EFFICIENT TOUR PLANNING FOR A MEASUREMENT VEHICLE BY COMBINING NEXT BEST VIEW AND TRAVELING SALESMAN

Joachim Gehrung[1,2,*], Marcus Hebel[1], Michael Arens[1], Uwe Stilla[2]

[1] Fraunhofer IOSB, Ettlingen, Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, 76275 Ettlingen, Germany - (joachim.gehrung, marcus.hebel, michael.arens)@iosb.fraunhofer.de
[2] Photogrammetry and Remote Sensing, Technische Universitaet Muenchen, 80333 Muenchen, Germany - stilla@tum.de

Commission II, WG II/10

KEY WORDS: Mobile Laser Scanning, Next Best View, Traveling Salesman Problem, Evidence Grids, Change Detection

ABSTRACT:

Path planning for a measuring vehicle requires solving two popular problems from computer science, namely the search for the optimal tour and the search for the optimal viewpoint. Combining both problems results in a new variation of the Traveling Salesman Problem, which we refer to as the *Explorational Traveling Salesman Problem*. The solution to this problem is the optimal tour with a minimum of observations. In this paper, we formulate the basic problem, discuss it in context of the existing literature and present an iterative solution algorithm. We demonstrate how the method can be applied directly to LiDAR data using an occupancy grid. The ability of our algorithm to generate suitably efficient tours is verified based on two synthetic benchmark datasets, utilizing a ground truth determined by an exhaustive search.

1. INTRODUCTION

Much of the literature on change detection deals with finding changes between two or more already recorded epochs. However, if the change detection takes place at scale of an entire city, which must be recorded again in the future, then it is beneficial to focus on the points where changes are most likely to happen. Such points can be determined, for example, by examining historical measurements using conventional change detection methods and then checking for clusters and hotspots. This procedure saves time and resources and, due to its predictive component, could also be referred to as *predictive change detection*. This results in a large amount of areas of interest, which must be visited by a measuring vehicle for the purpose of re-recording said locations.

Recording an urban environment efficiently with a measurement vehicle is a non-trivial task. This can be demonstrated on the overview of a small town district shown in Figure 1. There is no route to visit all green location markers without some streets being visited twice, which causes redundant measurements. For points that are close together, it may be possible that they can be measured at the same time from a certain location. Such points are symbolized by the red location markers. If such places are taken into account, then finding a tour that allows it to measure all green location markers is reduced to visiting the minimum subset of green and red markers. This leads to the blue route. The fields of research associated with both involved problems are referred to as **Traveling Salesman** and **Next Best View**.

Planning a measurement ride efficiently by hand in an area such as the one shown in Figure 1 is a cumbersome and time consuming task that most likely doesn't result in an efficient vehicle path. The example mentioned above is only a small area and efficient path planning is already almost impossible here, which leads to the conclusion that an algorithmic solution to the problem at hand is necessary. In this work, we propose a new class of theoretical problem resulting from the combination of the Next Best View and Traveling Salesman Problem referred to as **Explorational Traveling Salesman Problem**. We propose a solution to the problem in form of an iterative, heuristic algorithm. Furthermore, we demonstrate how to bridge the gap between the theoretical problem and LiDAR measurements.

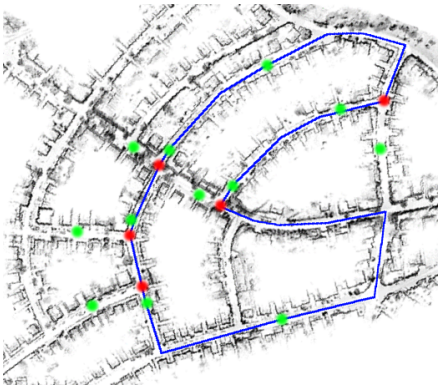

Figure 1. A point cloud projected onto the ground plane, which gives an overview of a district in the city of Ettlingen, Germany. Green markers are points of interest, red markers stand for indirect viewpoints. The blue line is a route that allows to measure all points of interest.

The structure of the paper is as follows. Section 2 provides an overview of the literature on the subjects of *Next Best View* and the *Traveling Salesman* problem. A theoretical analysis of the novel *Explorational Traveling Salesman Problem* as well as an explanation of the algorithm we developed for solving it is presented in Section 3. Section 4 includes the method's functional verification and its evaluation, the results are discussed in Section 5. Information about future work can be found in Section 6.

