A User-Aware Tour Proposal Framework using a Hybrid Optimization Approach

Matthias Joest
European Media Laboratory
Schloss-Wolfsbrunnenweg 33
Heidelberg, Germany
+49-6221-533-264
matthias.joest@eml.villa-bosch.de

Wolfgang Stille
Institute of Computer Science
Darmstadt University of Technology
Wilhelminenstrasse 7
Darmstadt, Germany
+49-6151-16-3358
stille@rbg.informatik.tu-darmstadt.de

ABSTRACT

This paper considers a context- and user-aware approach to the tour planning in Geographic Information Systems. We regard the following scenario: a person with specific preferences and a fixed amount of time is visiting a foreign place of interest. Therefore spatial as well as user specific information is modeled as a multimodally attributed digraph structure, where nodes represent the points of interest and arcs the user-optimal paths between them. The goal is to find a set of nodes and arcs representing a round trip that is best satisfying the priorities of the user. This results in an Enhanced Profitable Tour Problem (EPTP), which is - as a generalization of the well-known Traveling Salesman Problem (TSP) - an $\mathcal{NP}$-hard optimization problem. In this article, we introduce an efficient algorithm for this problem and discuss the modeling and implementation details.

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph Algorithms; H.4.m [**Infor– mation Systems**]: Miscellaneous

General Terms

Algorithms

Keywords

Profitable Tour Problem, Local Search Algorithms, User-Awareness

1. INTRODUCTION

Mobile computing, personal digital assistants, wireless networks have been the buzzwords in the public during the last years. Fed by the steadily growth of computational power, the miniaturization of devices and the increasing bandwidth in wireless networks, computers are more and more influencing people's daily lives.

In a couple of years from now, computing resources will be available for everyone, anytime, at any need. One major challenge, which is posed to computer scientists by this process, is the demand of intelligent strategies that allow the users to personally benefit from these mobile systems. Within the last years, some research groups have focused on mobile tourist information systems as a predestinate test bed for mobile context-aware and ubiquitous computing. The *Deep Map* project (see [11] and [12]) at the European Media Laboratory fits into that framework. In contrast to other approaches like *CyberGuide* [2] or *GUIDE* [5], Deep Map aims for multi-modality by means of speech recognition and generation as well as the interaction with a graphical user interface. Deep Map focuses on a context-aware, mobile tourist information system, which incorporates a wide variety of information sources. Deep Map is based on a distributed multi-agent platform based on FIPA-OS[1]. A major requirement on a tourist information system is the guidance for the users. In the following, we will introduce a user-adaptive tour proposal framework based on a hybridization of local search algorithms used in combinatorial optimization. Therefore, all information that is relevant for a tour calculation is represented by a graph, in which the mathematical model of the problem can be stated. After reducing the graph size in order to save computing time, two heuristics are applied: the first one generates a feasible start solution; the second one improves that solution successively.

2. THE TOUR PROPOSAL FRAMEWORK

2.1 The Agent Platform

The Deep Map system consists of several agents for the various tasks within a tourist scenario like dialog management with the user or the presentation planning. In this paper, we only focus on agents, which are directly involved in the tour proposal process. As shown in Figure 1, the *Tour Agent (TA)* essentially consists of two components: the core piece is the *Tour Server (TS)*, that performs the main functionality, i.e. the generation of a best possible solution to the *Enhanced Profitable Tour Problem* (described in section 3). Thereon, a *Tour Client (TC)* is situated, which is re-

