Thomas Tregel, Philipp Müller, Stefan Göbel, Ralf Steinmetz: Where's Pikachu: Route Optimization in Location-Based Games. In: Proceedings of the 10th International Conference on Virtual Worlds and Games for Serious Applications, p. 81- 87, IEEE, September 2018. ISBN 978-1-5386-7123-8.

Where's Pikachu: Route Optimization in Location-Based Games

Thomas Tregel
Multimedia Communications Lab
TU Darmstadt
Darmstadt, Germany
thomas.tregel@kom.tu-darmstadt.de

Philipp Müller
Multimedia Communications Lab
TU Darmstadt
Darmstadt, Germany
philipp.mueller@kom.tu-darmstadt.de

Stefan Göbel
Multimedia Communications Lab
TU Darmstadt
Darmstadt, Germany
stefan.goebel@kom.tu-darmstadt.de

Ralf Steinmetz
Multimedia Communications Lab
TU Darmstadt
Darmstadt, Germany
ralf.steinmetz@kom.tu-darmstadt.de

***Abstract*—Along with the sudden rise in popularity of location-based games, the demand for tools to assist players in performance optimization increased. One major aspect lies within the analysis and planning of routes that contain a high amount of desirable game locations for the given player. However, personalized routes cannot be created by hand due to the amount of available in-game locations and the associated time constraints for real-world travel.**

This paper presents a system to dynamically create personalized route for players, based upon previous game data. These routes can be fully customized with regards to their time and location as well as the player's desired in-game goal, allowing them to e.g. specifically target their favourite species or maximize the amount of visited locations. The system is evaluated using a dataset of Berlin containing over 30.000 distinct locations with different associated in-game behaviour. Regarding the system's performance it is designed to work as an assistance system on a mobile device to assure its applicability in the context of location-based games.

Keywords—location-based games, optimization, player assistance, mobile devices

I. Introduction

Popular location-based games like Ingress or Pokémon Go have demonstrated the public interest in this genre quickly reaching a user base of up to 45 million users worldwide [2]. Studies indicate that playing such games has a positive health influence due to the players' increase daily activity [1]. For the respective target audience the availability of smartphones for mobile games increases according to a yearly study in Germany, which shows a steady increase in smartphone availability for teenagers reaching 97% in 2017 [3].

Due to the nature of the game using a discovery-based approach for Pokémon collection, players quickly tried to investigate ways to improve their performance within the game, which resulted in different services to automatically scan the surrounding area.

However, in order to provide users with a way to use efficient routes to maximize their progress, a user guidance system is needed. The goal of this system is to allow users to define their start and goal position as well as the amount of time they intend to spend. Based upon previous data from the aforementioned scan system, an optimal route can be calculated on the mobile device which maximizes the user's goals.

II. Related Work

A. *Content representation in Pokémon Go*

In Pokémon Go a spawn points is a distinct geolocation that automatically generates a singular Pokémon every hour. In order to determine which of the wide range of Pokémon to spawn, the location's context is used. Context like being located in an industrial area or near a river directly influences the distribution of possible Pokémon. Additionally, each spawn point generates a Pokémon at a set number of seconds of each hour, with them being available for 30 minutes most of the time. Spawn points which are located within parks or similar recreational areas additionally have a behaviour called "nests". Spawn points that belong to a nest are assigned a single distinct Pokémon on a biweekly basis, which is spawned most frequently at these locations [4].

B. *Traveling Salesman Problem with Time Windows*

The traveling salesman problem (TSP) is a well-known problem in the fields of mathematics and computer science that has been extensively studied ever since it was first brought up as a mathematical problem by Karl Menger in 1930 [5, 6, 7, 8]. It is the problem of finding a cheapest (shortest) route that visits all cities on a given set of cities and then returns to the start location. The traveling salesman problem with time windows (TSPTW) is a generalization of the TSP which adds time windows to each city, limiting when a city may be visited [9].

Both problems are typically modelled as a pathfinding problem on a weighted graph where each node represents one city, each edge between two nodes represents a path between the two respective cities and each edge weight is equal to the respective cost of the path, i.e. the minimum length of the path between both cities. They can be shown to be in NP-hard and furthermore in NP-complete, suggesting that they cannot be solved in polynomial time unless $P = NP$ [10, 11, 12].

C. Solution Approaches

Solution approaches for both problems can be separated into exact algorithms and approximative algorithms. Whereas exact algorithms guarantee an optimal route, approximative algorithms utilize heuristics in an attempt to find a route that’s reasonably close to the optimal route within a significantly shorter runtime.

Because of their comparatively long runtimes, exact algorithms are nowadays almost exclusively used for small instances of the TSP(TW). Pure branch and bound algorithms such as proposed in [9] have been mostly replaced by branch and cut algorithms which combine branch and bound with cutting planes to solve the TSP(TW) formulated as an integer linear program [13, 14]. The best performing exact algorithm for the TSP we could find is a branch and cut algorithm called Concorde solver [15].

