An Arduino-based instrument for more intuitive expression of electronic music

Tina Wang

College of Engineering and Computer Science

The Australian National University

Canberra ACT 2000, Australia

u6076893@anu.edu.au

*Abstract*— Route navigation is a large and very relevant field of study in today’s complex world. While there exists much research and application concerning short-term route navigation (e.g. navigation from Location A to Location B on Google Maps), this is less true for both long-term routes and route generation. In this paper, we not only identify solutions for long-term route generation problems, but consider elements of customisation, and account for an existing schedule. Our goal is to provide a personalised route for individual users in a given timeframe.

We formulate this extended route navigation problem as a Selective Travelling Salesman Problem with both mandatory and selective cities. From this, integer linear programming (ILP) model is proposed for the problem and an algorithm using a combination of the Iterated Local Search (ILS) and Greedy Randomized Adaptive Search Procedure (GRASP) approaches is formulated.

*Keywords*—*electronic music,*

# Introduction

In this paper, we propose an algorithm that works around a user’s existing time schedule to provide further venue suggestions and increase productivity. Our approach will attempt to condense the enormous amount of data in the modern day to a fast, comprehensive route for the user. Essentially, the navigation software will make decisions when the user does not have the capacity to, and suggest personalised routes for the user. By taking into consideration the needs, requests and the schedule of the user in a Geographic Information Systems (GIS) context, the software would ideally return a route (or routes) from the entire geographical data set of the local area, something that the user is unlikely to have immediate access to. Furthermore, our application would attempt to create best possible suggestions for the user by prioritise for locations with the best ratings.

We would ideally devise an algorithm that could consider existing schedule and user preferences to create (an) optimised extended route(s) for the user. Since we cannot visit all desired venues due to time constraints and clashes, we seek a good comprise of optimisation – that is, maximising visits to the most/best places while minimising travel time/distance as much as possible. Furthermore, we would like to provide one or more route suggestions rather than a single rigid schedule that the user must follow. This still retains an amount of flexibility and autonomy on the user’s part. It is for this reason that the algorithm does not provide any optimisation for the *best* solution – however, it optimises for a solution that is *good enough*. Precisely, the algorithm will maximise productivity and enjoyment. Our goal is to make scheduling an easier task for the user through suggestion, not to control their entire schedule. Thus, finding an optimal solution is unnecessary.

This algorithm can be applied in an extended navigation app that tailors future recommendations to the user’s liking. This application will ask for essential details – the user’s prospective schedule, as well as both start and end times, and start and end destinations. The extended route can begin and end at the first and last items in the schedule respectively or can be distinct from these. User personalisation can be provided through optionally specifying extra details such as categories of interest (e.g. Relaxation, Historical, Sightseeing, Cafes) and traveller type (selection from Family, Couple, Solo, Business or Friends) after entering the essential details. Note that a venue may have more than one entry for both categories of interest and traveller type.

***Example 1****: If User A wanted to plan a day trip where they had class from 9-10AM, lunch with a friend from 1-3PM, and a meeting from 5-6PM, what would they do during the hours in the middle? The user is travelling alone by foot and has a priority on relaxation but would also like to visit a library for some short study.*

Example 1 is an example instance of the application where the start and end times are punctuated by schedule items:

* Schedule: “9:00-10:00, Class, Lecture Hall 1”, “13:00-15:00, Lunch with friend, Restaurant 1”, “17:00-18:00, Meeting, Seminar Room 1”
* Start time: 9:00
* End time: 18:00
* Start destination: Lecture Hall 1
* End destination: Seminar Room 1
* Categories of interest: Relaxation, Study
* Traveller type: Solo

***Example 1 potential solution****: The extended route would direct User A to study at a university library between 10AM-1PM (for short study), and take a walk through the nearby Botanical Gardens and dropping by a cafe from 3-5PM (for relaxation) while considering traffic and travel times.*

While this is a rather simple example, it could be used for any type of schedule for any day (or longer), and would arguably be more significant in an unfamiliar environment e.g. on vacation. Another example would be an employee on a business trip who would like to do some sightseeing outside of meeting times. However, even in a familiar environment, it could expand the user’s horizons to find new places in their local area that they were unaware of.