[1]Foundation for Intelligent Physical Agents, www.fipa.org

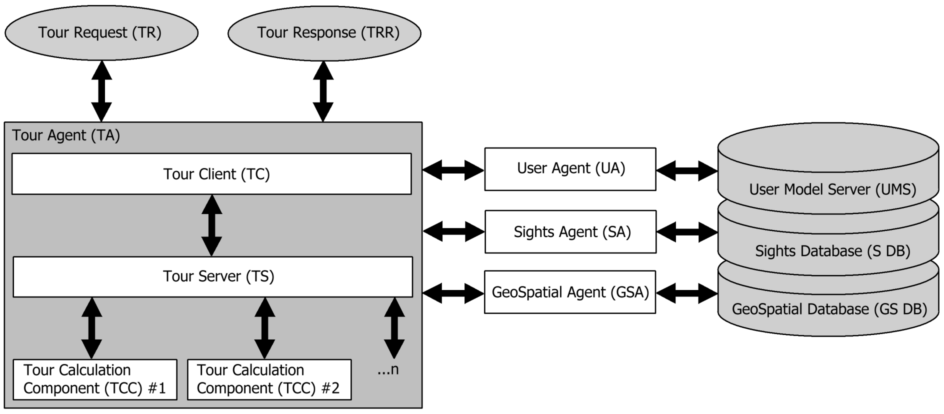


Figure 1: Agents involved in the tour proposal process

sponsible for handling the communication within the agent platform: it receives the tour request, entreats itself several agents to provide the information needed, preprocesses this data and mandates the TS to calculate the tour. The TS - listening at a specific port - receives the request and generates a new *Tour Calculation Component (TCC)*, which consists of a separate process. The communication between this process and the appropriate TC is then routed to another vacant port, so the server can accept further requests in the meantime. That way, different requests can be calculated simultaneously (e.g. in our case on a multi-processor machine).
When a new tour computation request is received, first the client collects the data needed for the tour computation. That is primarily spatial data in terms of a route network, which is necessary to generate a graph structure. This kind of data is handled by a *GeoSpatial Agent (GSA)*. The GSA manages all spatial data stored in a relational database with a spatial extension. It is able to execute extensive spatial queries on geographic data. Furthermore the client needs to have information about the points of interest. Within the Deep Map system, this content is governed by a *Sights Agent (SA)*. Points of interest are - in our case - sights in the first place, but surely not limited to that: restaurants, hotels, recreation areas, almost any kind of location is imaginable. Each point of interest is classified by the association with a profile vector describing its characteristic traits. Finally, we need information about the user preferences: Each user is associated with an individual *User Agent (UA)*, which manages his user preferences. These preferences are extracted from various sources (e.g. interaction with the graphical user interface or the speech interface) and are stored in a *User Model Server (UMS)*. The UMS allows statistical operations on this data in order to determine a correspondence to the points of interest by their profile vectors.

2.2 Generation and Weighting of the Graph

The data structure, on which the tour planning problem can be modeled, is naturally a graph composed of nodes and arcs. To obtain a graph structure from the given spatial data, a node is generated at each intersection of the route network. Any two nodes are generally connected by an arc, which represents one or more route fragments in the spatial database. Additionally, arcs are weighted with the distance between the nodes they interconnect, and nodes are associated with the time that is required to visit them. Moreover, we want to attach priority values to the nodes and arcs. How these values are gained is described as follows: Every user profile is characterized by a set of notes, each of them associated with an integer value expressing its intensity. For a very simple example, user A is very interested in Gothic churches, fairly in Renaissance buildings and not at all in Modern Art, his profile vector $(\mathrm{gothic}, \mathrm{renai}, \mathrm{modart})^T$ would look like $(100, 50, 0)^T$, if 100 is the maximal value. In the same manner, the points of interest in the database are characterized. Let's say $\vec{s}_i$ is the profile vector of sight i, and $\vec{a}_k$ represents the user profile of user k, the weighted scalar product $\omega < \vec{s}_i, \vec{a}_k >$ determines how much user k is interested in visiting point i.
Edges are weighted in the same way: let $\vec{s}_{ij}$ be the profile vector of arc (i, j), and $\vec{b}_k$ the profile vector of user k. Thus, we get an arc price $\kappa < \vec{s}_{ij}, \vec{b}_k >$ for each arc (i, j) and each user k. Compared to the nodes, arcs are constitutionally characterized by a different set of attributes, e.g. their attractiveness, vista, steepness, noise, dirt, etc. Therefore the weighting factors ω and κ are introduced to adjust the ratio between node and arc prizes, e.g. one can choose ω and κ the way the average arc prize equals the average node prize. Considering a sight-seeing tour by car or public transport, more importance would typically be attached to the node prizes.
Further initial parameters needed to specify a tour request are a start location, the total time the visitor is willing to spend on the tour, and the speed the person is traveling at. If all of this data has been received and processed by the Tour Client, it sends a tour calculation request to the Tour Server. Depending on whether the user requests his first tour, the Tour Server creates a new instance of a Tour Calculation Component, which starts calculating the tour. Otherwise, if the user has already requested a tour some

time ago and there are no significant changes in his preferences and thus no significant changes in the graph, the initial data processing and a preprocessing step (see section 4.1) can be left out to save time.
The TCC then computes the desired tour. Its output is an ordered list of attributed nodes and arcs, the sum of the prizes of the nodes and arcs visited and the total amount of time required for the tour. These results are passed to the TC, which requests the GeoSpatial Agent to store the tour in the spatial database and to reconstruct the real path geometries from the related arc IDs.

