310.5	3109	5.50
MCMTOPMTW-4-r103.txt	4843	4549.3	4233	5.30
MCMTOPMTW-4-r104.txt	7929	7516.9	7246	8.20
MCMTOPMTW-4-r105.txt	6574	6202.3	5847	3.30
MCMTOPMTW-4-r106.txt	12067	11640.7	11054	7.80
MCMTOPMTW-4-r107.txt	14017	13761.9	13591	5.40
MCMTOPMTW-4-r108.txt	17324	16861.7	16482	5.40
MCMTOPMTW-4-r109.txt	19649	18400.8	17758	6.50
MCMTOPMTW-4-r110.txt	20833	20312.1	19555	5.40
MCMTOPMTW-4-r111.txt	21553	20795.4	20205	6.60
MCMTOPMTW-4-r112.txt	21574	20805.2	19224	6.90
MCMTOPMTW-4-rc101.txt	1490	1419.9	1347	5.10
MCMTOPMTW-4-rc102.txt	3191	3021.5	2871	5.70
MCMTOPMTW-4-rc103.txt	5615	5403.4	5085	6.60
MCMTOPMTW-4-rc104.txt	8252	7947.3	7759	7.40
MCMTOPMTW-4-rc105.txt	7442	7129.9	6679	4.20
MCMTOPMTW-4-rc106.txt	9512	9304.6	8971	4.70
MCMTOPMTW-4-rc107.txt	14080	13745.8	13214	6.80
MCMTOPMTW-4-rc108.txt	18557	17587.8	17213	5.80
MCMTOPMTW-4-pr01.txt	2187	2170.3	2145	2.10
MCMTOPMTW-4-pr02.txt	5300	5079.5	4812	10.20
MCMTOPMTW-4-pr03.txt	8877	8095.5	7285	16.60
MCMTOPMTW-4-pr04.txt	12253	11315	10470	22.50
MCMTOPMTW-4-pr05.txt	19443	18574	17227	45.70
MCMTOPMTW-4-pr07.txt	18371	17620.3	16626	5.20
MCMTOPMTW-4-pr08.txt	27538	25866.2	23343	13.80
MCMTOPMTW-4-pr09.txt	33920	31027.7	28102	30.80

APPENDIX F

Results for MCMTOPTW problem in *Combined-Slow* mode

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-c101.txt	725	719.3	712	78.70
MCMTOPMTW-1-c102.txt	1106	1094	1076	64.40
MCMTOPMTW-1-c103.txt	1727	1722.1	1704	26.80
MCMTOPMTW-1-c104.txt	2320	2306.8	2268	16.20
MCMTOPMTW-1-c105.txt	2276	2242.8	2237	28.60
MCMTOPMTW-1-c106.txt	3262	3238.8	3230	17.40
MCMTOPMTW-1-c107.txt	3845	3814.7	3775	9.20
MCMTOPMTW-1-c108.txt	4038	4015.6	4008	9.80
MCMTOPMTW-1-c109.txt	5259	5188.1	5123	12.40
MCMTOPMTW-1-r101.txt	361	361	361	17.60
MCMTOPMTW-1-r102.txt	1038	1019.2	1005	131.80
MCMTOPMTW-1-r103.txt	1612	1612	1612	70.30
MCMTOPMTW-1-r104.txt	1554	1552.2	1552	20.10
MCMTOPMTW-1-r105.txt	1616	1520.3	1451	30.70
MCMTOPMTW-1-r106.txt	2917	2903.5	2877	32.70
MCMTOPMTW-1-r107.txt	3684	3683	3681	11.20
MCMTOPMTW-1-r108.txt	3844	3844	3844	17.30
MCMTOPMTW-1-r109.txt	4201	4169.9	4091	20.10
MCMTOPMTW-1-r110.txt	4700	4700	4700	11.80
MCMTOPMTW-1-r111.txt	4293	4290.1	4264	9.60
MCMTOPMTW-1-r112.txt	5316	5156.9	4989	9.50
MCMTOPMTW-1-rc101.txt	525	500.3	495	146.80
MCMTOPMTW-1-rc102.txt	832	830.2	830	111.70
MCMTOPMTW-1-rc103.txt	1266	1263.6	1263	40.00
MCMTOPMTW-1-rc104.txt	1652	1576.9	1552	22.10
MCMTOPMTW-1-rc105.txt	1969	1950.6	1948	51.10
MCMTOPMTW-1-rc106.txt	2555	2486.3	2434	36.30
MCMTOPMTW-1-rc107.txt	3045	2930.4	2745	8.50
MCMTOPMTW-1-rc108.txt	2795	2585.1	2482	18.00
MCMTOPMTW-1-pr01.txt	985	983	972	32.80
MCMTOPMTW-1-pr02.txt	1760	1737.6	1722	128.10
MCMTOPMTW-1-pr03.txt	2478	2440.6	2398	173.10
MCMTOPMTW-1-pr04.txt	3808	3694.4	3643	159.60
MCMTOPMTW-1-pr05.txt	6459	6308.8	6076	222.10