Particularly for real-time applications and larger problem instances, approximative algorithms are typically necessary in order to keep runtimes manageable. Some of the earliest approximative algorithms for the TSP use constructive heuristics in order to generate a route by iteratively adding arcs [16, 17]. A more recent approach utilizes local search heuristics such as Tabu search or k-Opt in order to find a local optimum in a neighbourhood of valid routes [18, 19, 20, 21]. Simulated annealing and evolutionary algorithms are currently some of the best-performing algorithms for the TSP(TW) utilizing such neighbourhood structures [21]. Simulated annealing is a metaheuristic which is inspired by the process of annealing in material science and has shown to perform well on many different problems [22, 23, 24, 25, 26, 27]. Inspired by biological evolution, on the other hand, evolutionary algorithms show similar performance to simulated annealing [28, 29]. A different approach to neighbourhood-based algorithms called variable neighbourhood search utilizes multiple different neighbourhoods in order to achieve a similar performance on the TSPTW [30]. Based on the behaviour of ants when searching for paths to food sources, ant colony optimization algorithms such as BEAM-ACO [31] and Ant-Q [32] have been successfully applied to the TSP, matching other state of the art algorithms in performance. Machine learning approaches, particularly self-organizing maps, have also been successful in achieving similar performance to other state of the art algorithms on the TSP [33, 34]. An approach which combines nested Monte-Carlo search with other techniques has been able to match but not surpass any of the commonly used algorithms for the TSPTW [35]. Other, more novel approaches such as programming a carbon nanotube [36] or simulating a shrinking material blob [37] have been successfully applied to the TSP but lack in either scalability or performance.

III. PROBLEM DEFINITION

A problem definition for the problem of route finding in location-based games can be developed by considering typical player requirements and system restrictions. Player requirements are typical requirements a player would have towards a desirable route. System restrictions, on the other hand, are imposed on a route based on the functionality of spawn locations in typical location-based games.

A player would typically want a route to start at a specific point in time and location, e.g. at their current time and location. Since a player cannot be expected to have an unlimited amount of time available, the route would have to finish at a specific location before a specific point in time. Depending on the player’s circumstances, the target location of a route can either be the same as the start location or another location entirely. In order to maximize the game progress a player is expected to achieve on a route, the route needs to maximize the number of viable spawn locations visited. In the context of Pokémon Go, a viable spawn location is one that is currently active and capable of spawning a Pokémon that is beneficial to the player. Depending on the player’s goals in the game, this could either be any Pokémon or a number of specific Pokémon such as those the player has not yet caught. A player would therefore want an option to specify which Pokémon they are interested in catching, i.e. which Pokémon are relevant to them.

A typical system-based restriction for routes found in location-based games is the limited active time of spawn locations, i.e. a route may only visit a spawn location when it is currently active. In Pokémon Go, a spawn location is considered currently active if there is currently a chance of a Pokémon being found at that location. Since Pokémon at a given spawn location respawn periodically and regardless of whether a player has previously caught a Pokémon at that location, a spawn location may be revisited at a later active period to increase the chances of encountering a desired Pokémon or catching multiple Pokémon at that location.

Additional properties necessary to determine whether a route is feasible in practice include an estimation for the retention time of a player at each location as well as an estimated travel time between two locations for each pair of spawn locations. In Pokémon Go, the expected retention time would estimate how long it takes for a player to catch a Pokémon at a given location. In practice, this could either be a constant value selectable by the player or be determined adaptively by the route-finding system based on statistical player data. To determine the expected travel time between two spawn locations, different approaches are thinkable. A simple approach would let the player select an estimated travel speed and calculate an estimated travel time based on the linear distance between both locations. More sophisticated approaches might incorporate advanced route data and expected traffic conditions such as provided by the Google Maps Directions API [38] when estimating travel times between locations.

In order to apply the problem to other location-based games which function similar to Pokémon Go but provide different properties to determine whether a spawn location is desirable to a player, we introduce a single numerical value for each spawn location representing the value it provides to a player when visited. For Pokémon Go, this value is calculated by summing up the spawn probabilities of all Pokémon relevant to a player for a given location, resulting in a value between 0 (no Pokémon capable of spawning at that location are relevant to the player) and 1 (all Pokémon capable of spawning at that location are relevant to the player). When applying the problem to other location-based games, only this conversion formula would have to be adjusted to correspond to the respective game's spawn location properties.