* Corresponding author

2. RELATED WORK

2.1 Next Best View

2.1.1 About Topics such as environment modeling, autonomous exploration as well as object inspection, recognition and reconstruction require solving a common problem known as the *View Planning Problem (VPP)* or more commonly as *Next Best View (NBV)*. An extensive overview of the topic is given by Scott et al. (Scott et al., 2003). The task itself requires finding a sequence of viewpoints, which usually have six or more degrees of freedom. These include three degrees of freedom each for rotation and translation, but also additional ones for sensor parameters such as zoom or exposure time.

The problem in general includes reasoning about the state of the environment, the possible viewpoints and the imaging space (Scott et al., 2003). It seems to be a simple, yet computationally complex task that has shown to be NP-complete (Tarbox and Gottschlich, 1995). Different definitions of the problem exist, where the following informal definition by Scott et al. is a good fit for this work. "For a given environment, find a suitably short view plan satisfying the specific observation goals and achieve this within an acceptable computational time" (Scott et al., 2003).

2.1.2 Model-based approaches Scott et al. distinguish between **model-based** and **non-model-based** approaches for solving NBV. The former ones utilize a priori knowledge of the environment at some level of fidelity to compute a view plan. Methods of this kind can be categorized by the representation used for the knowledge embedded in the model (Scott et al., 2003).

Set theory methods, such as the one presented by Tarbox and Gottschlich, utilizes a visibility matrix describing the visibility of discrete points-of-interest from each point in a quantized viewpoint space (Tarbox and Gottschlich, 1995). The authors argue that finding the shortest possible view plan is NP-complete and therefore suggest heuristic methods to find good, but not necessarily optimal solutions.

Another category of methods is based on **graph theory**. An aspect graph represents the neighborhood between different aspects of a region or object. Each node in the graph summarizes a set of viewpoints that all share the same view in a qualitative manner, the arcs represent adjacency in viewpoint space (Tarbox and Gottschlich, 1995, Bowyer and Dyer, 1990). Although it is an interesting concept, there are practical difficulties, since aspect graphs for even moderately complex objects quickly become huge, as does the required computational complexity.

The **art gallery problem** is a well known problem from the category of computational geometry. It deals with the question of how many guards are required to monitor all walls of an art gallery, the latter being represented by a 2D polygon (J. Kahn, 1980). Solutions are usually of a geometric nature. There are numerous extensions of the problem, including some that deal with three dimensions.

2.1.3 Non-model-based approaches The category of non-model-based view planning algorithms requires minimal to no prior knowledge. Since these algorithms are not of importance for this work, only a short overview over non-model-based techniques is given. Scott et al. suggested to categorize non-model-based approaches by the domain of reasoning about viewpoints (Scott et al., 2003).

Volumetric methods favor those viewpoint that will extract the greatest amount of unknown scene information (Banta and Abidi, 1996). Both workspace and object are represented by a voxel grid, ray tracing is applied for visibility analysis. **Surface-based techniques** can further be categorized in occlusion edge methods (Maver and Bajcsy, 1990), contour following (Pudney, 1994) and parametric surface representations (Whaite and Ferrie, 1991). **Global view planning methods** derive viewpoints from global rather than local characteristics (Yuan, 1995).

2.2 Traveling Salesman Problem

2.2.1 About The *Traveling Salesman Problem (TSP)* is a well-known problem from graph theory that is assumed to be NP-hard. A comprehensive overview over the topic is given by Matai et al. (Matai et al., 2010). Johnson and McGeoch provide an in-depth analysis of multiple solution approaches (Johnson and McGeoch, 2007). The problem can be described as finding a tour of N cities, with the following conditions:

1. visit every city just once
2. return to the starting point
3. the tour must be of minimum distance

According to (A. J. Hoffman, 1985), the TSP was first mentioned in a handbook for traveling salesmen from 1832. The problem was studied in the 18th century by the Irish mathematician Sir William Rowan Hamilton and by the British mathematician Thomas Penyngton Kirkman (Biggs et al., 1986). It applies to a variety of real-world problems, such as the drilling of printed circuit boards, X-ray crystallography and the order-picking problem in warehouses (Matai et al., 2010).