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-pr07.txt	4187	4134.6	4099	31.40
MCMTOPMTW-1-pr08.txt	7933	7863.2	7729	64.50
MCMTOPMTW-1-pr09.txt	11575	10795.2	10325	147.10
MCMTOPMTW-2-c101.txt	1511	1473	1448	203.20
MCMTOPMTW-2-c102.txt	2539	2509.9	2489	138.10
MCMTOPMTW-2-c103.txt	3615	3575	3531	64.80
MCMTOPMTW-2-c104.txt	4804	4740.4	4692	25.30
MCMTOPMTW-2-c105.txt	5695	5539.8	5479	29.60
MCMTOPMTW-2-c106.txt	6199	6104.2	5971	20.90
MCMTOPMTW-2-c107.txt	8277	8195.6	8041	18.80
MCMTOPMTW-2-c108.txt	10021	9972	9880	14.50
MCMTOPMTW-2-c109.txt	11588	11358.1	11219	12.80
MCMTOPMTW-2-r101.txt	643	625.8	609	81.20
MCMTOPMTW-2-r102.txt	1866	1849	1830	169.70
MCMTOPMTW-2-r103.txt	2886	2861.7	2832	122.20
MCMTOPMTW-2-r104.txt	4358	4322.7	4283	86.70
MCMTOPMTW-2-r105.txt	3175	3145.7	3078	59.80
MCMTOPMTW-2-r106.txt	5671	5574.2	5444	33.90
MCMTOPMTW-2-r107.txt	6873	6777.9	6699	20.30
MCMTOPMTW-2-r108.txt	10609	10492.7	10377	23.80
MCMTOPMTW-2-r109.txt	9340	9193.8	8948	22.80
MCMTOPMTW-2-r110.txt	9829	9698.5	9562	19.10
MCMTOPMTW-2-r111.txt	9305	8885.6	8769	18.50
MCMTOPMTW-2-r112.txt	9996	9829.7	9685	13.80
MCMTOPMTW-2-rc101.txt	1045	1027.2	1018	235.80
MCMTOPMTW-2-rc102.txt	1785	1764.1	1745	214.30
MCMTOPMTW-2-rc103.txt	2749	2729.2	2707	97.30
MCMTOPMTW-2-rc104.txt	3298	3239	3177	51.70
MCMTOPMTW-2-rc105.txt	4028	3965.9	3906	57.00
MCMTOPMTW-2-rc106.txt	4877	4801.2	4742	39.00
MCMTOPMTW-2-rc107.txt	6680	6622	6602	34.80
MCMTOPMTW-2-rc108.txt	6519	6496.5	6481	20.50
MCMTOPMTW-2-pr01.txt	1282	1272.7	1251	178.70
MCMTOPMTW-2-pr02.txt	3138	3104.4	3075	262.60
MCMTOPMTW-2-pr03.txt	5075	4964.4	4860	328.80
MCMTOPMTW-2-pr04.txt	8209	7949.3	7764	398.20
MCMTOPMTW-2-pr05.txt	12082	11722.5	11303	352.40
MCMTOPMTW-2-pr07.txt	10224	10037.3	9950	73.70
MCMTOPMTW-2-pr08.txt	14995	14508.8	14053	101.40
MCMTOPMTW-2-pr09.txt	21025	19727.4	19073	146.00

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-3-c101.txt	1892	1864.8	1847	252.60
MCMTOPMTW-3-c102.txt	3738	3704.2	3680	203.70
MCMTOPMTW-3-c103.txt	5207	5180.1	5146	81.00
MCMTOPMTW-3-c104.txt	6870	6814.9	6756	28.70
MCMTOPMTW-3-c105.txt	8072	7975.7	7892	34.20
MCMTOPMTW-3-c106.txt	9769	9485.2	9287	12.50
MCMTOPMTW-3-c107.txt	11854	11642.5	11352	14.00
MCMTOPMTW-3-c108.txt	14989	14706.3	14562	15.90
MCMTOPMTW-3-c109.txt	17026	16629.8	16290	18.30
MCMTOPMTW-3-r101.txt	1137	1109	1073	210.10
MCMTOPMTW-3-r102.txt	2624	2589.4	2534	212.20
MCMTOPMTW-3-r103.txt	4850	4786.6	4728	213.80
MCMTOPMTW-3-r104.txt	6411	6316.1	6254	115.60
MCMTOPMTW-3-r105.txt	5539	5384	5239	72.90
MCMTOPMTW-3-r106.txt	7814	7547.1	7388	29.80
MCMTOPMTW-3-r107.txt	11964	11901.5	11695	61.70
MCMTOPMTW-3-r108.txt	13570	13378	13155	27.60
MCMTOPMTW-3-r109.txt	12316	11937.1	11607	18.70
MCMTOPMTW-3-r110.txt	15717	15282.6	14535	44.80
MCMTOPMTW-3-r111.txt	16180	15763.3	15539	27.10
MCMTOPMTW-3-r112.txt	16638	16203	15397	27.00
MCMTOPMTW-3-rc101.txt	1519	1449.6	1387	291.90
MCMTOPMTW-3-rc102.txt	2716	2679	2640	279.70
MCMTOPMTW-3-rc103.txt	3912	3863.7	3829	203.70
MCMTOPMTW-3-rc104.txt	6459	6399.5	6350	143.10
MCMTOPMTW-3-rc105.txt	6281	6066.6	5890	128.50
MCMTOPMTW-3-rc106.txt	8498	8400.4	8264	65.00
MCMTOPMTW-3-rc107.txt	9942	9693.8	9587	90.30
MCMTOPMTW-3-rc108.txt	12399	12204	11890	18.00
MCMTOPMTW-3-pr01.txt	1966	1944.9	1933	215.10
MCMTOPMTW-3-pr02.txt	4390	4339.9	4290	403.90
MCMTOPMTW-3-pr03.txt	6070	5965.4	5895	478.30
MCMTOPMTW-3-pr04.txt	10914	10572.9	10265	362.80
MCMTOPMTW-3-pr05.txt	15868	15528.8	14902	307.00
MCMTOPMTW-3-pr07.txt	14226	13885.3	13592	107.90
MCMTOPMTW-3-pr08.txt	20862	20345.9	19344	135.20
MCMTOPMTW-3-pr09.txt	29418	28242.3	27632	103.20
MCMTOPMTW-4-c101.txt	2442	2410.4	2388	413.90
MCMTOPMTW-4-c102.txt	4750	4724.6	4698	282.00
MCMTOPMTW-4-c103.txt	6949	6887.2	6799	109.20