Based on aforementioned requirements, restrictions and properties, we specify the following problem definition for route finding in location-based games:

Given a start location s_s, a target location s_t, an earliest start time t_s, a latest finish time t_f, a set of n spawn locations $S = \{s_1, \dots, s_n\}$ and for each spawn location s an expected retention time $t_r(s) \geq 0$, a value $v(s) \geq 0$, an expected travel time $d(s, s_2) \geq 0$ to each other spawn location s_2 and uptime information consisting of a period length between spawns $t_{period}(s)$, a relative spawn time $t_{spawn}(s)$ and a relative despawn time $t_{despawn}(s)$, find the shortest valid route R consisting of a sequence of spawn locations L and a sequence of arrival times T which maximizes the sum of all values on the route.

A valid route is a route which starts at s_s no earlier than t_s, ends at s_t no later than t_f and visits spawn locations only when they are currently active but never twice within the same active time window. For each spawn location s_1 on the route, given its arrival time t_1, the arrival time t_2 of the next spawn location s_2 on the route can be no earlier than $t_r(s_1) + d(s_1, s_2)$. The time between arrivals on two consecutive spawn locations s_1, s_2 on the route has to be at least equal to the expected travel time in addition to the retention time on the first spawn location $d(s_1, s_2)$

It can be shown that the TSP is a specialization of the specified problem when particular input values are chosen. The specified route finding problem is therefore at least as computationally difficult as the TSP and has to be in the complexity class of NP-hard problems as well.

IV. CONCEPT

In the following we evaluate existing approaches to the TSP and TSPTW with respect to performance and adaptability to the specified route-finding problem. We then outline optimization algorithms capable of approximatively solving the problem in real-time which were developed based on two versatile metaheuristics that have shown to perform well on the TSPTW.

A. *Feasibility of existing approaches*

Whereas we would ideally want to find an optimal route for any given problem instance, we don't consider exact algorithms to be currently feasible to be used in real-time for typical instance sizes. The best-performing exact algorithm for the TSP known to us, the Concorde solver, already shows runtimes of multiple days per problem instance for an instance size of just 4,500 nodes [16]. For a more computationally complex problem such as discussed in this paper and for larger problem instances, e.g. the Berlin dataset consisting of more than 30,000 spawn locations that is available to us, exact algorithms can currently only be expected to further exceed this runtime figure and therefore be unfit for use in real-time.

This leaves only approximative algorithms as feasible to generate routes for location-based games in real-time. Amongst the many different approximative approaches showing a similar performance on the TSP(TW), we decided to base our algorithm on those which we consider particularly simple to adapt to our problem, allowing us to put a larger focus on optimization over adaptation. We therefore chose to use two approaches which are very similar in nature, namely simulated annealing and an evolutionary algorithm. Both approaches utilize a neighbourhood structure to generate new route candidates and an objective function to evaluate generated route candidates.

B. *Pre-processing*

Before generating a route using either of the two selected metaheuristics, multiple pre-processing steps are applied to a problem instance. Frequently used values are pre-calculated and the number of locations to be considered is reduced in order to improve the performance of our route finding algorithms.

In the first pre-processing step, a value $v(s)$ is calculated for each spawn location s which represents the player's expected desirability of Pokémon spawning at that location. Based on the relevancy r_i for each Pokémon i and its spawn probability $p_i(s) \in [0,1]$ at a given spawn location s, it is calculated as follows:

$$v(s) = \sum_{i=1}^{p} (p_i(s) \cdot r_i)$$

Under the assumption that the expected travel time $d(s_1, s_2)$ always takes the shortest route between s_1 and s_2, any spawn location with a value of 0 cannot be on the optimal route and is therefore filtered out of the dataset.

When using approximated linear distances and an expected travel speed to calculate expected travel times between spawn locations, all necessary conversions are applied to their coordinates during pre-processing in order to allow efficient calculation of travel times during route generation. Coordinates are first realigned relative to the smallest occurring coordinates. These are then converted into approximated metric units as described in [39]. Lastly, the approximated metric units are converted into seconds of travel time based on the expected travel speed. Optionally, if sufficient memory is available, a distance matrix consisting of distances between all spawn location pairs can be generated.

In a third pre-processing step, all spawn locations are filtered out which cannot be reached within the available amount of time. The filtered set of spawn locations S' is calculated as follows:

$$S' = \{s \in S | d(s_s, s) + t_r(s) + d(s, s_t) \leq t_f - t_s\}$$

In a fourth pre-processing step, a grid is generated which divides the area of all remaining spawn locations into quadratic cells with a fixed side length. For each cell, the spawn locations it contains as well as its centre coordinates are stored. Furthermore, a distance matrix is generated which stores distances between all pairs of cells. This distance matrix is later