2.2.2 Symmetric and asymmetric TSP There are two main categories, namely the symmetric and the asymmetric TSP. In case of the former one, the travel costs between two cities are the same in both directions. This means that the problem can be described using an undirected graph. The latter one is represented by a directed graph, since the travel costs are not the same in both directions. This case is more general, but less extensively explored (Matai et al., 2010, Johnson and McGeoch, 2007).

2.2.3 Evaluation of solution strategies Since the problem is at least NP-hard, heuristics are applied to find approximate solutions. To provide a metric for the performance of a new proposed algorithm for solving the TSP, the **Held-Karp lower bound** has been established (Valenzuela and Jones, 1997). It provides a lower bound for a graph's optimal TSP tour. The lower bound is calculated using an iterative algorithm with a gradient descend method. Said algorithm generates gradually higher-cost minimum 1-trees, the cost and shape of which grows closer and closer to that of the optimal tour.

2.2.4 Tour construction Tour construction algorithms are heuristics for solving the TSP. They terminate once a solution is found and do not improve it further, which in turn is the task of *tour improvement*. Approximate algorithms do not generate the exact solution. Their performance can be measured using the Held-Karp lower bound mentioned above. A comprehensive comparison of various approaches using multiple benchmarks has been done by Johnson and McGeoch (Johnson and

McGeoch, 2007). The structure of this section reflects in part their way of classifying the approaches they examined.

There are heuristics such as **Strip** (Beardwood et al., 1959), **Spacefilling Curve** (Bentley, 1992) and **Fast Recursion Partitioning** (Bentley, 1992) that focus more on speed, less on tour quality. They construct the tour one edge at a time and their results are within 31-35 % of the Held-Karp lower bound. It can clearly be seen that the results are biased by the way the algorithms work. The complexity of the approaches depends on the implementation decisions made. In case of the first two it depends on the search algorithm used, in case of the last one it may be similar to the one of a *k-d tree*.

Augmentation based approaches construct tours by adding one edge at a time. The simplest and most straightforward approach in this category is **Nearest Neighbors**. It can also be used to solve the asymmetric TSP (Johnson et al., 2007). Starting from a randomly picked city, the closest city is added to the tour. The result is generally within 23 % of the Held-Karp lower bound. The complexity of the approach is $O(N^2)$, but using a *k-d tree* can reduce this to something like $O(N \log(N))$ in practice (Johnson and McGeoch, 2007). The **Greedy** heuristic gradually constructs a tour by repeatedly adding the shortest edge to the tour. This does not happen in relation to any particular node as in the *Nearest Neighbors* method, but globally in the graph. This must not lead to more than two edges per node or to a circle less than the number of all cities N. The complexity is $O(N^2 \log(N))$ and the result is within 14 % of the Held-Karp lower bound (Johnson and McGeoch, 2007). In the variations dubbed **Boruvka** and **Quick Boruvka** devised by Applegate, Bixby, Chvátal and Cook in an analogy to the classic minimum spanning tree of Borůvka (Borůvka, 1926), the priority queue is replaced by sorting or no sorting is used at all. It is not evident whether **Greedy** or **Boruvka** generates the better tours, but **Quick Boruvka** is presumed to trade tour quality for speed. Their Held-Karp lower bound is similar to the one of **Greedy** (Johnson and McGeoch, 2007). **Savings** is another Greedy-like heuristic proposed by Clarke and Wright (Clarke and Wright, 1964). The basic idea is to combine several smaller tours into one. It works in the same way as Greedy, but with a surrogate distance function and is considered to be 12 % above the Held-Karp lower bound (Johnson and McGeoch, 2007). In many practical cases, it is considered to dominate other approaches (Johnson and McGeoch, 2007).

2.2.5 Tour improvement The refinement of an already existing tour is referred to as **tour improvement**. Since this topic is only mentioned for the sake of completeness, the overview is kept brief. There are several solutions for tour improvement, the most common ones are the **2-opt** and **3-opt** local searches (David S. Johnson, 1995). A generalized variant of this is called **k-opt**. Lin and Kernighan proposed an algorithm that allows to get within 2 % of the Held-Karp lower bound. It's a variation of the k-opt algorithm and decides which k is the most suitable at each iteration step. The complexity is approximately $O(N^{2.2})$. An in-depth study of improvements of the algorithm can be found in (Helsgaun, 2000). There are other well researched solutions such as **Tabu search**, **simulated annealing** and **genetic algorithms**. Further information can be found in (David S. Johnson, 1995).