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-4-c104.txt	9130	8999.6	8888	36.80
MCMTOPMTW-4-c105.txt	10206	10044.5	9960	41.80
MCMTOPMTW-4-c106.txt	13131	12798.2	12531	17.50
MCMTOPMTW-4-c107.txt	17397	17154.7	16733	19.60
MCMTOPMTW-4-c108.txt	19133	18857.8	18541	21.20
MCMTOPMTW-4-c109.txt	24181	23842.3	23240	25.10
MCMTOPMTW-4-r101.txt	1383	1349.4	1320	206.80
MCMTOPMTW-4-r102.txt	3520	3469.1	3417	273.20
MCMTOPMTW-4-r103.txt	4960	4881.9	4830	216.10
MCMTOPMTW-4-r104.txt	7971	7804.8	7688	240.30
MCMTOPMTW-4-r105.txt	7121	6892.2	6706	62.20
MCMTOPMTW-4-r106.txt	12678	12444.2	12267	115.20
MCMTOPMTW-4-r107.txt	14443	14274.6	14085	56.70
MCMTOPMTW-4-r108.txt	17858	17416.6	17025	43.10
MCMTOPMTW-4-r109.txt	19571	19054.8	18196	67.70
MCMTOPMTW-4-r110.txt	21333	21156.1	20972	54.50
MCMTOPMTW-4-r111.txt	21698	21228.4	20625	38.10
MCMTOPMTW-4-r112.txt	21617	21098.9	20001	38.70
MCMTOPMTW-4-rc101.txt	1634	1612.1	1595	333.70
MCMTOPMTW-4-rc102.txt	3358	3324.7	3288	295.40
MCMTOPMTW-4-rc103.txt	5763	5676.6	5627	303.10
MCMTOPMTW-4-rc104.txt	8281	8206.3	8142	141.70
MCMTOPMTW-4-rc105.txt	7780	7654.5	7521	81.50
MCMTOPMTW-4-rc106.txt	10244	9949.5	9732	55.80
MCMTOPMTW-4-rc107.txt	14385	14145.6	13728	79.90
MCMTOPMTW-4-rc108.txt	18521	18153.8	17721	27.00
MCMTOPMTW-4-pr01.txt	2205	2200.3	2196	808.60
MCMTOPMTW-4-pr02.txt	5417	5341.1	5279	1906.70
MCMTOPMTW-4-pr03.txt	8851	8620.8	8362	2346.50
MCMTOPMTW-4-pr04.txt	12863	12650	12402	1422.50
MCMTOPMTW-4-pr05.txt	20165	19455.8	18841	1239.90
MCMTOPMTW-4-pr07.txt	18741	18380.6	18056	233.10
MCMTOPMTW-4-pr08.txt	27653	27027.5	25800	88.80
MCMTOPMTW-4-pr09.txt	35008	33818.4	32524	182.80

APPENDIX G

Results for MCMTOPTW problem in *Combined-Fast* mode

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-c101.txt	725	707.3	648	18.80
MCMTOPMTW-1-c102.txt	1092	1074.6	1055	16.30
MCMTOPMTW-1-c103.txt	1725	1722.9	1704	8.30
MCMTOPMTW-1-c104.txt	2320	2280.4	2239	5.70
MCMTOPMTW-1-c105.txt	2246	2237.9	2237	4.50
MCMTOPMTW-1-c106.txt	3239	3230.9	3230	6.00
MCMTOPMTW-1-c107.txt	3845	3774.6	3701	4.50
MCMTOPMTW-1-c108.txt	4014	4014	4014	9.00
MCMTOPMTW-1-c109.txt	5187	5133.7	5113	4.70
MCMTOPMTW-1-r101.txt	361	361	361	5.20
MCMTOPMTW-1-r102.txt	1038	1006.4	986	19.30
MCMTOPMTW-1-r103.txt	1612	1611.2	1604	12.70
MCMTOPMTW-1-r104.txt	1552	1549.6	1548	4.20
MCMTOPMTW-1-r105.txt	1554	1474.3	1451	7.70
MCMTOPMTW-1-r106.txt	2912	2867.9	2793	10.20
MCMTOPMTW-1-r107.txt	3690	3680.9	3671	5.20
MCMTOPMTW-1-r108.txt	3844	3841.8	3822	10.20
MCMTOPMTW-1-r109.txt	4261	4055.5	3876	5.60
MCMTOPMTW-1-r110.txt	4700	4700	4700	6.10
MCMTOPMTW-1-r111.txt	4293	4277.6	4261	5.80
MCMTOPMTW-1-r112.txt	5316	5023.2	4989	5.60
MCMTOPMTW-1-rc101.txt	513	495.9	489	23.40
MCMTOPMTW-1-rc102.txt	830	830	830	9.70
MCMTOPMTW-1-rc103.txt	1263	1262.2	1255	8.50
MCMTOPMTW-1-rc104.txt	1563	1542.3	1497	7.10
MCMTOPMTW-1-rc105.txt	1948	1919.7	1894	4.10
MCMTOPMTW-1-rc106.txt	2555	2426.8	2389	6.40
MCMTOPMTW-1-rc107.txt	2936	2876.8	2725	3.70
MCMTOPMTW-1-rc108.txt	2559	2517.7	2488	7.40
MCMTOPMTW-1-pr01.txt	985	981.9	972	16.20
MCMTOPMTW-1-pr02.txt	1742	1719.7	1697	15.90
MCMTOPMTW-1-pr03.txt	2486	2383.9	2268	16.90
MCMTOPMTW-1-pr04.txt	3671	3563.2	3458	22.30
MCMTOPMTW-1-pr05.txt	6393	6099.5	5864	30.90