3. THE ENHANCED PROFITABLE TOUR PROBLEM

More formally, the problem to calculate a user-optimal tour can be stated mathematically as the *Enhanced Profitable Tour Problem (EPTP)*. This is a variation of the *Prize-Collecting Traveling Salesman Problem (PCTSP)*, which was first mentioned by Balas[3]. The EPTP is defined as follows: A weighted digraph $G = (V, A)$ is given, where V is the set of nodes (of size $|V| =: n$) and A is the set of arcs (of size $|A| =: m$). Without loss of generality, we assume that $V = \{1, \dots, n\}$ and that node 1 is the start node of the tour. Every arc $(i, j) \in A$ is associated with a prize p_{ij} and a time t_{ij}, and every node $i \in V$ is associated with a prize p_i and a time t_i.
The EPTP consists in finding a particular cycle $\chi = (V_\chi, A_\chi)$, $V_\chi \subseteq V, A_\chi \subseteq A$, with $1 \in V_\chi$, that maximizes the sum of the prizes of the nodes and arcs it is composed of, whereas each node is visited at most once. The total amount of time required for that cycle must not exceed a given $t_{\max}$. We introduce the binary variables $x_{ij} \in \{0, 1\}$ and $y_i \in \{0, 1\}$. Assuming $x_{ij} = 1$ if arc $(i, j) \in A_\chi$ and 0 otherwise, and $y_i = 1$ if node $i \in V_\chi$ and 0 otherwise, the problem can be stated as follows:

$$\text{maximize} \quad \sum_{i\in V} p_i y_i \quad + \quad \sum_{i\in V}\sum_{j\in V\setminus\{i\}} p_{ij}x_{ij} \qquad (1)$$

$$\text{subject to} \quad \sum_{j\in V\setminus\{i\}} x_{ij} - y_i \quad = \quad 0, \qquad \forall i \in V, \qquad (2)$$

$$\sum_{i\in V\setminus\{j\}} x_{ij} - y_j \quad = \quad 0, \qquad \forall j \in V, \qquad (3)$$

$$y_1 \quad = \quad 1, \qquad (4)$$

$$\sum_{i\in V}\sum_{j\in V\setminus\{i\}} t_{ij}x_{ij} + \sum_{i\in V} t_i y_i \quad \le \quad t_{\max}, \qquad (5)$$

$$\sum_{i\in S}\sum_{j\in V\setminus S} x_{ij} \quad \ge \quad y_h,$$
$$\forall S \subset V \quad : \quad 1 \in S, h \in V\setminus S, \qquad (6)$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A, \qquad (7)$$

$$y_i \in \{0, 1\}, \qquad \forall i \in V. \qquad (8)$$

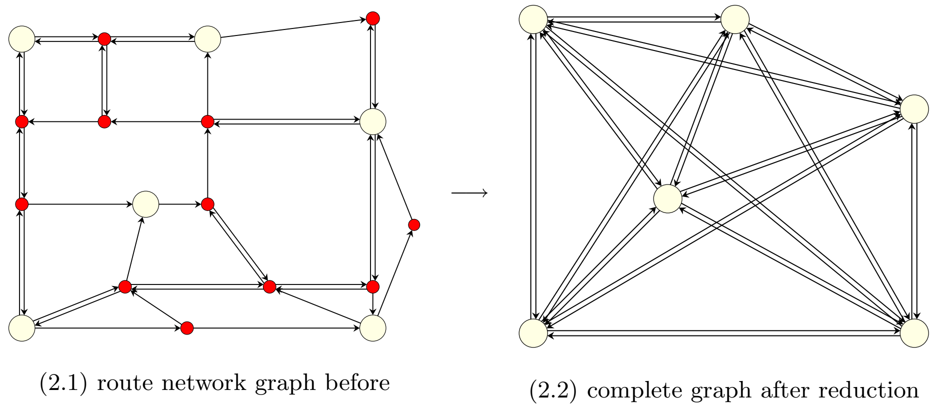
(1) is the objective function expressing the maximization of the prizes of the nodes and the arcs included in the cycle χ. (2) and (3) are the well-known assignment constraints making sure that every node in the cycle is connected with exactly one inbound and exactly one outbound arc. (4) determines the start node to be part of the cycle and (5) states the total time of the cycle to be less or equal than $t_{\max}$. (6) forces every node in the cycle to be connected to the start node. This is called the *sub-tour elimination constraint.* Together with (2) and (3), it provides that there is only one cycle. Finally, (7) and (8) guarantee the variables x_{ij} and y_i to be binary. Mathematically speaking, the expressions (1) to (8) define an *Integer Linear Program (ILP).* The ILP stated above is an $\mathcal{NP}$-hard optimization problem. This generally means one cannot expect an algorithm that finds an optimal solution in polynomial time (e.g. $\mathcal{O}(n^a)$, where n is the problem size and $a \in \mathbb{N}$).