3. FINDING A SUITABLY SHORT PATH THAT VISITS ALL AREAS OF INTEREST

3.1 Problem description

The problem dealt with in this work can be summarized as follows. Given a 3D scan of an area, a measurement vehicle is supposed to determine a path that allows observing a previously defined set of points of interest and then return back to its starting point. For two reasons, simple path planning is not sufficient for this. First, the subset of all possible viewpoints required to observe all areas of interest in an efficient way is unknown. This problem category is also known as *Next Best View*. As mentioned in Section 2.1, it is presumed to be NP-complete. Second, the order in which said viewpoints are to be visited is unknown. This is called the *Traveling Salesman Problem*, which is presumed to be NP-hard (cf. Section 2.2).

Finding the solution of the problem at hand is challenging, since there is a circular dependency between the solution of both partial problems. The best subset of viewpoints depends on the chosen tour. Choosing the tour depends on the subset of viewpoints. In other words, it is a version of the classical chicken-egg-problem. There are additional conditions that are more practical in nature. The vehicle may only move on navigable surfaces and also must avoid collisions. The path proposed by the solution must be supported by the kinematics of the vehicle. We demonstrate how to handle these practical challenges in Section 3.8. In order to address all those problems as well as the ones related to visibility calculations required by Next Best View, we decided to apply a framework used for spatial representation developed in earlier works (Gehrung et al., 2019). Within this framework, arbitrary spatial information is represented as a three-dimensional distribution called a **density function**. Such functions can easily be used to derive new information such as visibility or be combined with other density functions. The functions themselves are represented by octrees, as this is an efficient data structure for spatial data.

3.2 Combining NBV and TSP

The first step in solving the TSP is to map the problem onto a graph. For the case at hand that means creating a node for each available viewpoint. However, one of the basic reasons for solving the Next Best View problem is that not all possible viewpoints have to be visited. Regarding the graph, this means that not all nodes are part of the solution. The best way to explain this is to look at the problem from an iterative perspective. Whenever a partial solution is expanded by adding another node to the tour, then some of the nodes become irrelevant and are dropped from the set of nodes to be considered.

Due to the nature of the problem, the arc costs are also based on the solution of the TSP. This is because the exact path between two viewpoints depends on the orientation of the vehicle and therefore on its previous path. Every partial solution of the TSP is therefore a function of the previous partial solution.

Since both the subset of valid nodes as well as the arc costs have a certain dynamic, we call the problem that arises the **Dynamic Graph Traveling Salesman Problem** or, to emphasize its practical application, the **Explorational Traveling Salesman Problem**. We consider it to be a special category of the TSP, such as for example the *Vehicle Loading Problem* proposed by (Clarke and Wright, 1964).

3.3 Symmetric or asymmetric TSP?

The dynamic graph formulation of the problem leaves the question whether or not the problem is rooted in the symmetric or asymmetric TSP. This information is highly relevant in order to choose a suitable heuristic for solving the problem. The authors believe that in principle it can be both, but it cannot be decided definitively. The reason for this is as follows.

The path driven from tour point A to B may be identical to the one between B and A, but it not has to be. If a vehicle drives from A to B, switches to reverse and drives the same way back, both ways are equal. If the car turns before making its way back, the way between B and A is longer. In the first case the graph would be symmetric, in the second case asymmetric.

Whether or not it is a symmetric or asymmetric graph depends on the graph's arcs, these in turn are based on the solution. The solution is largely determined by the vehicle kinematics. The reason why this cannot be decided definitively is that once an arc has been added to a partial solution and therefore is fixed, the inverse arc cannot be evaluated without violating the conditions for a valid tour and therefore invalidate the solution. One may argue that it is possible to evaluate an arc without it being part of the solution, but adding the corresponding arc in the inverse direction would cause a tour point to be part of the same tour twice, which would invalidate the tour. So since there is no possibility to check the reverse arc, it is not possible to classify the underlying graph as symmetric or asymmetric.

3.4 Proposed solution

Based on the reasoning in the section above, we decided to build our algorithm for solving the **Explorational Traveling Salesman Problem** based on a method that is able to handle both the symmetric and asymmetric TSP. The **Nearest Neighbor** algorithm was chosen, because it leads to reasonable good results in terms of the Held-Karp lower bound, has a lower complexity than most of the other approaches and works in an iterative manner, which simplifies designing a solution strategy for the problem at hand. Our algorithm is outlined in Algorithm 1. A brief overview is given below, the individual parts of the algorithm are explained in more detail in the following sections.