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-1-pr07.txt	4155	4027.4	3782	5.10
MCMTOPMTW-1-pr08.txt	7921	7750	7261	13.70
MCMTOPMTW-1-pr09.txt	11388	10801	10319	27.90
MCMTOPMTW-2-c101.txt	1471	1451.5	1434	29.70
MCMTOPMTW-2-c102.txt	2531	2504.3	2459	27.40
MCMTOPMTW-2-c103.txt	3613	3564	3494	15.60
MCMTOPMTW-2-c104.txt	4774	4710	4616	10.70
MCMTOPMTW-2-c105.txt	5535	5479.5	5403	7.90
MCMTOPMTW-2-c106.txt	6229	6119.9	5951	8.90
MCMTOPMTW-2-c107.txt	8232	8103.9	7889	7.00
MCMTOPMTW-2-c108.txt	10038	9947.3	9742	8.60
MCMTOPMTW-2-c109.txt	11626	11393	11204	10.20
MCMTOPMTW-2-r101.txt	629	621.2	602	13.00
MCMTOPMTW-2-r102.txt	1866	1811.1	1787	24.70
MCMTOPMTW-2-r103.txt	2880	2850.8	2817	14.40
MCMTOPMTW-2-r104.txt	4350	4221.9	4134	13.00
MCMTOPMTW-2-r105.txt	3116	3046.3	2978	6.70
MCMTOPMTW-2-r106.txt	5530	5403.5	4985	10.00
MCMTOPMTW-2-r107.txt	6974	6720.2	6532	8.00
MCMTOPMTW-2-r108.txt	10558	10295.5	10046	8.60
MCMTOPMTW-2-r109.txt	9245	8996.7	8601	9.50
MCMTOPMTW-2-r110.txt	9807	9534.5	9165	9.50
MCMTOPMTW-2-r111.txt	9212	8686	8277	10.00
MCMTOPMTW-2-r112.txt	9772	9737.8	9628	11.60
MCMTOPMTW-2-rc101.txt	1023	1008.5	988	23.70
MCMTOPMTW-2-rc102.txt	1768	1742.1	1706	19.00
MCMTOPMTW-2-rc103.txt	2728	2710.4	2676	12.30
MCMTOPMTW-2-rc104.txt	3215	3138.7	3064	9.20
MCMTOPMTW-2-rc105.txt	4028	3917.7	3819	11.00
MCMTOPMTW-2-rc106.txt	4877	4729.9	4645	9.20
MCMTOPMTW-2-rc107.txt	6613	6542.4	6098	7.90
MCMTOPMTW-2-rc108.txt	6496	6456.1	6290	6.40
MCMTOPMTW-2-pr01.txt	1282	1251.4	1215	21.50
MCMTOPMTW-2-pr02.txt	3104	3034.8	2811	34.30
MCMTOPMTW-2-pr03.txt	4989	4832.1	4669	28.20
MCMTOPMTW-2-pr04.txt	7996	7735.7	7420	36.70
MCMTOPMTW-2-pr05.txt	11864	11622.4	11437	69.20
MCMTOPMTW-2-pr07.txt	9944	9680.4	9179	10.30
MCMTOPMTW-2-pr08.txt	14643	14117.6	13514	20.80
MCMTOPMTW-2-pr09.txt	20202	18656.1	16789	33.80

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-3-c101.txt	1927	1851.2	1808	44.10
MCMTOPMTW-3-c102.txt	3720	3693.8	3656	37.90
MCMTOPMTW-3-c103.txt	5194	5150.9	5086	19.30
MCMTOPMTW-3-c104.txt	6922	6814.4	6710	15.60
MCMTOPMTW-3-c105.txt	8129	7978.8	7825	8.50
MCMTOPMTW-3-c106.txt	9711	9468.1	9272	10.80
MCMTOPMTW-3-c107.txt	11964	11701.9	11406	10.80
MCMTOPMTW-3-c108.txt	14817	14663.5	14306	12.50
MCMTOPMTW-3-c109.txt	16945	16672.3	16296	14.40
MCMTOPMTW-3-r101.txt	1100	1061.5	1043	28.20
MCMTOPMTW-3-r102.txt	2620	2562.2	2512	27.10
MCMTOPMTW-3-r103.txt	4860	4690	4562	31.00
MCMTOPMTW-3-r104.txt	6307	6146.2	5974	14.40
MCMTOPMTW-3-r105.txt	5368	5234.1	5087	11.60
MCMTOPMTW-3-r106.txt	7594	7399.7	7071	12.70
MCMTOPMTW-3-r107.txt	11880	11470.8	11225	11.30
MCMTOPMTW-3-r108.txt	13617	13348.2	13145	12.80
MCMTOPMTW-3-r109.txt	12099	11644.2	10433	11.20
MCMTOPMTW-3-r110.txt	15401	15081	14727	16.90
MCMTOPMTW-3-r111.txt	15749	15462.2	15082	14.20
MCMTOPMTW-3-r112.txt	16529	16037.6	15438	13.60
MCMTOPMTW-3-rc101.txt	1455	1382.2	1349	34.50
MCMTOPMTW-3-rc102.txt	2700	2648	2578	29.20
MCMTOPMTW-3-rc103.txt	3855	3785.3	3744	25.60
MCMTOPMTW-3-rc104.txt	6435	6313.6	6206	12.90
MCMTOPMTW-3-rc105.txt	6031	5851.6	5706	14.10
MCMTOPMTW-3-rc106.txt	8423	8069.1	7691	9.80
MCMTOPMTW-3-rc107.txt	9692	9377.1	9090	10.50
MCMTOPMTW-3-rc108.txt	12722	12213.8	11616	11.30
MCMTOPMTW-3-pr01.txt	1948	1932.7	1912	25.60
MCMTOPMTW-3-pr02.txt	4366	4272.5	4212	51.10
MCMTOPMTW-3-pr03.txt	6074	5905.4	5712	48.70
MCMTOPMTW-3-pr04.txt	10720	10371.1	9847	47.40
MCMTOPMTW-3-pr05.txt	15957	15145.5	14614	70.40
MCMTOPMTW-3-pr07.txt	13853	13547	12873	11.70
MCMTOPMTW-3-pr08.txt	20796	19983.7	19133	28.10
MCMTOPMTW-3-pr09.txt	29055	27835	25251	57.20
MCMTOPMTW-4-c101.txt	2430	2396	2376	52.60
MCMTOPMTW-4-c102.txt	4776	4725.6	4656	49.20
MCMTOPMTW-4-c103.txt	6919	6865.9	6811	21.30