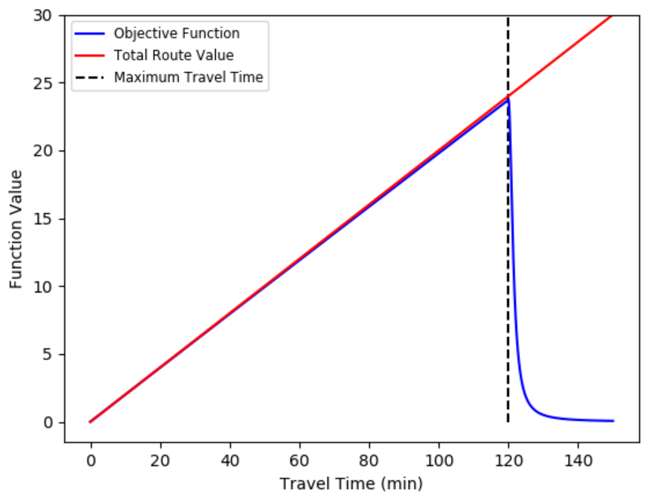

Fig. 1. Objective function value compared to total route value for routes with varying total travel times.

used in our heuristic neighbourhood function to efficiently determine feasible spawn locations to add to the route.

C. Optimization algorithms

Because of their similar nature, simulated annealing and evolutionary algorithms can utilize the same structures, functions and heuristics in order to be applied to a problem. This property allowed us to simultaneously develop two optimization algorithms, each based on one metaheuristic. In the following, we provide an overview over concepts which we utilize in our optimization algorithms to adapt both metaheuristics to the problem of route finding in location-based games.

For simulated annealing, we chose a cooling schedule of $T_t = \frac{C}{\log(t)}$ based on its convergence properties shown in [26]. C is set to the maximum value of all spawn locations in the dataset to ensure that the algorithm doesn't get stuck in a local minimum. In order to achieve more consistent results, the algorithm is restarted a variable number of times. Configuration parameters include the number of temperatures, iterations per temperature and restarts. The returned route is the best seen route during any iteration as defined by its objective value.

For the evolutionary algorithm, we chose a genetic algorithm with mutations based on a single parent. A variable survival rate determines the percentage of all routes in the population which are removed between generations and replaced by mutations of the remaining routes. Roulette wheel selection such as described in [40] is used in order to select routes during the removal and mutation processes based on their objective value. Configuration parameters include the population size, number of generations and survival rate. Again, the best seen route during any generation as defined by its objective value is returned.

We store routes as sequences of visited locations and notably don't explicitly store arrival times. The represented route is then the shortest valid route with that sequence of visited locations for a given dataset. In particular, arrival times can be iteratively calculated as the earliest possible arrival times which adhere to the requirements of a valid route according to the problem specification. By representing routes like this, the problem's search space can be reduced, allowing for better performance of optimization algorithms on the problem. Furthermore, a representation similar to those commonly used in genetic algorithms, loosely based on DNA sequences, is achieved, allowing for our neighbourhood function to apply similar mutation steps.

In order to evaluate a route's quality, we present the following objective function:

$$o_{route} = \frac{v_{route}}{1 + \frac{\sqrt{\min\{t_{route}, t_{max}\}} + \max\{0, t_{route} - t_{max}\}^2}{t_{max}}}$$

Fig. 1 exemplarily shows the behaviour of this objective function for a desired maximum travel time t_{max} of 120 minutes under the assumption that the summed value v_{route} of all spawn locations on the route increases proportionally to its total travel time t_{route}. The objective value is proportional to the total route value v_{route} and decreases for routes with larger travel times. In order to promote routes which maximize total value, any time spent before the desired maximum travel time t_{max} is exceeded is only taken into account as its square root. By squaring the amount of time exceeding t_{max}, the function simultaneously penalizes routes which don't adhere to t_{max} while still assigning them a meaningful objective value for optimization algorithms to work with.

In order to generate a new route based on an existing route (mutate a route), we developed two different neighbourhood functions. A simple neighbourhood function generates a new route based on an existing route by randomly adding a spawn location, removing a spawn location, replacing a spawn location by another or swapping the position of two spawn locations on the route. An advanced neighbourhood function, called heuristic neighbourhood in the following, prioritizes operations which are likely to improve a route's quality. Most notably, when adding a spawn location to the route, the route's remaining time before t_{max} is reached is calculated and spawn locations which are likely to cause the route to exceed t_{max} are not considered. This is achieved by filtering out grid cells and thus spawn locations on the grid generated during pre-processing based on their distance to the two consecutive spawn locations on the route between which a new spawn location is supposed to be added.

Because of limited availability of advanced route data such as provided by the Google Maps Directions API [38], we calculate estimated travel times based on Euclidean distances between spawn locations and a variable estimated travel velocity. Given the frequency of estimated travel time calculations in our algorithms and a relatively close proximity of spawn locations, we furthermore estimate distances as linear distances instead of calculating great circle distances using e.g. the Haversine formula described in [41]. In order to reduce the number of calculations, calculated values can be stored in a distance matrix if sufficient memory is available.