4. THE ALGORITHM

In some cases, ILP can be solved to a proven approximation factor or even to optimality by sophisticated polynomial algorithms. Within the scope of this research, we have modified some solution methods from combinatorial optimization such as branch and bound [7], classic edge exchange operators [10] and more advanced local search strategies as presented in [1] in order to meet our specific needs: The resulting algorithms were not usable in most instances, primarily owing to performance issues and a bad solution quality. The problem is far more complex than the TSP, because the tour only consists of a subset of nodes. This subset has to be chosen during the algorithm. Furthermore, the generated graph gets very large in many cases. Our test scenario, the city of Heidelberg in Germany, is not very huge with its about 140,000 inhabitants. Nevertheless,the graph contains approximately 6,000 nodes and 15,000 arcs. This number is increasing due to the fact that new points of interest are added to the sights database. So even the modified quadratic PCTSP approach based on [9] resulted in computation times that were not acceptable for the on-the-fly computation of user-aware round trips, which was the main goal of this project. Thus, a new hybrid approximation algorithm was developed, whose functionality and advantages are described as follows.

4.1 Reduction of the Original Graph

As mentioned above, the generated graph may grow pretty large, depending on the size of the route network and the richness of additional information available. The computing time should remain in reasonable limits to ensure the user does not have to wait too long to get a tour upon request. For this reason, the graph should be kept small. The graph size can be reduced significantly, if the nodes that do not represent points of interest are eliminated as shown in Figure 2. This is obtained by calculating paths between all pairs of points of interest. In our case, we use a Dijkstra shortest path algorithm, that performs very well on real road networks (for a comparison see [16]). As we do not necessarily need the shortest paths, but the best possible ones concerning the user priorities, the original distances t_{ij} are modified by subtracting a certain amount of their prizes $\kappa < \vec{s}_{ij}, \vec{a}_k >$. Thereby, an arc grows virtually shorter, if it is associated with a high prize, and is that way rather chosen by the algorithm than an arc with a low or even negative prize. The algorithm is modified to that effect that the decision to admit an arc to be part of the path is motivated by the modified distance values, whereas the real length of the path is calculated and saved. As the result of this pro-


(2.1) route network graph before

(2.2) complete graph after reduction

Figure 2: Reduction of nodes and arcs of the original route network graph

cedure, we get a complete graph consisting only of points of interest and user-optimal paths between them (see Figure (2.2)). There is, of course, a trade-off between a decreasing number of nodes and an increasing number of arcs in the reduced graph as the resulting graph is complete. Since the complexity of the algorithm presented in this section solely depends on the number of nodes, this is not crucial at all. The reduction procedure has to take place only in the case a new user requests his first tour. All further tours can be calculated on the same reduced graph in order to save computing time.

4.2 Initial Tour Construction

Based on an idea in [13], we construct a feasible cycle by iteratively inserting the best possible node in each step. In our case, feasible means feasible with regard to our model in section 3. Anyway, as we always maintain the cycle structure during this process, we have to concentrate on equation (5): the given $t_{\max}$ must not be exceeded during the construction of the cycle. To choose the node s_{best} to be inserted in the next step, all nodes s that are not part of the current cycle χ are associated with a

$$\delta_{uv}(s) := \frac{\Delta p_{suv}}{\Delta t_{suv}}, \quad \Delta t_{suv} \neq 0,$$

where

$$\Delta t_{suv} := t_{us} + t_{sv} - t_{uv} + t_s$$

is the change regarding the time (resp. distance or cost), and

$$\Delta p_{suv} := p_{us} + p_{sv} - p_{uv} + p_s$$

is the change regarding the prize, if node s is inserted between the nodes u and v in the cycle χ. $\delta_{best}(s)$ quantifies the change in the ratio of the cycle prize and the cycle time, that is effected by inserting node s at its optimal position in the cycle maintaining feasibility of the cycle. The optimal position of a node s is determined as the position the node s reaches a maximal $\delta_{uv}(s)$ in the cycle and feasibility is kept. In the next step, the feasible node with the maximum $\delta_{best}(s)$ is inserted into the cycle. This is repeated as long as there are feasible candidates for insertion. Each insertion requires $\mathcal{O}(n^2)$ computations.