Instance	Best	Average	Worst	Average time (s)
MCMTOPMTW-4-c104.txt	9049	8946.8	8865	22.30
MCMTOPMTW-4-c105.txt	10163	9976.8	9863	14.50
MCMTOPMTW-4-c106.txt	12957	12693.5	12373	12.20
MCMTOPMTW-4-c107.txt	17454	17307.8	16982	13.10
MCMTOPMTW-4-c108.txt	19345	18753.6	18483	15.90
MCMTOPMTW-4-c109.txt	24195	23850.6	23666	21.40
MCMTOPMTW-4-r101.txt	1380	1312.5	1235	29.70
MCMTOPMTW-4-r102.txt	3496	3429.5	3380	53.40
MCMTOPMTW-4-r103.txt	5022	4860	4785	30.50
MCMTOPMTW-4-r104.txt	7780	7592.6	7235	25.60
MCMTOPMTW-4-r105.txt	6864	6665.8	6465	10.70
MCMTOPMTW-4-r106.txt	12553	12197.6	11917	17.80
MCMTOPMTW-4-r107.txt	14251	13826.7	12845	19.20
MCMTOPMTW-4-r108.txt	17694	17117.7	16702	18.60
MCMTOPMTW-4-r109.txt	19469	18337.4	17019	16.30
MCMTOPMTW-4-r110.txt	21052	20513	19593	16.10
MCMTOPMTW-4-r111.txt	22201	21157.7	19871	18.70
MCMTOPMTW-4-r112.txt	21338	20687.8	19957	15.80
MCMTOPMTW-4-rc101.txt	1658	1587.3	1516	47.00
MCMTOPMTW-4-rc102.txt	3447	3292.3	3216	51.80
MCMTOPMTW-4-rc103.txt	5781	5586.4	5474	24.00
MCMTOPMTW-4-rc104.txt	8302	7947	7402	14.10
MCMTOPMTW-4-rc105.txt	7898	7593.9	7388	16.20
MCMTOPMTW-4-rc106.txt	10067	9778.2	9444	14.10
MCMTOPMTW-4-rc107.txt	14203	13828.3	13159	15.20
MCMTOPMTW-4-rc108.txt	18333	17628.3	16571	14.60
MCMTOPMTW-4-pr01.txt	2205	2194.1	2188	38.60
MCMTOPMTW-4-pr02.txt	5485	5288.9	5190	49.70
MCMTOPMTW-4-pr03.txt	8824	8492.4	8247	61.60
MCMTOPMTW-4-pr04.txt	12619	12390.2	12030	76.70
MCMTOPMTW-4-pr05.txt	20234	19469.5	18887	105.90
MCMTOPMTW-4-pr07.txt	18103	17759.7	17405	15.60
MCMTOPMTW-4-pr08.txt	28011	27118.7	26034	39.10
MCMTOPMTW-4-pr09.txt	35541	33557.7	31119	88.80

Bibliography

[1] E. H. L. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[2] L. Ahmedi, K. Rrmoku, and K. Sylejmani. Tourist tour planning supported by social network analysis. In *SocialInformatics*, pages 295–303, 2012,.

[3] H. Alshabib, O. F Rana, and A. Shaikh Ali. Deriving ratings through social network structures. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE, 2006.

[4] C. Archetti, A. Hertz, and M. Grazia Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2007.

[5] J. Baldzer, S. Boll, Klante P., J. Krösche, J. Meyer, N. Rump, A. Scherp, and H. Appelrath. Location-Aware Mobile Multimedia Applications on the Niccimon Platform. In *Braunschweiger Symposium – Informationssysteme für mobile Anwendungen*, 2004.

[6] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 166–174. ACM, 2004.

[7] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[8] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *INFORMS Journal on Computing*, 6(2):126–140, 1994.

[9] S. Bergamaschi, D. Beneventano, F. Guerra, and M. Vincini. Building a tourism information provider with the MOMIS system. *Information Technology and Tourism, Applications, Methodologies, Techniques, Hannes Werthner (Ed.)*, 1(1):15–31, 1998.

[10] E. Berscheid. The greening of relationship science. *American Psychologist*, 54(4):260, 1999.

[11] J. Blazewicza, E. Peschb, M. Sternaa, and F. Wernerc. Meta-heuristic Approaches for the Two-Machine Flow-Shop Problem with Weighted Late Work Criterion and Common Due Date. *Computers and Operations Research*, 35(2):574–599, 2008.