Under a schedule with events at periodic times, the average user would typically lose time on indecision, fall into a usual routine (more applicable in a familiar environment) or waste time on inefficient travel time between venues (more applicable in an unfamiliar environment). By handling the effort of creating a newly routed schedule, the application would both boost productivity for the user and provide them with more control over their schedule (through preference specifications). It can also be used to give variety to people’s lives in an otherwise mundane setting. Just like a typical navigation app, it would also encapsulate the usual transport mediums – by car, foot, bike or public transport.

We are hoping that this navigation project will improve people’s lives by providing simplified solutions to today’s complex data-immersed world. While data provides us with the luxury of choice, too much of it can result in indecision and inhibition to our lives. People can be more productive with their time in all aspects of life, and may even find someplace new from the personalised routes given by the system. Additionally, routes that minimise time and distance would save time and travel expenses (e.g. fuel costs, public transport fees) for the user.

Our proposal has much in common to the Traveling Salesman Problem, one of the most prominent problems in computer science: “Given a set of cities and a starting location, what is the shortest route the travelling salesman can take to visit all cities?” Although our problem varies in aspects, a prominent one being that not all venues in the complete set of venues are required to be visited, they share a common essence in distance minimisation.

The paper is structured as follows. In section 2 we discuss current navigation research and its limitations, and propose existing solutions for TSP-based problems. In section 3, we derive a mathematical model to describe our GIS problem. In section 4, we propose an algorithm using optimised insertion and deletion techniques. In section 5, we provide an evaluation of this algorithm and an overall discussion. Finally, in sections 6 and 7, we summarise the paper and discuss possibilities for future work.

# Related Work

Current solutions for GIS typically focus on improved spatial descriptions or short-term solutions (i.e. navigating from A to B). The most commonly used navigation software is Google Maps, which usually offers the shortest route or the fastest route (if they differ at all). Most research on the topic of real-world map navigation is focused on the optimisation of these short-term routes, including empirical algorithm evaluation [1] and shortcut detection [2]. Further research in this area considers the minimisation of cost factors representing dangerous geographical features such as uneven or dangerous terrain [3].

Distance minimisation between a set of specified points, where each point is only visited once, is essentially the NP-hard Travelling Salesman Problem (TSP). As the TSP is inherently a GIS problem, its applications are widely seen as real-world routing solutions. Additionally, since the TSP is NP-hard, most route navigation problems are also NP-hard. Current applications of the TSP involve the optimisation of problems ranging between computer wiring, vehicle routing, job-shop scheduling [4] and efficient municipal waste collection [5]. Although the project is focused on reducing uncertainty and indecision by filtering large amounts of data, we still want to maintain a feasible margin of choice for the user. By considering optimisation solutions for the Travelling Salesman problem, we can optimise our route.

Although these are extremely useful for navigating to a specific location, or interpreting information that has already been provided, they do not provide opportunities to give route or venue suggestions. Moreover, representation using the standard TSP would not suffice after considering additional factors such as pre-defined vs. potential locations (items in the user’s existing schedule vs. the pool of possibilities), duration of stay (in a practical sense, the user would not leave immediately after arrival), opening hours, and a weighted metric (venue ratings).

A specialisation of the TSP known as the *TSP with Selective Cities and Multiple Time Windows* by Mesquita, Murta, Paias, and Wise [6] addresses these issues by selecting a route that maximises profit across a weighted graph of mandatory and selective points while also taking visit durations into account. The objective is to find a tour that maximises the total weight and where the total travel time does not exceed that of some given value. Mandatory points are to be visited exactly once and selective points are to be at most once. Most significant is the relaxation of the requirement to visit all nodes in the decision space (of selective points). Rather, points are typically selected to maximise some pre-defined heuristic.

These heuristic-based algorithms can also be done under an approach known as branch-and-bound. Gendreau, Laporte, and Semet [7] have defined one such algorithm for the Selective TSP.

To define our algorithm, we can expand upon the algorithm devised for the Multi-Constraint Team Orienteering Problem with Multiple Time Windows (MCTOPMTW) of Souffriau et al. [8], which implements an Iterated Local Search (ILS) mixed with Greedy Randomised Adaptive Search Procedure (GRASP). GRASP has been known to provide successful results for closely-related specialisations of the Selective TSP, the MCTOPMTW [8] and the Team Orienteering Problem [9].