Our algorithm works based on a virtual, fully connected graph generated by connecting all potential viewpoints. Arc costs are initialized with the Euclidean distance. Starting from the vehicle position and orientation, the next node is determined based on a cost-benefit factor that includes the usefulness of the node as well as the distance to the current position. This is realized by a priority queue. Since the actual distance may be larger than the Euclidean distance, path planning based on a variation of hybrid A-Star (Kurzer, 2016) is applied to verify that there is actually a path and, if that is the case, to determine its length. If the determined path still has the best cost-benefit factor, then the node is added to the tour. If not, the new best candidate is taken into consideration. Once a new tour point is added to the tour, the usefulness of the other nodes in the graph is decreased accordingly. Once a valid tour has been found, it is improved using the 2-opt algorithm.

3.5 Graph construction

The nodes of the graph correspond to all possible points that can be visited by the vehicle. These can, for example, be a predefined list of points of interest or the position of all hotspots in a density function representing multiple areas of interest. We have chosen the latter one, since a density function allows the combination of multiple information sources, such as manually defined points and results of a change detection algorithm. A detailed explanation of density functions and the possibilities they offer can be found in (Gehrung et al., 2019). For the sake of simplicity, we define that the measuring vehicle must be close to the point of interest in order to measure it. Of course, any number of sophisticated heuristics are possible to determine arbitrary viewpoints, but the one mentioned should be considered sufficient for demonstrating the method. Due to the use of these points of interest, we name them **direct viewpoints**.

Additional nodes referred to as **indirect viewpoints** are also taken into consideration. These allow more than one point of interest to be observed at once. In order to determine the indirect viewpoints, the **visibility density functions** of all points of interest are determined and added. It describes the space that can be seen from said points of interest. Clustering and a hotspot search are applied on all elements of the density function that are in the intersection of at least two viewports. Mixing direct and indirect viewpoints leads to a **set of graph nodes** where some nodes are more useful than others in terms of the points of interest they allow to observe.

Based on this set of nodes, only the arcs need to be specified to obtain a valid graph. Creating a fully connected graph that represents the actual arcs, as well as their costs is not feasible. This is because each arc corresponds to the actual path between two locations, which in turn is based on the vehicle orientation, which is influenced by previous driving maneuvers. One may argue that it is viable to create a graph with a reasonable large subset of all possible vehicle orientations, but for larger problems this is highly inefficient. Determining an arc requires path planning in 3D, which is a rather expensive operation, considering that the number of nodes expanded in the hybrid A-Star algorithm grows exponential with the depth of the solution (Kurzer, 2016).

For the reasons listed above, we have chosen a naive approximation of the fully connected graph. The graph itself is not generated explicitly. Instead, only the parts that are required are generated at runtime. The arc costs are then initialized with the Euclidean distance between the nodes, since this corresponds to the minimum path length. In the case of large graphs, it would also be possible to only connect nodes within a certain local neighborhood. As already mentioned, no explicit instantiation of the arcs is necessary, since they can be created on the fly if required.

3.6 Solving NBV

Our approach used to solve the Next Best View problem is loosely based on the one by Tarbox and Gottschlich, since we also use a **visibility matrix** (Tarbox and Gottschlich, 1995). The **visibility matrix** is a two-dimensional matrix, the columns of which correspond to the points of interest and the rows to the viewpoints. If a point of interest is able to be seen by a viewpoint, then the corresponding element of the matrix is set to one, otherwise to zero. The **viewpoint weights vector** is a vector that corresponds to the sum of the rows of the visibility matrix. In other words, it contains the number of points of interest seen from a viewpoint. The purpose of this is to describe the usefulness of each viewpoint. If there is a viewpoint that is able to observe three points of interest, than it is more useful than another one that can only see one point of interest.