[12] H. Bouly, D. C. Dang, and A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70, 2010.

[13] S. Boussier, D. Feillet, and M. Gendreau. An exact algorithm for team orienteering problems. *4or*, 5(3):211–230, 2007.

[14] D. Buhalis and R. Law. Progress in information technology and tourism management: 20 years on and 10 years after the internet—the state of etourism research. *Tourism management*, 29(4):609–623, 2008.

[15] R. Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[16] S. E. Butt and T. M. Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, 21(1):101–111, 1994.

[17] S. E. Butt and D. M. Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4):427–441, 1999.

[18] T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[19] I. Chao, B. L. Golden, and E. A. Wasil. Theory and methodology—the team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

[20] I. Chao, B. L. Golden, E. A. Wasil, et al. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

[21] I. Chao, B. L. Golden, E. A. Wasil, et al. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

[22] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms (TALG)*, 8(3):23, 2012.

[23] H. Chun, H. Kwak, Y. H. Eom, Y. Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 57–70. ACM, 2008.

[24] A. Colorni, M. Dorigo, V. Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[25] J. F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[26] M. Crampes, J. de Oliveira-Kumar, S. Ranwez, and J. Villerd. Visualizing social photos on a hasse diagram for eliciting relations and indexing new photos. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):985–992, 2009.

[27] D. C. Dang, R. N. Guibadj, and A. Moukrim. A pso-based memetic algorithm for the team orienteering problem. In *Applications of Evolutionary Computation*, pages 471–480. Springer, 2011.

[28] A. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3):231–252, 1993.

[29] A. Divsalar, P. Vansteenwegen, and D. Cattrysse. A memetic algorithm for the orienteering problem with intermediate facilities. *status: published*, 2013.

[30] A. Divsalar, P. Vansteenwegen, and D. Cattrysse. A variable neighborhood search method for the orienteering problem with hotel selection. *International Journal of Production Economics*, 2013.

[31] D. Easley and J. Kleinberg. *Networks, crowds, and markets*, volume 8. Cambridge Univ Press, 2010.

[32] H. Ebel, L. I. Mielsch, and S. Bornholdt. Scale-free topology of e-mail networks. *arXiv preprint cond-mat/0201476*, 2002.

[33] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[34] R. Eberhart, P. Simpson, and R. Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., 1996.

[35] A. Ebner. TIS - Tirol Informations System - Die Konsequenz einer Idee. *Tourismus als Informationsgesellschaft. W. Schertler (eds.)*, Ueberreuter, 1994 (in German).

[36] C. Emmanouilidis, R. A. Koutsiamanis, and A. Tasidou. Mobile guides: taxonomy of architectures, context awareness, technologies and applications. *Journal of Network and Computer Applications*, 36(1):103–125, 2013.

[37] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[38] M. Fischetti, J. J. S. Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[39] F. V. Fomin and A. Lingas. Approximation algorithms for time-dependent orienteering. *Information Processing Letters*, 83(2):57–62, 2002.

[40] G. N. Frederickson and B. Wittman. Approximation algorithms for the traveling repairman and speeding deliveryman problems. *Algorithmica*, 62(3-4):1198–1221, 2012.

[41] L. M. Gambardella, E. Taillard, and G. Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New ideas in optimization*. Citeseer, 1999.

[42] A. Garcia, O. Arbelaitz, P. Vansteenwegen, W. Souffriau, and M. T. Linaza. Hybrid approach for the public transportation time dependent orienteering problem with time windows. In *Hybrid Artificial Intelligence Systems*, pages 151–158. Springer, 2010.

[43] A. Garcia, P. Vansteenwegen, W. Souffriau, O. Arbelaitz, and M. Linaza. Solving multi constrained team orienteering problems to generate tourist routes. *status: published*, 2009.

[44] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou. Mobile recommender systems in tourism. *Journal of Network and Computer Applications*, 2013.

[45] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and Y. Tasoulas. *A Survey on Algorithmic Approaches for Solving Tourist Trip Design Problems*, 2012.

[46] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and Y. Tasoulas. Cluster-based heuristics for the team orienteering problem with time windows. In *Experimental Algorithms*, pages 390–401. Springer, 2013.

[47] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.

[48] Z. W. Geem, C. L. Tseng, and Y. Park. Harmony search for generalized orienteering problem: best touring in china. In *Advances in natural computation*, pages 741–750. Springer, 2005.

[49] M. Gendreau, A. Hertz, and G. Laporte. A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, 40(10):1276–1290, 1994.

[50] M. Gendreau, G. Laporte, and F. Semet. Solving an ambulance location model by tabu search. *Location Science*, 5(2):75–88, 1997.

[51] M. Gendreau, G. Laporte, and F. Semet. A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32(4):263–273, 1998.

[52] M. Gendreau, G. Laporte, and F. Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2):539–545, 1998.

[53] F. Glover. Tabu Search - Part 1. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[54] F. Glover. Tabu Search - Part 2. *ORSA Journal on Computing*, 2(1):4–32, 1989.

[55] F. Glover. *Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem*. Graduate School of Business, University of Colorado, 1991.

[56] F. Glover and C. McMillan. The general employee scheduling problem. An integration of MS and AI. *Computers and Operations Research*, 13(5):563–573, 1986.

[57] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[58] B. L. Golden, Q. Wang, and L. Liu. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics (NRL)*, 35(3):359–366, 1988.

[59] U. Gretzel. Intelligent systems in tourism: A social science perspective. *Annals of Tourism Research*, 38(3):757–779, 2011.

[60] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, pages 53–60. ACM, 2009.

[61] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 607–615. LAWRENCE ERLBAUM ASSOCIATES LTD, 1995.