# Problem Description

In this section, we will define all the components we need for a formal problem proposal and algorithm implementation. To formally define our problem of extended route navigation, we would like to use the TSP with Selective Cities and Multiple Time Windows (TSPSNTW) [6] as a framework, with the exception that each mandatory item has a fixed time window (as a pre-defined schedule item). Elements from the work of Gendreau, Laporte, and Semet [7] are also in our formulation. This defines a route that maximises an overall weighted sum of prize metrics while considering the following:

* The user’s schedule will be converted to a set of mandatory items, where the ‘gaps’ are to be ‘filled in’ with selective items to optimality.
* For the sake of simplicity and to encourage a variety of visited venues, we assume that each location point is to be only visited once. If a mandatory point is defined twice in the schedule (e.g. a student has two lectures at the same lecture theatre at different times of the day), we count this as two distinct location points. However, selective points can strictly only be visited once to encourage diversification.

We decided not to assume anything else about the user’s schedule, including mealtimes, since we would also like our algorithm to cater for any pre-defined schedule. i.e. We allow the user to specify their own mealtimes. The user also has the option of listing “Cafes” as a category of interest.

Consider the problem as a directed graph , with venues and the distances between represented by vertices and arcs respectively. This graph is used to plot out each venue and keep track of venues that have already been visited.

Here, is a vertex set of venues where is the starting location (known as a *depot* in most vehicle routing problems). A subset is used to denote the compulsory vertices, with . Essentially, we want to optimise for the selection of the points based on the user’s preference i.e. the members of are filtered by the selected categories of interest. Each has a specified duration time which refers to the expected time that the user will spend at . This excludes the endpoint , where the route ends upon arrival to .

Additionally, each venue will have descriptive information – categories of interest (e.g. Museum, Study, Relaxation), a rating , and opening and closing hours denoted by and respectively. Opening and closing hours are irrelevant to mandatory tours since it is implied that they are open.

is an arc set with each representing the distances between and . Since distances in real-world routing problems are not typically straight-line distances, members of needn’t satisfy the triangle inequality. Each also has a corresponding non-negative cost value . For our problem, is equivalent to the travel time from to .

Each has an assigned profit with a total overall prize to be maximised. This value also takes each included cost into account. For all , the profits can be set to . Since they must be included in the solution regardless, having a quantified profit value assigned is trivial. For , since an item of zero or negative profit would never be chosen by a maximisation algorithm. The profit is calculated as a function of venue rating and duration below (rating has higher priority than duration, since we follow the principle of quality being preferable to quantity):

Define binary variables and (with ) for each vertex and arc respectively. We define if and only if appears in our final (optimised) solution and otherwise. Similarly, if and only if appears in the tour of our solution and otherwise. Additionally, we can assume that the solution contains at least 3 total vertices since a tour with 2 vertices is simply the start and end destinations, and anything less than 2 vertices is trivial.

Let us define as the total time frame of the completed extended route, punctuated by and , the start and end times respectively. and denote the start and finish times of the user’s visit to the venue . There is no finish time for the last venue since it is simply a destination to be arrived at, without the requirement to have its own duration. Note that . Let be a continuous time variable indicating the arrival time of the user to venue . For a mandatory venue , will be the scheduled start time.

We can describe our problem through an Integer Linear Programming (ILP) formulation where we aim to maximise the total profit minus the total cost:

The problem is further detailed by the following constraints:

Constraints (3) and (4) are simply formalisations of the binary variables and as previously defined.

Constraints (5) – (7) further define to restrict the connections between venues. (5) and (6) guarantee that the first and last location points must be included in the tour, as well as at least one extra vertex. (7) guarantees that any venue included in the solution must have associated feasible routes – excluding the first and last venues, any route arriving to the venue from a previous venue must have an exit leaving for the next venue.

Constraints (8) – (12) formally define the binary variable possibilities for both mandatory and selective venues. (8) and (9) respectively state that the user must visit each mandatory venue exactly once, and each selective venue at most once. (10) and (11) follow on from this with the connecting edges, with routes guaranteed to connect to a mandatory venue, and optional routes for connections to selective venues. (12) connects the variables and by ensuring that if a venue is visited, there must have been a route there from a previous venue (excluding the start point).

Constraints (13) – (15) describe the possible value for time at venue . (13) states that visits to each selective venue, including travel times back and forth, must be between visits to the mandatory scheduled venues (this includes the start and end venues). (14) maintains consistency with the fact that a given venue will commence at a later point in time than a previous venue (with being the start time). (15) defines the continuous time variable more precisely as a sum of the time point from the previous venue, the duration spent there, and the travel time from to .