V. IMPLEMENTATION

In the following, we provide a small overview of the implementation that was used to evaluate our proposed algorithms. Included are a parser for datasets of spawn sightings, a command-line interface (CLI) for quick access to the implemented algorithms and a prototypical graphical user interface (GUI) providing a visualization of spawn locations and generated routes.

A. Dataset preparation

Based on a list of spawn sightings in the .csv format, our implemented dataset parser generates a serialized set of spawn locations to be used by our route finding algorithms. Each spawn sighting consists of the Pokémon sighted (by Pokémon number), a location ID with coordinates and a despawn time. The despawn time specifies the number of minutes into an hour after which the Pokémon despawns. Since all spawn locations in Pokémon Go use the same period of one hour between spawns and despawns always occur exactly 30 minutes later, it is possible to precisely calculate all active time windows of a spawn location based on just the despawn time of one of its respective spawn sightings. This is used in order to generate a set of unique spawn locations, each consisting of its coordinates, its relative active time window and a list of Pokémon with their spawn probabilities for that location.

B. GUI

The GUI consists primarily of a geographical world map which is used to display generated routes between spawn locations. By clicking on the map, a user can select desired start and target locations respectively. Additionally, the user can input desired start and target times, their expected travel speed and whether they're looking for a specific Pokémon. In a mobile application, some of these inputs could be automatically set to relevant default values, e.g. using the user's current time and location.

A quality slider provides users with the option of choosing between faster route generation and better route quality. In order for this quality slider to be useful for a large range of use cases and systems, a quadratic function is used to translate the quality slider's setting into configuration parameters for our optimization algorithms. The quality scale ranging from 0 to 10 results in a number of temperatures between 30 and 30,030 for simulated annealing or a number of generations between 1,000 and 1,001,000 for the evolutionary algorithm. The respective configuration parameters to be adjusted were chosen based on how good of a trade-off they provide between runtime and route quality. Other configuration parameters were kept at the default values used in our evaluation.

VI. EVALUATION

We assessed our algorithms for different configuration parameters by generating a test set of 100 routes for each combination of parameter values. Unless the assessed parameter was specific to either metaheuristic, this was done once for each metaheuristic. The generated route data was then statistically evaluated to obtain information on runtime performance of route generation and quality of generated routes as defined by our objective function for each test set. Utilizing our GUI implementation, we furthermore confirmed the validity of generated routes with respect to the problem specification and practical applicability on a sample basis.

Based on the runtime performance of exact solvers such as the Concorde solver [15] on the less computationally complex TSP, it doesn't appear feasible to us to determine optimal routes for the problem instances used in our evaluation. A comparison of generated routes to optimal routes was therefore not possible.

The test sets were generated on a desktop PC utilizing an Intel Core i5-4670K processor running at a maximum clock speed of 3.4 GHz with a maximum heap size of 2 GB.

The dataset utilized in our assessment is generated based on 5,181,910 unique spawn sightings in Berlin and consists of 32,148 unique Pokémon spawn locations across an area of 309 square kilometres, resulting in a density of 104 spawn locations per square kilometre. Within this dataset, the Brandenburger Tor and Potsdamer Platz were selected as respective start and target locations for the routes to be generated. Given their location within the city of Berlin and their close proximity to another, the number of spawn locations filtered out during pre-processing is kept comparatively low in order to obtain close to worst case performance data for our algorithms.

A. Default Parameter Values

The default problem parameter values are chosen to be representative of a typical use case of our route finding algorithms. The player starts at the Brandenburger Tor at 18:00, wants to be at the Potsdamer Platz by 20:00 and hasn't recently visited any spawn locations. They move at an average speed of four kilometres per hour and stay at each spawn location visited for one minute. They don't have any preferred Pokémon to catch and therefore the relevancy r_i is set to 1 for all Pokémon. The time between spawn events is set to 60 minutes and the time before Pokémon despawn is set to 30 minutes based on observations in Pokémon Go.

The default configuration parameter values were empirically obtained in an attempt to achieve the best trade-off between runtime and route quality for use in real-time applications. In order not to exceed a few seconds of runtime on slower mobile CPUs, we decided upon a maximum median runtime of one second per generated route on our system. We then searched for configuration parameter values which maximize route quality as defined by a route's objective value without exceeding the one second runtime limit.

By default, distances are approximated as linear distances and not stored in a distance matrix. Instead of a simple random neighbourhood function, our heuristic neighbourhood function is used. The grid used within our heuristic neighbourhood function has a cell size of 500x500 metres. Simulated annealing restarts five times, each going through 300 different temperatures and 120 iterations per temperature for a total of 216,000 iterations. The evolutionary algorithm uses a population size of 40 route candidates, a survival rate of 50% per generation and runs for 10,000 generations, resulting in approximately 200,000 mutations in total.