[62] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[63] H. Hochmair and G. Navratil. Computation of scenic routes in street networks. In *Geospatial Crossroads@ GI_Forum'08: Proceedings of the Geoinformatics Forum Salzburg*, pages 124–133, 2008.

[64] Q. Hu and A. Lim. An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 2013.

[65] R. James. *Long Term Memory Strategies for Solving the Early/Tardy Scheduling Problem*.

[66] M. G. Kantor and M. B. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, pages 629–635, 1992.

[67] L. Ke, C. Archetti, and Z. Feng. Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, 54(3):648–665, 2008.

[68] D. A. Kenny and L. La Voie. The social relations model. *Advances in experimental social psychology*, 18:142–182, 1984.

[69] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[70] P. Klante, J. Krösche, and S. Boll. Accessights - a multimodal location-aware mobile tourist information system. In *International Conference on Computers Helping People with Special Needs (ICCHP)*, pages 187–294. Springer-Verlag, 2004.

[71] J. Knox. Tabu Search Performance on the Symmetric Traveling Salesman Problem. *Computers and Operations Research*, 21(8):867–876, 1994.

[72] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[73] N. Labadie, R. Mansini, J. Melechovskỳ, and R. Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European journal of operational research*, 220(1):15–27, 2012.

[74] N. Labadie, J. Melechovskỳ, and Roberto W. Calvo. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics*, 17(6):729–753, 2011.

[75] M. Laguna, J. P. Kelly, J. L. Gonzfilez-Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82(1):176–189, 1995.

[76] C. Lamsfus, C. Grün, A. Alzua-Sorzabal, and H. Werthner. Context-based matchmaking to enhance tourists' experiences. *Novatica*, 11:17–23, 2010.

[77] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete applied mathematics*, 26(2):193–207, 1990.

[78] A. Likas, N. Vlassis, and J. J Verbeek. The global< i> k</i>-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[79] S. W. Lin and V. F. Yu. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94–107, 2012.

[80] S. Lloyd. Least squares quantization in pcm. In *Unpublished Bell Lab. Tech. Note*. portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, 1957.

[81] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[82] H. R. Lourenço, O. C. Martin, and T. Stutzle. Iterated local search. *arXiv preprint math/0102188*, 2001.

[83] R. Mansini, M. Pelizzari, and R. Wolfer. A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows. Technical report, Technical Report RT 2006-02-52, University of Brescia, Italy, 2008.

[84] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 1995.

[85] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[86] R. Montemanni and L. M. Gambardella. An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34:287–306, 2009.

[87] R. Montemanni, D. Weyland, and L. M. Gambardella. An enhanced ant colony system for the team orienteering problem with time windows. In *Computer Science and Society (ISCCS), 2011 International Symposium on*, pages 381–384. IEEE, 2011.

[88] J. M. Montoya and R. V. Solé. Small world patterns in food webs. *Journal of theoretical biology*, 214(3):405–412, 2002.

[89] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[90] G. Mota, M. Abreu, A. Quintas, J. Ferreira, L. S. Dias, G. A. B. Pereira, and J. A. Oliveira. A genetic algorithm for the topdtw at operating rooms. In *Computational Science and Its Applications–ICCSA 2013*, pages 304–317. Springer, 2013.

[91] Edited by Wassim Jaziri Multiple authors. *Local Search Techniques: Focus on Tabu Search*. I-Tech Education and Publishing, 2008.

[92] N. Musliu. *Intelligent search methods for workforce scheduling: new ideas and practical applications*. PhD thesis, Technische Universität Wien, 2001.

[93] World Tourism Organization. *UNWTO technical manual: Collection of Tourism Expenditure Statistics*, 1995.

[94] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[95] A. Pashtan, R. Blattler, A. Heusser, and P. Scheuermann. CATIS: A Context-Aware Tourist Information System. In *Proceedings of the 4th International Workshop of Mobile Computing*, 2003.

[96] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[97] B. Pröll and W. Retschitzegger. Discovering Next Generation Tourism Information Systems: A Tour on TIScover. *Journal of Travel Research*, 39(2):182–191, 2000.

[98] B. Pröll, W. Retschitzegger, R.R. Wagner, and A. Ebner. Beyond Traditional Tourism Information Systems - The Web-Based Approach TIScover. *Information Technology and Tourism, Applications, Methodologies, Techniques, Hannes Werthner (Ed.)*, 7(3/4):221–238, 2004.

[99] R. Ramesh and Kathleen M. Brown. An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research*, 18(2):151–165, 1991.

[100] R. Ramesh, Y. S. Yoon, and M. H. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2):155–165, 1992.

[101] I. Rechenberg. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

[102] C. R. Reeves. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

[103] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[104] F. Ricci. Travel recommender systems. *IEEE Intelligent Systems*, 17(6):55–57, 2002.

[105] F. Ricci. Mobile recommender systems. *Information Technology & Tourism*, 12(3):205–231, 2010.

[106] G. Righini and M. Salani. Dynamic programming for the orienteering problem with time windows. 2006.

[107] G. Righini and M. Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203, 2009.

[108] Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

[109] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[110] M. Schilde, K. F. Doerner, R. F. Hartl, and G. Kiechle. Metaheuristics for the bi-objective orienteering problem. *Swarm Intelligence*, 3(3):179–201, 2009.

[111] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

[112] D. Y. Sha and C. Y. Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791–808, 2006.

[113] P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[114] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer, 1998.

[115] J. Silberholz and B. Golden. The effective application of a new approach to the generalized orienteering problem. *Journal of Heuristics*, 16(3):393–415, 2010.