Finally, constraints (16) and (17) consider the opening and closing hours of the venue. In (16), is only considered as a possible point in the tour if the venue is open upon the user’s arrival from a previous location. On the other hand, (17) requires that the venue closes after the arrival time and given duration of stay i.e. if the user can spend a reasonable amount of time at the venue.

# An ILS-based Algorithm

The problem statement defined above assumes only the essential components for route generation. However, much of the information provided to the algorithm is used for effective pruning of data, with the variables and constraints above holding purely for the route navigation aspect.

Our algorithm will take the following inputs from the user:

* Start and end time
* Start and end destination
* Existing schedule
* Categories of interest (optional – all possible selective venues will be included if nothing is selected)
* Type of traveller (optional – this will be set to Solo by default)

We propose a two-stage algorithm for our current problem. Our algorithm will first go through a pre-processing stage involving route initialisation and pruning of unnecessary venues. In the second stage, we use model (1)–(17) and combine the ILS and the GRASP approaches by using the MCTOPMTW solution [8] as a framework. This stage gradually builds up the set of venues in a feasible manner, while improving the solution through successive iterations of inserting and deleting venues.

We can implement this algorithm by extracting data from Google Maps and/or TripAdvisor and taking schedule data from the user.

## Pre-Processing

Pre-processing is a two-stage process – initialisation of the mandatory venues and preparation of the selective venues.

Firstly, an empty extended route is initialised as an empty set of venues between and . The first venue and the endpoint are added at points and respectively. Now, the remaining mandatory venues are filled out with their respective time slots. At this stage, where there are mandatory venues. All remaining (non-included) venues now make up the pool of selective venues.

Secondly, only the relevant data is to be used as selected by the user’s preferences. This can be done with a simple filter function over the entire set of selective venues using categorical tags (if the user did not select any categories, this filter will not change the selective venue set). A second filter function can be applied to the “Traveller Type” field. This will account for the user’s preference and prune the pool of selective venues accordingly.

## An ILS-GRASP Hybridisation

Now that we have a suitable pool of selective venues, we can construct our solution by iteratively inserting them between rigid mandatory venues. We do this with a hybridisation of the ILS and GRASP approaches.

Fig. 1: Pseudo-code for a combined ILS-GRASP algorithm

Existing research by Ibaraki et al. [10] and Hashimoto et al. [11] has proven the ILS approach to be extremely useful in navigation problems with time windows. While our model does not have strictly define time windows, the ‘gaps’ between visits to mandatory venues are conceptually similar. Let us define each of these gaps as where and are consecutive mandatory venues. Each (where ) can be considered independently from each other, so we apply the ILS-GRASP approach on each . Let the added vertices between each be defined by e.g. the second vertex added into would be .

For the sake of the algorithm, define a continuous time variable used as a global timer (a variable that returns the current time at the point it is called at).

ILS is an iteration route-building approach, done in two stages:

1. Insertion stage – Used to create a feasible route through iterative insertion
2. Shake stage – Used to diversify venues through removal of some venues to prevent local venue clustering

### Insertion stage

We can use GRASP to produce the Insertion stage. GRASP is an adaptive greedy procedure consisting of two stages:

#### Stage One – Construction Stage

The route is initially constructed by a greedy algorithm that iterates through the solution set and adds elements one by one. First, we must consider that a visit will occur between the opening and closing hours of the venue and i.e. that constraints (16) and (17) are respectively held. To further limit the solution set to possible venues, we define as the time consumed by inserting a venue into the route between and :

This is valid as long as the addition of does not violate the pre-defined time constraints, namely (13). For this to hold true, there must be enough time between the previous venue and the mandatory venue (deadline of that particular ) to visit this venue:

From our pool of possible elements, each venue is listed in order of some greedy heuristic. In our case, we use Vansteenwegen’s ratio heuristic [12]:

The venue with the highest value is taken, with the corresponding and arcs (distances) added to the solution. Between each new selective venue addition, the solution set is adapted to reflect the new state accordingly. That is, the values of and . The finish time of the newly inserted venue is also required for calculation of the next iteration i.e. .

If there exists no (i.e. no selective venue) that fits these time constraints, there will likely be a time gap between the previous and the next venue in the solution that translates to free time for the user.

#### Stage Two – Local Search Stage

This stage improves upon the construction stage by iteratively replacing the current with a better solution in its neighbourhood, with one replacement per iteration until no better solution can be found.