4.3 Tour Improvement

Starting with the feasible tour from the previous paragraph, the next step consists in improving its objective function value. 3-Opt and other classic edge exchange operators do not take an exchange of nodes into account, others in turn replace nodes and need many repairing arcs to maintain the cycle structure. Therefore, we enlarge the solution space by considering a set of exclusive nodes, whilst always maintaining the cycle structure in every pass. This kind of diversification is an advancement of an approach from [6] for the Prize-Collecting Traveling Salesman Problem. The tour improvement phase mainly consists of two important components: the *Extension Phase* and the *Collapse Phase*. These are alternated iteratively until there is no further improvement gained.

4.3.1 *Extension Phase*

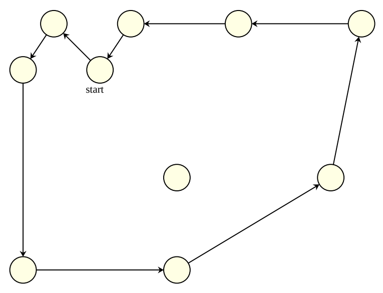
The Extension Phase enlarges the feasible cycle χ by dedicated nodes. In particular, for each node $j \in V \setminus V_\chi$ the *unitary gain* is given by

$$R_j = \max\ \delta_{uv}(j) = \max\Big\{\frac{\Delta p_{juv}}{\Delta t_{juv}}\ : \Delta t_{juv} > 0\Big\}. \quad (9)$$

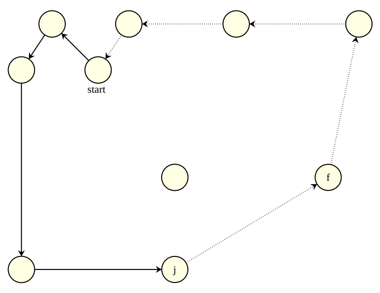
The *average gain* $\bar{R}$ for all nodes $i \in V \setminus V_\chi$ is defined as

$$\bar{R} = \sum_{i \in V \setminus V_\chi} \frac{R_i}{|V \setminus V_\chi|}. \quad (10)$$

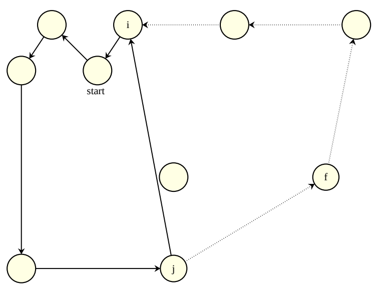
Then, all the nodes $j \in V \setminus V_\chi$ such that $R_j > \bar{R}$ are possibly added to χ. Whenever a node j with $R_j > \bar{R}$ is added to the cycle χ, the value R_k for all $k \in V \setminus V_\chi$ is updated accordingly. Thus, nodes previously excluded from consideration may now be taken into account. The average gain $\bar{R}$ is not changed during this sequence. Additionally, we restrict the extension phase by extending the cycle χ by maximal $\lfloor \alpha |\chi| \rfloor$ nodes, where $0 < \alpha < \frac{|V \setminus V_\chi|}{|V_\chi|}$. Whilst this modus operandi, it is necessary to add always the node with the highest unitary gain. This prevents the algorithm from adding too many nodes with a comparatively bad unitary gain to the current cycle. Hence - as validated in extensive test runs - the computing time of the collapse phase can be strongly reduced without disregarding possibly good solutions. The Extension Phase needs $\mathcal{O}(n^2)$ computations.



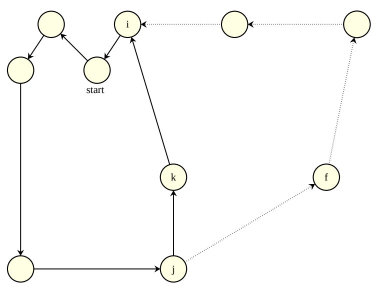
(3.1) the cycle after extension



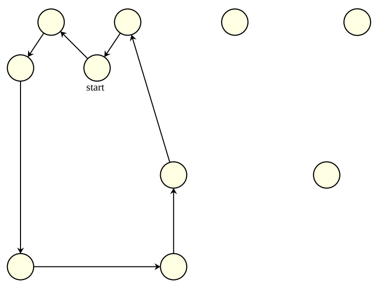
(3.2) found infeasibility point f



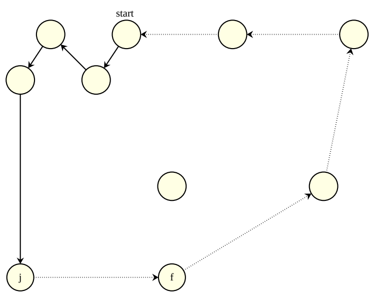
(3.3) close cycle with collapse point i



(3.4) insert additional collapse point k



(3.5) best cycle so far ? $\rightarrow$ save



(3.6) search for new infeasibility point from new start point, continue with (3.3)