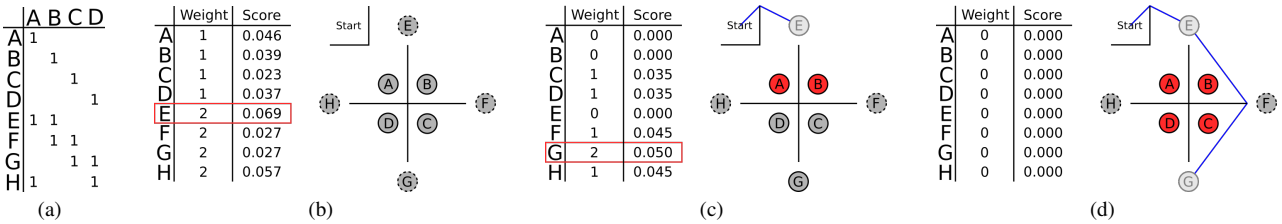

Figure 2. Example scenario illustrating our solution for tour planning. The visibility matrix (a) is used to derive the viewpoint weights vector (left side of b,c,d). In the first iteration (b), point E is chosen, since it is the nearest one with the highest score. The second iteration (c) leads to the choice of point G, after which the algorithm terminated, since the viewpoint weights vector is zero and all points have been observed (d).

Algorithm 1: Nearest neighbor based solving of the Dynamic Graph Traveling Salesman Problem.

Data: Occupancy density d_{occ}, start point s.
Result: The path between all chosen tour points.

Function `solve`(d_{occ}, s)**:**
- $nodes \leftarrow$ determineViewpoints(d_{occ})
- $vm \leftarrow$ calculateVisibilityMatrix(nodes, d_{occ})
- $weights \leftarrow$ calculateWeigths(vm)
- $tour \leftarrow \{s\}$
- $paths \leftarrow \{\}$
- **while** $weights.sum() > 0$ **do**
  - $l \leftarrow tour.lastElement()$
  - `// Add tour end neighbors to queue.`
  - $queue \leftarrow \{\}$
  - **for** n : *getNeighbors*$(nodes, l)$ **do**
    - $n.score \leftarrow \frac{weights[n]}{\|n-l\|}$
    - $queue.insert(n)$
  - `// Find next tour point.`
  - **while** $\neg queue.empty()$ **do**
    - $c \leftarrow queue.pop()$
    - `// Only the Euclidean distance between nodes is known.`
    - **if** $\neg paths[l, c]$ **then**
      - $paths[l, c] \leftarrow$ calculatePath(l,c)
      - **if** $\neg paths[l, c]$ **then**
        - **continue**
      - **else**
        - $c.score \leftarrow \frac{weights[c]}{paths[l,c].length()}$
        - $queue.insert(c)$
    - `// Path between nodes known.`
    - **else**
      - $tour.append(c)$
      - $weights.update(vm, c)$
      - **break**
    - `// Tour finding failed?`
    - **if** *queue.empty()* **then**
      - return {}
- $tour.append(s)$
- **return** constructFullPath(tour, paths)

Algorithm 2: Update of the viewpoint weights vector.

Data: The visibility matrix vm, the viewpoint weights vector $weights$, the chosen viewpoint n.
Result: The updated viewpoint weights vector $weights$.

Function `update`(vm, $weights$, n)**:**
- **for** c : *weights.columns()* **do**
  - **if** $vm[n][c] > 0$ **then**
    - **for** r : *weights.rows()* **do**
      - $weights[r] \leftarrow max(0, weights[r] - vm[r][c])$
- **return** $weights$

The selection of the next viewpoint is based on the score function described in the next section. When the next viewpoint has been chosen, the viewpoint weight vector is updated. The update procedure is described in Algorithm 2. The principle behind it is simple. The usefulness of every viewpoint that sees the same points of interest as the chosen viewpoint is downgraded. This is done by subtracting the visibility matrix columns of all seen points of interest from the weight vector.

3.7 Finding the next tour point

Finding the next tour point is done by selecting the one with the best cost-benefit factor. In this case, the benefit is measured by the number of points of interest that can be seen from a viewpoint. This information can be derived from the viewpoint weights vector. Costs are defined as the distance that the vehicle has to cover. The former one is taken from the viewpoint weight vector, the latter one equals the arc cost between two nodes. The score is defined as

$$score = \frac{weight}{distance}. \quad (1)$$

Note that the score for all tour points that are already part of the tour is automatically zero. A priority queue is used to efficiently determine the arc with the highest score. It is initialized with all arcs between the current tour node and all other nodes that are not part of the tour. The score is calculated based on the estimated arc cost, i.e. the Euclidean distance. Whenever it happens that a path between the current node and the top element of the queue is not known, it is calculated by means of the hybrid A-Star path planning algorithm (Kurzer, 2016). The score is then updated and the corresponding node is reinserted into the queue. If the path is already known, which happens only after the node has been reinserted into the queue, than the next tour point has been found. This process is repeated until all points of interest have been observed, which is the case when

the sum of the viewpoint weight vector elements is zero. If the queue is empty, but the next tour point has not been found, than the problem cannot be solved. This is the case when no valid path between the latest tour point and all other non-tour points can be found, at least not given the current vehicle orientation. If this happens, then one possible reason is that a driving maneuver is required that can't be performed by the vehicle. It may also be the case that a possible path is too narrow for the vehicle or that the area has not been sufficiently explored. An example illustrating how the algorithm works is given in Figure 2.