Overall, using the GRASP formula requires a decimal input between 0 and 1 as a measure of greediness. The closer this value is to 0, the greedier; whereas values close to 1 may be too generalised.

### Shake stage

Selective venues are deleted to provide opportunities for a better overall solution. It prevents venues from clustering up in a single area of the solution set. For every iteration of the Shake stage, at least one selective venue is deleted from the problem set. This is done using two variables:

1. represents the number of consecutive tours to delete
2. represents the venue from which to start deletion

and are both initialised as 1, however, these numbers are subject to change throughout the overall algorithm. After each iteration, variables , and will similarly need to be updated as required.

### An ILS-GRASP algorithm

By applying both the GRASP-based Insertion and Shake phases, the final algorithm for the ILS-GRASP hybrid is generated. Pseudo-code is provided in Fig. 1.

We want to initialise and to their initial values, as well as a large positive integer for the number of iterations. The GRASP Insertion stage is applied, and the algorithm then searches for a more optimal solution than the current solution, up until all iterations (with no improvement) have been used up or the optimal solution has been found in the local area. is required to be a sufficiently large number so that many iterations are made e.g. 150. If a more optimal solution is found, (a better solution has been found, implying less incentive to delete more venues). Otherwise, will stay at its current value.

After the Shake stage is applied, is incremented, as the number of deletions must grow in relation to the number of insertions. is updated accordingly using so that the areas being ‘shaken’ are keeping up with the new additions.

We also use three different greed values for GRASP, becoming greedier each time. Initially, a generalised greed value is recommended for initial scouting of the solution set, however, we can afford to specialise by taking greedier solutions more after an initial sequence has been established. These greed values can be altered to return slightly different routes to provide different suggestions for the user, as long as the values progressively become strictly greedier.

Finally, we end up with many separate optimised routes for each instance of . Under the rare condition that no selective routes are available for a , return the sub-route as it is. Now, if we join these by filling in all the gaps between every mandatory venue, we gain our final sequence of venues.

## Algorithm solution example with Example 1

We can use Example 1 as defined in the introduction and apply it to the algorithm to show a possible solution it could return.

We re-introduce Example 1 as a more rigorously defined version in terms of the problem statement:

* Schedule:
  + “Lecture Hall 1”, 9:00, 10:00
  + “Restaurant 1”, 13:00, 15:00
  + “Seminar Room 1”, 17:00, 18:00
* 9:00
* 18:00
* Categories of interest: “Relaxation”, “Study”
* Traveller type: “Solo”

A possible solution as given by the algorithm is detailed as follows:

Firstly, the pool of selective venues are filtered such that only those with the categories of interest “Relaxation” or “Study” are chosen, as well as the venues with traveller type as “Solo”. There are two s in this example, 10:00–13:00 and 15:00–17:00. These are inspected in detail below:

* For , we could have:
  + “Library 1” (Study), 6 minutes, 161 minutes
  + 13 minutes
* For , we could have:
  + “Botanical Gardens” (Relaxation) (Solo), 28 minutes, 40 minutes
  + “Cafe 1” (Relaxation), 15 minutes, 20 minutes
  + 17 minutes

Putting together all the venues gained from each , we have the following solution creating a route in order of venues visited:

* “Lecture Hall 1”, 9:00, 10:00
* “Library 1”, 9:06, 12:47
* “Restaurant 1”, 13:00, 15:00
* “Botanical Gardens”, 28 minutes, 40 minutes, 15:28, 16:08
* “Cafe 1”, 16:23, 16:43
* “Seminar Room 1”, 17:00, 18:00

This corresponds to the solution that we proposed in the introduction – however, now everything is rigorously defined, and we can see how the times fit accordingly.

## Sourcing of maps and other external data

Existing geographical data, ratings, recommended duration, and categorical tags (of interest) per venue. All this information is available on Google Maps and TripAdvisor. Geographical and navigational data can be sourced from Google Maps; gaining this information is its intended purpose. Google Maps also provides ratings and can provide a recommended duration based on crowdsourced location data to find the average time. This recently added feature comes in the form “*People typically spend min here*”. We can take the objective value of as our recommended duration for the corresponding vertex . Sometimes this is provided as a range e.g. “*People typically spend 15-45 min here*”; under these circumstances we can simply take the median value. Finally, TripAdvisor provides tags with the venues in its database, as well as tags for types of travellers (Families, Couples, Solo, Business or Friends).