Figure 3: Collapse Phase in the Extension/Collapse tour improvement algorithm

4.3.2 *Collapse Phase*

The Extension Phase tries to improve the overall profit of the current cycle. Feasibility of the cycle, however, is not maintained. The Collapse Phase focuses on the reduction of the expanded cycle intending to make the cycle feasible again. Therefore, starting at node 1, a path within the cycle is constructed unto an infeasibility point f (see Figure 3.2). The infeasibility point f is reached if the path gets infeasible, i.e. if the time (or length) of the path exceeds $t_{\max}$. We exclude f and the appendant arc (j, f) from this path and try to close the cycle by adding a *collapse point* i (Figure 3.3). If there is no possibility to close the cycle at all - even without adding a collapse point, we exclude the last node and the appropriate arc from the path and try again.
The collapse point i is exactly the point, that maximizes the ratio prize/time of the cycle without losing feasibility.
In our application, the planning of user-optimal tours, it is desirable that the overall tour length gets very close to the given $t_{\max}$. Therefore, we allow the insertion of more than one collapse point, again feasibility provided (Figure 3.4).
If the cycle constructed by the Collapse Phase is the best seen so far, it is saved as χ_{best} (Figure 3.5). Then the Collapse Phase is re-run from scratch, now constructing a path starting at the predecessor node of node 1 in the extended cycle (Figure 3.6). These restarts are done as long as node 1 is part of the feasible path, since it is the start location of the tour, and so must be contained in the cycle. The Collapse Phase needs $\mathcal{O}(n^2)$ computations.

4.3.3 *Extension/Collapse Driver*

As stated above, the Extension and Collapse Phases are applied time and again in an alternating sequence. This is realized by a driver consisting of two nested loops: at the beginning, we set $\chi_{best} := \chi$, which is the cycle generated by the insertion heuristic from section 4.2, and initialize the extension parameter α. That is, in the first extension step, the cycle is extended by maximal α times as many nodes as it contains before the extension. The inner loop consists of the Extension Phase followed by the Collapse Phase. Whenever a cycle is found with a better objective function value than the current χ_{best}, the current cycle χ is saved as the new χ_{best}. The two phases are repeated in turn, until there is no further improvement of the currently best cycle.
The outer loop restarts the inner loop with a steadily decreasing α as long as $\lfloor\alpha|\chi|\rfloor > 0$. Thus, the Extension Phase in the inner loop extends the current cycle by less and less nodes until there is at least one node the current cycle is extended by. The algorithm terminates, if $\lfloor\alpha|\chi|\rfloor = 0$. In our implementation, we set $\alpha := 1$ in the beginning and bisect it in every outer loop pass. As our experiments have shown, this proceeding provides a very good solution quality at a passable complexity. Depending on the network data, other configurations may be more reasonable.
As there is either an improvement of the cycle in every step or a bisection of the parameter α, the algorithm terminates after a certain number of iterations. Since both, the Extension and the Collapse Phase need $\mathcal{O}(n^2)$ computations, the algorithm has an overall complexity of $\mathcal{O}(n^2 \log_2 i)$, where i is the average number of nodes occurring in feasible cycles

during the algorithm.