For the default configuration, both algorithms show performance within 1% of another with respect to runtime and

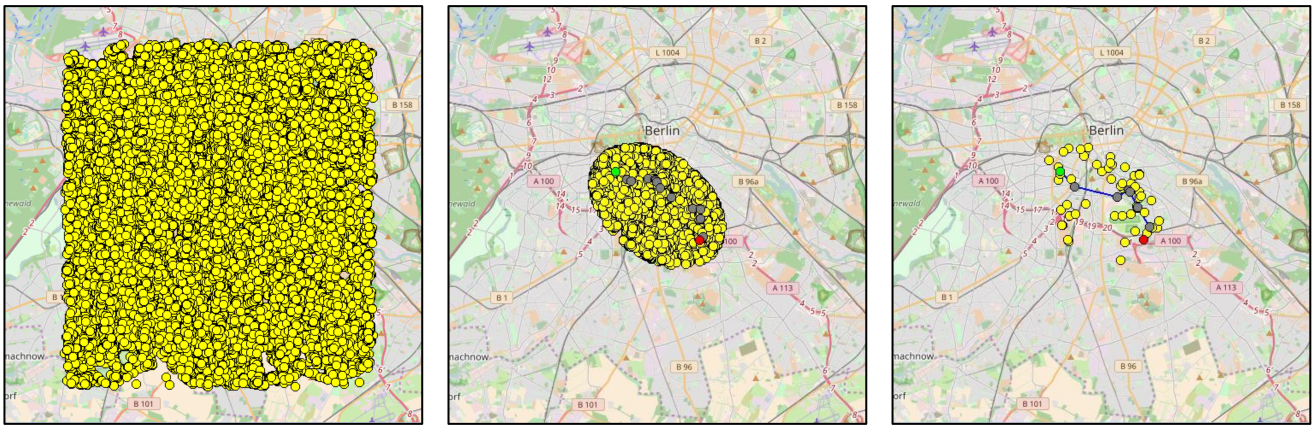

Fig. 2. Berlin dataset without pre-processing (left), after distance filtering (centre) and after value filtering (right).

quality of generated routes as indicated by their objective value. Compared to the evolutionary algorithm, simulated annealing shows a significantly lower variance in both metrics. Routes generated by both algorithms never exceeded the desired maximum route travel time of 120 minutes by more than 30 seconds, suggesting that our objective function adequatly limits route travel times for use in practice. As expected, the number of locations visited on generated routes is distributed similarly to the objective value. The median number of locations visited is 42.0 for both algorithms. If you exclude the retention time of one minute, this results in an average travel time of 1.93 minutes between two visited spawn locations. Based on the assumed travel speed of four kilometres per hour, an average distance between visited spawn locations of 128 metres can be calculated. After manually assessing generated routes and available spawn locations, we believe these to be very reasonable values for good routes. In particular, we expect these routes to be significantly better than routes users would typically come up with when manually planning their routes, even if they had all spawn information available to them in an accessible format.

A limitation based on our usage of linear distances instead of more advanced route data such as provided by the Google Maps Directions API [38] is the lack of practicability for some of the generated routes. Some routes would, for example, repeatedly cross a river in a location where no bridge is nearby. Our approach using linear distances is therefore mostly suited for more open areas such as fields and forests whereas areas with large scale obstacles such as rivers or housing blocks would benefit from using more sophisticated distance functions, e.g. by pre-calculating a distance matrix for each dataset based on advanced route data in order to keep route generation times low.

Based on these results, we consider both optimization algorithms to be feasible for use in practice if distance calculation is appropriately handled. Whereas simulated annealing is showing more consistent results compared to the evolutionary algorithm, we consider the difference to be small enough to be dominated by other factors such as implementation details or chosen configuration parameters in practice.

B. *Selected parameters with interesting outcomes*

In total, we individually assessed 15 different problem and configuration parameters with respect to their impact on runtime and route quality when generating routes with both optimization algorithms. In the following, we will present select parameters with noteworthy effects on route generation.

Whereas both optimization algorithms generally performed very similarly when varying different problem parameters, the evolutionary algorithm performed notably better for higher travel speeds. For the highest tested travel speed of 25 km/h, it was able to generate routes with a 5% larger median objective value in 15% less median runtime compared to simulated annealing. When varying maximum travel time, objective values no longer increased proportionally to maximum travel time for routes longer than one hour whereas runtimes increased overproportionately, suggesting a decline in relative route quality and overall performance for routes longer than one hour. We expect this to be a result of the larger problem instance size, partially caused by fewer spawn locations being filtered out by the distance filter during pre-processing.