# Evaluation and Discussion

## Computational Complexity

Let us observe Fig. 1 with mandatory venues, thus with selective venues and s. The computational complexity of the ILS-GRASP algorithm is:

In a single iteration, both the GRASP and the Shake stages will take at most a proportion of calculations to define or update values for each selective venue available for that particular . As long as there are a constant number of iterations, the while loop is of runtime . The second-most loop runs for a constant amount of time , and the outer-most loop runs times.

The worst-case scenario occurs when all selective venues are available for all gaps (i.e. available opening and closing hour times, and duration). This would translate to more calculation time in the Insertion stage required for values such as the . However, fortunately this is not disproportionately larger since most of these calculations are of constant time .

## Overall Evaluation and Discussion

Our algorithm addresses a problem based on the Traveling Salesman Problem with Time Windows. Each point in the graph-based tour is either a rigid mandatory or potential selective venue, with consideration of extra features such as ratings, categories and opening/closing hours.

The major difference between the MCTOPMTW model and ours is our distinction of mandatory and selective venues. Another difference is that they use separate constraints to define cost (represented by knapsack constraints of pricing [8]) and time (total time budget – where only start and end times are fixed), while our cost function is directly proportional to the travel time. This simplifies our solution as we do not need to consider these two setbacks separately.

Regarding the Shake stage in our ILS algorithm, although perhaps unintuitive to delete what are already routes with the maximum local profit, the ILS has proven to be very useful in many orienteering problems [10]. The Shake stage seeks to diversify and expand the areas in which potential venues are being sourced from. Without it, our algorithm may yield solutions that ‘trap’ the user with venues in one very small neighbourhood, especially if the start and destination venues are nearby or the same [12]. This can be avoided during the Shake stage through correction of initial bad choices.

Many other implementations consider a waiting time alongside the arrival time in case the scheduled event hasn’t started yet i.e. the store is not yet open. However, we chose to discard this part to maximise user productivity. Even breaks can be carefully scheduled by the algorithm by choosing categories such as “Relaxation”. Slight gaps are likely to appear in the timeline, since it is unlikely for a mandatory venue (excluding the start point) to perfectly match up with the venue immediately preceding it, considering the rigid travel times and durations required. These gaps appear without the definition of waiting times. The algorithm is used to maximise profit in a limited time frame, so waiting time is not of heavy consideration.

The overall algorithm, including the pre-processing stage, considers many more factors than existing implementations of Selective TSP-based problems. These factors include the consideration of categories and traveller types, as well as mandatory venues with rigid times, rather than just the presence of these mandatory venues. described mandatory venues to take place during set time windows, however, our algorithm assumes specific times. For this reason, our algorithm implements the ILS procedure multiple times (directly proportional to the number of mandatory venues) rather than the usual single iteration.

Finally, the algorithm maximises for the best solution that yields the best value of within a given number of iterations (150). Since the optimal solution is unnecessary, this is a good way to cap the number of iterations to return a route that yields a good enough result of rather than search for the optimal solution indefinitely. By capping the number of iterations, there is a greater focus on efficiency rather than optimising , which is generally of higher priority to users. Users typically prefer a fast app with good suggestions to a slow app with an optimal solution. We also cannot guarantee that our optimal solution will be the best route for the users, so it is preferred to give several suggested routes. We can generate as many different suggestions as we want by altering the greed values. Roughly 3-5 different routes can be suggested to provide the user with freedom of choice without overloading them with too much choice.

Regarding the solution given for Example 1, it gives the user a few suggestions that directly relate to their preferences (Relaxation and Study) while giving them adequate time to enjoy their venues with carefully formulated values of .

# Conclusion

In this paper, we devised an algorithm based on the ILS and GRASP approaches for the generation of a personalised route, both of which are used in specialised TSP optimisation. The algorithm and the problem itself have much in common to the TSP, so TSP-based solutions were examined and applied to our problem.

We are hoping that this algorithm can help boost user productivity as well as find venues that they would be interested in visiting. By combining map data and user preference, we seek to provide the best and most enjoyable route for the individual.

# Future Work

The algorithm can be implemented in actual code to generate test data for more detailed analysis with the sourcing of maps and other data as discussed. Furthermore, it could be implemented into a real-life smartphone application with a proper GUI to be used on-the-go. We can apply the algorithm in our application by using it to calculate the relevant routes after requesting the user data (schedule, start and end information) and preferences (categories, traveller type) in an initial selection screen. Then, the resultant tour could be integrated with an existing navigation app to show a timeline with the actual routes for between-venue travel. This can be applied with 3-5 generated routes for the user to pick from.