5. COMPUTATIONAL EXPERIMENTS

Comprehensive computational experiments were performed on an SGI Origin 2000 on the network topology of the city of Heidelberg. After the reduction from section 4.1 its graph contains 750 nodes. To evaluate the performance of the algorithm, various tours were computed taking each node of the network as a start node for 37 tours of different length. We have chosen $t_{\max} \in \{120, 150, \ldots, 1200\}$ minutes for the tour length and thus get tours containing at an average from 6 up to 58 points of interest. As the computing time of both, the tour construction (section 4.2) and the tour improvement phase (section 4.3) depends on the number of nodes in the tour, we have obtained computing times for the tour construction ranging from $20ms$ to $1.7s$. For the Extension/Collapse tour improvement, the computing times range from $140ms$ to $20s$. The calculation time for an average day trip of 8 hours averaged out at $3.7s$. Therefore, this approach is very suitable for the on-the-fly computation of user-optimal tours.
To evaluate the solution quality, we compared our algorithm to the exact solution by a branch and bound approach based on the research of [7] and an approximative algorithm from Goemans and Williamson [9] (in the following abbreviated as GW). Since the computing times of both approaches are much higher (exponential in the worst case for the branch and bound approach, $\mathcal{O}(n^2 \log n)$ for the GW algorithm), we were forced to extract smaller subgraphs from the network: the exact algorithm was fed by subgraphs containing 20 resp. 25 nodes, the GW algorithm was run with graphs consisting of 50, 100 and 200 nodes.
In comparison to the branch and bound algorithm, the sum of the prizes of the tours provided by our algorithm averaged out at throughout over 90% of the exact solution. Compared to the GW algorithm, our approach performed better in most of the cases. Solely on the smallest graph (50 nodes), the solution quality provided by the GW algorithm was slighty better. On the graphs with 100 respectively 200 nodes, the sum of the prizes of tours computed by the Extension / Collapse algorithm was approximately $5 - 15\%$ higher. It was noticeable that the larger the graph got, the higher the gap between the solution quality of our approach and the GW algorithm grew. The detailed computational results of this research can be found in [15].
To manually verify the solutions calculated by the algorithm, a visualization component was incorporated. Figure 4 shows such a visualization of a tour visiting the castle area and parts of the Old Town of Heidelberg.

6. CONCLUSION AND OUTLOOK

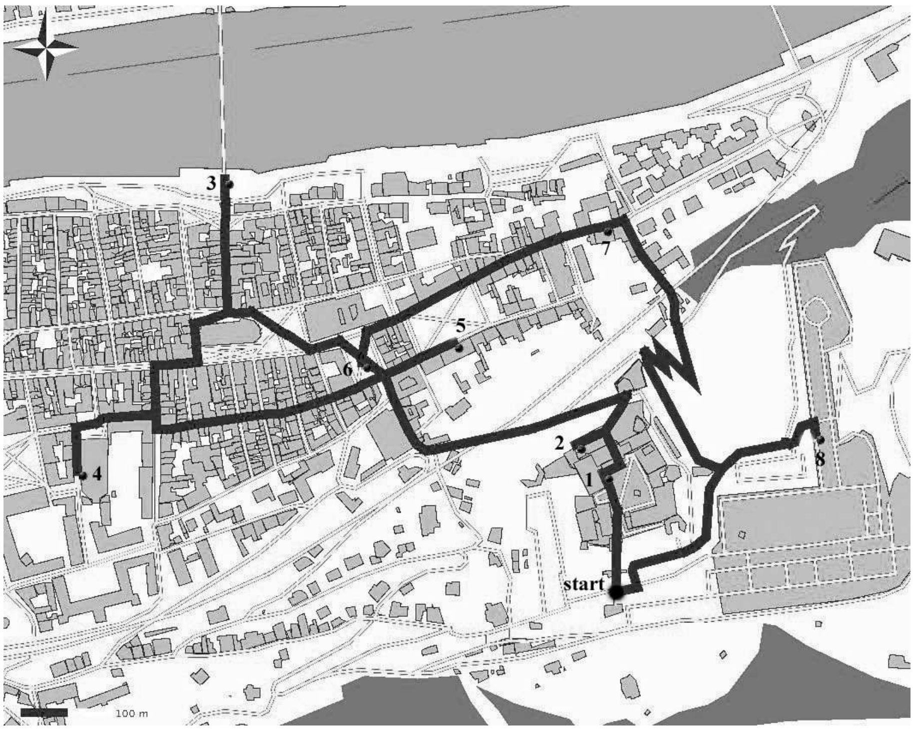
With the tour planning approach presented in this work, a very powerful tool for the computation of user-optimal tours has been developed. On the one hand, the quality of the results is convincing, on the other hand, the calculation time is very moderate compared to other algorithms. Furthermore, the framework has an open architecture and thus is extensible concerning future demands. For example, it is easy to modify the algorithm to calculate tours with specified start and end nodes by interconnecting them by a dummy arc of length 0 and then calculating the tour as a round trip. It is also possible to specify points of interest that should be visited by setting their prizes to a maximum value. The algorithm then will choose them with a very high probability.
As more and more information is collected electronically, future enhancements will be the attribution of the road network with further and more distinguished attributes like street condition, house facades or weather exposition. This will allow a more and more sophisticated calculation of user-optimal paths. Furthermore, the sights database is growing steadily, so one could think about a dynamic modulation of the network size, depending on the start point and the overall tour time. Therefore, the GeoSpatial Agent would calculate a buffer region around the start point. The size of this region depends on the given $t_{\max}$ and the means of transportation. This would lead to a smaller graph and faster calculation times, as the solution of hard optimization problems is a time-critical process anyway.
As some kind of information is collected by agents during the tour, a kind of online optimization is desirable, where additional information is dynamically involved in the tour optimization process. For example, a weather agent that manages the actual weather forecast would be useful in the temperate zone.
Within the first prototype of the Deep Map system, the Tour Agent regards only one transportation mode per tour. It is conceivable that there are several graphs for different kinds of transportation. That poses the question on how to incorporate time windows and timetable-based optimization approaches into the algorithm.