[116] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[117] W. Souffriau and P. Vansteenwegen. Tourist trip planning functionalities: State–of–the–art and future. In *Current Trends in Web Engineering*, pages 474–485. Springer, 2010.

[118] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. A path relinking approach for the team orienteering problem. *Computers & Operations Research*, 37(11):1853–1859, 2010.

[119] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. The multi-constraint team orienteering problem with multiple time windows. *Transportation Science, Articles in Advance*, pages 1–11, 2011.

[120] R. H. Strotz. Myopia and inconsistency in dynamic utility maximization. *The Review of Economic Studies*, 23(3):165–180, 1955.

[121] Zomaya. A. Y. Subrata, R. and B. Landfeldt. Artificial life techniques for load balancing in computational grids. *Journal of Computer and System Sciences*, 73(8):1176–1190, 2007.

[122] C. S. Sung and H. W. Jin. A tabu-search-based heuristic for clustering. *Pattern Recognition*, 33(5):849–858, 2000.

[123] K. Sylejmani, J. Dorn, and N. Musliu. A tabu search approach for multi constrained team orienteering problem and its application in touristic trip planning. In *Hybrid Intelligent Systems (HIS), 2012 12th International Conference on*, pages 300–305. IEEE, 2012.

[124] K. Sylejmani, J. Dorn, and N. Musliu. Touristic trip planning: solo versus group traveling. In *PlanSIG 2012*, 2012.

[125] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186, 1997.

[126] H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32(6):1379–1407, 2005.

[127] D. Tapscott. *The digital economy: Promise and peril in the age of networked intelligence*, volume 1. McGraw-Hill New York, 1996.

[128] F. Tricoire, M. Romauch, K. F. Doerner, and R. F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010.

[129] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, pages 797–809, 1984.

[130] P. Vansteenwegen. *Planning in tourism and public transportation – attraction selection by means of a personalised electronic tourist guide and train transfer scheduling*. PhD thesis, Katholieke Universiteit Leuven, Centre for Industrial Management, Belgium, 2008.

[131] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127, 2009.

[132] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden. Metaheuristics for tourist trip planning. In *Metaheuristics in the service industry*, pages 15–31. Springer, 2009.

[133] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[134] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & operations research*, 36(12):3281–3290, 2009.

[135] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: an or opportunity. *OR Insight*, 20(3):21–27, 2007.

[136] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. *Proc., Practical Application of Constraint Technology (PACT'96), London*, pages 337–356, 1996.

[137] Q. Wang, X. Sun, B. L Golden, and J. Jia. Using artificial neural networks to solve the orienteering problem. *Annals of Operations Research*, 61(1):111–120, 1995.

[138] X. Wang, B. L. Golden, and E. A. Wasil. Using a genetic algorithm to solve the generalized orienteering problem. In *The vehicle routing problem: latest advances and new challenges*, pages 263–274. Springer, 2008.

[139] S. Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[140] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[141] H. Werthner and S. Klein. *Information technology and tourism: a challenging relationship*. Springer Verlag Wien, 1999.

[142] Z. Xiang and U. Gretzel. Role of social media in online travel information search. *Tourism management*, 31(2):179–188, 2010.

[143] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

Curriculum Vitae

Kadri Sylejmani was born on May 10, 1977 in Podujeva, Republic of Kosova. He received his secondary school diploma in 1995 in his home town. He finished his diploma studies in 2004 in the field of computer engineering and telecommunications at Electro-technical Faculty, University of Prishtina, Kosova. In 2010, he got a master degree in computer science, in an interdisciplinary study program that was organized by University of Prishtina in cooperation with Universite de La Rochelle, France and Institute of Technology Carlow, Ireland. In the meantime, in 2008 he started his PhD studies in the field of computer science at Vienna University of Technology.

His working experience began just after the war in Kosova, back in 2000, when he was employed as language assistant in the United Nations Mission in Kosova. After, completing his studies, in 2005, he got employed as university assistant at Department of Computer Engineering in Faculty Electrical and Computer Engineering, University of Prishtina, where he is still working. He teaches the courses of programming languages, algorithms and data structures and digital circuits. His research engagement includes application of optimization techniques in the domain of e-tourism.

In the meantime, in form of part time basis, he has been engaged in a number of professional projects that were developed within several private companies and NGOs in Kosova. His engagements have mainly been in the fields of software development, consultancy and training. During the year of 2012, he was engaged into two research projects, namely "Tourist Tour Planning and Social Network Analysis" and "Text to speech conversion – case of Albanian language", where the earlier one was within his PhD project.

Publications

K. Sylejmani, J. Dorn, N. Musliu, Tourist trip planning: solo versus group traveling, 30th Workshop of the UK Planning And Scheduling Special Interest Group, Middlesbrough, United Kingdom, December 2012.

K. Sylejmani, J. Dorn, N. Musliu, A Tabu Search approach for Multi Constrained Team Orienteering Problem and its application in touristic trip planning, 12th International Conference on Hybrid Intelligent Systems (HIS 2012), December 4-7, 2012, Pune, India, IEEE.

L. Ahmedi, K. Rrmoku, K. Sylejmani, Tourist Tour Planning Supported by Social Network Analysis, 2012 ASE International Conference on Social Informatics, December 14-16, 2012, Washington, IEEE.

K. Sylejmani, Optimizing tourist group trip planning problem, PhD workshop of ENTER 2012 organized by International Federation for Information Technology and Travel & Tourism (IFITT), Helsingborg, Sweden.

K. Sylejmani, A. Dika , A taboo search algorithm for tourist trip planning, Workshop of The Journal of Information Technology and Tourism – JITT, International Federation for IT and Travel & Tourism – IFITT, Queen Margaret University, Edinburgh, Scotland, November 2010.