Compared to the simple neighbourhood function, the heuristic neighbourhood function has shown to have significantly different effects on runtime and route quality depending on which optimization algorithm is used. For simulated annealing, it has shown to be largely beneficial as it resulted in a 162% increase in median objective value at a 71% increase in median runtime with a lower runtime variance. For the evolutionary algorithm, however, the median objective value only increased by 78% at a 159% increase in median runtime with higher variances in both values. A more favourable trade-off between runtime performance and route quality can be achieved by choosing a larger value for the size of grid cells used by the heuristic neighbourhood function.

C. *Pre-processing*

Pre-processing runtimes were tracked for all routes generated during our evaluation. The highest recorded pre-processing time was still below 50ms with typical values between 2ms and 5ms, resulting in a relatively small impact on total runtime.

Whereas some pre-processing steps are mandatory for our algorithms to properly function, distance and value filtering have proven to be valuable tools to keep the problem instance size manageable. For our default parameter values in particular, the number of spawn locations is reduced from 32,148 to 7,375 during pre-processing, a reduction of more than 77%. Fig. 2 shows the Berlin dataset unfiltered (left), after distance filtering (centre) and after distance plus value filtering with one sought-after Pokémon (right). Each dot represents one spawn location with the colour determining whether a spawn location is the selected start location (green), the selected target location (red), on the generated route (grey) or none of the aforementioned (yellow). As can be seen, value filtering can further significantly reduce the number of spawn locations by up to more than 99% depending on which Pokémon is sought after.

VII. SUMMARY & OUTLOOK

We have developed, implemented and evaluated different algorithms and heuristics for route optimization in location-based games. In order to do so, we analysed a location-based game and developed a problem definition for route finding in this game. We then abstracted this problem definition to be applicable to other location-based games and identified its properties with respect to potential solution approaches. We assessed the feasibility of applying different solution approaches commonly used in dealing with the traveling salesman problem to the problem of route finding in location-based games. Based on this assessment, we elaborated on promising solution approaches and developed different concepts necessary to apply selected solution approaches to the specified problem. This includes a series of pre-processing steps to improve performance and offer additional functionality, a route representation, an objective function and two different neighbourhood functions which allow two different metaheuristics to be applied to the problem.

We implemented two configurable optimization algorithms which utilize all developed concepts. Additionally, we implemented parsers for spawn location data, an extensive command-line interface (CLI) and a prototypical graphical user interface (GUI) which serves as a prototype for a mobile application which could allow users to effortlessly optimize their routes in location-based games.

We evaluated all concepts with respect to functionality and performance for different problem and configuration parameter values and when using different heuristics. We manually reviewed routes generated within the GUI regarding their quality and practical value. Using the CLI, we generated a large number of test sets which we then statistically analysed to obtain and evaluate data about runtime performance and route quality for both optimized algorithms. Based on this data, we evaluated the performance and usability of both optimization algorithms on different problem instances. Furthermore, we evaluated the impact on overall performance that different heuristics and configuration parameter values have.

In addition to the simulation of realistic movement using distance matrices on the real road network, modality-based interaction is an interesting aspect. Using the user's current movement type or to incorporate public transport routes to possibly bridge larger gaps in suggested routes is a promising extension possibility.

REFERENCES

[1] T. Althoff, R. W. White, and E. Horvitz, "Influence of Pokémon Go on physical activity: Study and implications" Journal of Medical Internet Research 18(12). 2016.

[2] The Guardian.: Pokémon No: game's daily active users, downloads and engagement are down. URL https://www.theguardian.com/technology/2016/aug/23/pokemon-go-active-users-down-augmented-reality-games. 2016. [Online, accessed April 23, 2017].

[3] JIM 2017. Jugend, Information, (Multi-) Media. Basisstudie zum Medienumgang 12- bis 19-Jähriger in Deutschland. Medienpädagogischer Forschungsverbund Südwest. 2017.

[4] Researching Pokémon GO Spawn Mechanics. URL https://pokemongohub.net/generation-2/researching-pokemon-go-spawn-mechanics/ [Online, accessed April 06, 2018].

[5] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. The traveling salesman problem: a computational study. Princeton university press, 2011.

[6] Karla L Hoffman, Manfred Padberg, and Giovanni Rinaldi. Traveling salesman problem. In Encyclopedia of operations research and management science, pages 1573–1578. Springer, 2013.

[7] Jérôme Monnot and Sophie Toulouse. The traveling salesman problem and its variations. Paradigms of Combinatorial Optimization: Problems and New Approaches, Volume 2, pages 173–214, 2014.

[8] Karl Menger. Untersuchungen über allgemeine metrik. Mathematische Annalen, 103(1):466–501, 1930.

[9] Edward K Baker. Technical note-an exact algorithm for the time-constrained traveling salesman problem. Operations Research, 31(5):938–945, 1983.