7. ACKNOWLEDGMENTS

This paper is based on research carried out within the German Federal Ministry for Education and Research (BMBF) Project SmartKom. We would like to thank Prof. Dr. Reinelt, Prof. Dr. Weihe and Dr. Malaka for their support. Thanks to the anonymous referees for their helpful comments.

8. REFERENCES

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*, chapter 9: Vehicle Routing: Modern Heuristics, pages 311–336. Wiley, Chichester, 1997.
[2] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. In *ACM Wireless Networks, 3*, pages 421–433, 1997.
[3] E. Balas. The Prize Collecting Traveling Salesman Problem. *Networks*, 19:621–636, 1989.
[4] B. Cherkassky, A. Goldberg, and T. Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. In *Proceedings of the 5th Annual ACM SIAM Symposium on Discrete Algorithms*, pages 516–525, Arlington, VA, January 1994.
[5] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In *Proceedings of CHI'00, ACM Press*, Netherlands, 2000.
[6] M. Dell'Amico, F. Maffioli, and A. Sciomachen. A Lagrangian Heuristic for the Prize Collecting



#	Point of Interest	c	$\frac{p}{10}$	#	Point of Interest	c	$p/10$
1	Ladies Building	15	39	5	Academy of Sciences	15	90
2	Great Barrel	5	45	6	Corn Market	5	75
3	Carl-Theodor Statue	5	42	7	Villa Buhl	15	90
4	Jesuit Church	20	174	8	Castle Garden	25	210

Figure 4: Round trip computed by the Extension / Collapse algorithm starting at the castle and visiting the castle area and parts of the Old Town of Heidelberg

Travelling Salesman Problem. *Annals of Operations Research*, 81(1):289–305, 1998.

[7] M. Dell'Amico, F. Maffioli, and P. Värbrand. On Prize-Collecting Tours and the Asymmetric Travelling Salesman Problem. *International Transactions on Operational Research*, 81(1):289–305, 1995.

[8] M. Gendreau, A. Hertz, and G. Laporte. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, 40(6):1086–1094, 1992.

[9] M. Goemans and D. Williamson. A General Approximation Technique for Constrained Forest Problems. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 307–315, 1992.

[10] S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[11] R. Malaka, R. Porzel, A. Zipf, and V. Chandrasekhara. Integration of Smart Components for Building your Personal Mobile Guide. In *Proceedings of AIMS 2000. Workshop on Artificial Intelligence in Mobile Systems*, Berlin, 2000.

[12] R. Malaka and A. Zipf. Deep map - Challenging IT Research in the Framework of a Tourist Information System. In *Information and Communication Technologies in Tourism 2000. Proceedings of ENTER 2000, 7th. International Congress on Tourism and Communications Technologies in Tourism. Barcelona. Spain*, pages 15–27, 2000.

[13] J. Mittenthal and C. E. Noon. An Insert/Delete Heuristic for the Travelling Salesman Subset-Tour Problem with one Additional Constraint. *Journal of the Operational Research Society*, 43(3):277–283, 1995.

[14] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 1994.

[15] W. Stille. Lösungsverfahren für Prize-Collecting Traveling Salesman Subtour Probleme und ihre Anwendung auf die Tourenplanung im Touristeninformationssystem Deep Map. Master's thesis, University of Heidelberg, Heidelberg, March 2001.

[16] F. B. Zhan and C. E. Noon. Shortest Path Algorithms: An Evaluation using Real Road Networks. *Transportation Science*, 32:65–73, 1998.