Many extensions can be made to future implementations of our schedule-based extended route software.

The algorithm could be extended to cater for mealtimes e.g. provide restaurant suggestions for breakfast, brunch, lunch, and dinner at certain popular mealtimes. Although this may not be suitable for users with irregular schedules, which was of heavy consideration in the algorithm proposed in this paper, the practical implications would be useful for most users. This would make the consequent application a powerful suggestion tool rather than one that just “fills in the gaps” for the user.

The problem could also consider more real-world restrictions such as involving a total budget for the trip. By integrating the cost function with money, we would extend the application to cater for people of all financial situations and provide even more personalised routes.

# Acknowledgements

I would like to thank my supervisor, Jochen Renz from the Research School of Computer Science at The Australian National University, for his support and guidance throughout this project.

References

|  |  |
| --- | --- |
| [1] | F. Zhan and C. Noon, "Shortest Path Algorithms: An Evaluation Using Real Road Networks," *Transportation Science,* vol. 32, no. 1, pp. 65-73, 1998. |
| [2] | H. B. Yan and Y. C. Liu, "A New Algorithm for Finding Shortcut in a City's Road Net Based on GIS Technology," *Chinese Journal of Computers,* no. 2, pp. 201-215, 2000. |
| [3] | D. Douglas, "Least-cost Path in GIS Using an Accumulated Cost Surface and Slopelines," *Cartographica: The International Journal for Geographic Information and Geovisualization,* vol. 31, no. 3, pp. 37-51, 1994. |
| [4] | J. K. Lenstra and A. H. G. Rinnooy Kan, "Some Simple Applications of the Travelling Salesman Problem," *Operational Research Quarterly (1970-1977),* vol. 26, no. 4, pp. 717-733, 1975. |
| [5] | E. J. Beltrami and L. D. Bodin, "Networks and Vehicle Routing for Municipal Waste Collection," *Networks,* vol. 4, no. 1, pp. 65-94, 1974. |
| [6] | M. Mesquita, A. Murta, A. Paias and L. Wise, "TSP with Multiple Time-Windows and Selective Cities," in *Computational Logistics*, Heidelberg, Germany, Springer Berlin Heidelberg, 2013, pp. 158-172. |
| [7] | M. Gendreau, G. Laporte and F. Semet, "A Branch-and-Cut Algorithm for the Undirected Selective Traveling Salesman Problem," *Networks,* vol. 32, no. 4, pp. 263-273, 1998. |
| [8] | W. Souffriau, P. Vansteenwegen, G. V. Berghe and D. V. Oudheusden, "Multi-Constraint Team Orienteering Problem with Multiple Time Windows," *Transportation Science,* vol. 47, no. 1, pp. 53-63, 2011. |
| [9] | W. Souffriau, P. Vansteenwegen, G. V. Berghe and D. V. Oudhousden, "A Path Relinking approach for the Team Orienteering Problem," *Computers & Operations Research,* vol. 37, no. 11, pp. 1853-1859, 2010. |
| [10] | T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno and M. Yagiura, "Effective local search algorithms for routing and scheduling with general time-window constraints," *Transportation Science,* vol. 39, no. 2, pp. 206-232, 2005. |
| [11] | H. Hashimoto, M. Yagiura and T. Ibaraki, "An iterated local search algorithm for the time-dependent vehicle routing problem with time windows," *Discrete Optimization,* vol. 5, pp. 434-456, 2008. |
| [12] | P. Vansteenwegen, W. Souffriau, G. V. Berghe and D. V. Oudheusden, "Iterated local search for the team orienteering problem with time windows," *Computers & Operations Research,* vol. 36, no. 12, pp. 3281-3290, 2009. |
| [13] | M. J. Egenhofer and R. D. Franzosa, "Point-set topological spatial relations," *International journal of geographical information systems,* vol. 5, no. 2, pp. 161-174, 1991. |
| [14] | J. Renz and M. Westphal, "Evaluating and minimizing ambiguities in qualitative route instructions," in *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '11*, Chicago, IL, USA, 2011. |
| [15] | S. Dutta, "Approximate spatial reasoning: Integrating qualitative and quantitative constraints," *International Journal of Approximate Reasoning,* vol. 5, no. 3, pp. 307-330, 1991. |