[10] Richard M Karp. Reducibility among combinatorial problems. In 50 Years of Integer Programming 1958-2008, pages 219–241. Springer, 2010.

[11] Jan Karel Lenstra and AHG Kan. Complexity of vehicle routing and scheduling problems. Networks, 11(2):221–227, 1981.

[12] Lance Fortnow. The status of the p versus np problem. Communications of the ACM, 52(9):78–86, 2009.

[13] Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Time dependent traveling salesman problem with time windows: Properties and an exact algorithm. Technical report, 2014.

[14] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming, 90(3):475–506, 2001.

[15] Holger H Hoos and Thomas Stützle. On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. European Journal of Operational Research, 238(1): 87–94, 2014.

[16] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research, 46(3):330–335, 1998.

[17] Andrew B Kahng and Sherief Reda. Match twice and stitch: a new tsp tour construction heuristic. Operations Research Letters, 32(6):499–509, 2004.

[18] Martin WP Savelsbergh. Local search in routing problems with time windows. Annals of Operations research, 4(1):285–305, 1985.

[19] Fred Glover. Tabu search-part i. ORSA Journal on computing, 1(3):190–206, 1989.

[20] Fred Glover. Tabu search-part ii. ORSA Journal on computing, 2(1):4–32, 1990.

[21] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization, 1:215–310, 1997.

[22] Anthony Rollett, FJ Humphreys, Gregory S Rohrer, and M Hatherly. Recrystallization and related annealing phenomena. Elsevier, 2004.

[23] Emile Aarts, Jan Korst, and Wil Michiels. Simulated annealing. In Search methodologies, pages 265–285. Springer, 2014.

[24] Lester Ingber. Simulated annealing: Practice versus theory. Mathematical and computer modelling, 18(11):29–57, 1993.

[25] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. science, 220(4598):671–680, 1983.

[26] Bruce Hajek. Cooling schedules for optimal annealing. Mathematics of operations research, 13(2): 311–329, 1988.

[27] Mathias Ortner, Xavier Descombes, and Josiane Zerubia. An adaptive simulated annealing cooling schedule for object detection in images. PhD thesis, INRIA, 2007.

[28] David B Fogel. Evolutionary algorithms in theory and practice, 1997.

[29] Huai-Kuang Tsai, Jinn-Moon Yang, Yuan-Fang Tsai, and Cheng-Yan Kao. An evolutionary algorithm for large traveling salesman problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(4):1718–1729, 2004.

[30] Nenad Mladenovi´c, Raca Todosijevi´c, and Dragan Uroševi´c. An efficient general variable neighborhood search for large travelling salesman problem with time windows. Yugoslav Journal of Operations Research, 23(1):19–30, 2013.

[31] Manuel López-Ibáñez and Christian Blum. Beam-aco for the travelling salesman problem with time windows. Computers & Operations Research, 37(9):1570–1583, 2010.

[32] Marco Dorigo and LM Gambardella. Ant-q: A reinforcement learning approach to the traveling salesman problem. In Proceedings of ML-95, Twelfth Intern. Conf. on Machine Learning, pages 252–260, 2016.

[33] Thiago AS Masutti and Leandro N de Castro. A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. Information Sciences, 179(10):1454–1468, 2009.

[34] SM Abdel-Moetty. Traveling salesman problem using neural network techniques. In Informatics and Systems (INFOS), 2010 The 7th International Conference on, pages 1–6. IEEE, 2010.

[35] Stefan Edelkamp, Max Gath, Tristan Cazenave, and Fabien Teytaud. Algorithm and knowledge engineering for the tsptw problem. In Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on, pages 44–51. IEEE, 2013.

[36] Kester Dean Clegg, Julian Francis Miller, Kieran Massey, and Mike Petty. Travelling salesman problem solved ´Sin materioŠby evolved carbon nanotube device. In International Conference on Parallel Problem Solving from Nature, pages 692–701. Springer, 2014.

[37] Jeff Jones and Andrew Adamatzky. Computation of the travelling salesman problem by a shrinking blob. Natural Computing, 13(1):1–16, 2014.

[38] Google Maps Directions API. URL https://developers.google.com/maps/documentation/directions/ [Online, accessed January 12, 2017].

[39] Martin Kompf. Distance calculation. URL http://mkompf.com/gps/distcalc.html. [Online, accessed January 12, 2017].

[40] Jinghui Zhong, Xiaomin Hu, Jun Zhang, and Min Gu. Comparison of performance between different selection strategies on simple genetic algorithms. In Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on, volume 2, pages 1115–1121. IEEE, 2005.

[41] Andrew Hedges. Finding distances based on latitude and longitude, July 2002. URL http://andrew.hedges.name/experiments/haversine/. [Online, accessed May 20, 2017].