3.8 Density based path planning

Path planning is done using a rudimentary implementation of the hybrid A-Star algorithm, which allows to apply A-Star within a continuous space (Kurzer, 2016). The algorithm can simulate the driving behavior of a vehicle and is able to generate solutions that are considered to be almost optimal. Our implementation of the algorithm is rather fast, however, this leads to long runtimes if no path is possible.

In order to prevent the vehicle from colliding with obstacles, we sample an occupancy representation in form of an **occupancy density function**, generated from LiDAR range measurements. Samples are created in regular intervals using the vehicle's oriented bounding box. If a certain threshold in the accumulated evidence for *occupancy* is exceeded, a collision is assumed and the corresponding vehicle pose is pruned from the search space. To keep the vehicle on navigable surfaces, first all navigable surfaces are determined and then shifted upwards by the distance from the center of the vehicle to the ground. The **navigable areas density** derived in this way is sampled at the vehicle center point. The same threshold based test as before is applied to sort out invalid vehicle positions.

In order to encourage a smooth driving behavior, driving backward or not straight ahead is weighted with a higher cost. Since the density functions can be combined with each other, it is possible to make some areas more or less important than others by multiplying them with an appropriate density function. This can be used to keep the vehicle away from certain regions or to encourage it to give preference to these. This all is achieved using the g-score update of the A-Star algorithm:

$$g(n+1) = g(n) + dist(n, n+1) * penalty \tag{2}$$

The new g-score $g(n+1)$ is calculated by adding the distance covered since the last step $dist(n, n+1)$ to the previous g-score $g(n)$. The behavior mentioned above is achieved by multiplying the distance with a penalty factor:

$$penalty = f_d * f_o * (2.0 - utility(\boldsymbol{a})) \tag{3}$$

The direction factor f_d is larger than one whenever the vehicle moves backwards. The orientation factor f_o is one whenever the vehicle drives straight ahead and larger than one, if not. The last part including the expected utility $utility(\boldsymbol{a})$ is larger than one whenever the density function associates something else than a full intensity with the sampled location a. It can be used to implement the behavior described above.

4. EXPERIMENTS

4.1 Challenges

Evaluating our approach is challenging, since it is a novel variation of the Traveling Salesman Problem. The solution in general cannot be verified in polynomial time, no benchmarks exist and it is not straight forward to compare our approach to the Held-Karp lower bound, since a problem solution contains only a subset of all available graph nodes. The authors belief that the problem is still NP-hard, since it is based on the Traveling Salesman Problem and Next Best View is merely used as a supplemental in order to reduce the set of graph nodes contained in the solution. Since a discussion from the perspective of theoretical computer science would go beyond the scope of this paper, we decided to generate a benchmark in order to demonstrate both the validity of the algorithm for finding suitably short paths as well as the interaction with a LiDAR data set.

4.2 Benchmark datasets

We created two synthetic datasets for the evaluation of our approach. The associated 3D models were created using the 3D modeling software Blender. Cloud Compare was used in order to generate point clouds from the 3D model.

The first synthetic dataset (A) is a room that contains two walls placed in such a way that they resemble a cross. Four points are placed in the corners of said structure so that there is no visual contact between them. The starting point of the vehicle is behind another structure shielding the vehicle from the points (cf. Figure 3(a)). The second synthetic dataset (B) is another room that is divided into nine sections by walls (cf. Figure 3(c)). These walls do not extend to the outer walls so that a vehicle can pass between them. Points of interest are distributed among all sections except the central one. All walls except the northern one are modeled so that there are locations that allow to observe two points of interest at once.

The first dataset has been chosen because it is possible to determine the optimal result in a short time using an exhaustive search. The second data set is significantly more complex. Again, the result is easy to verify, at least for a person. From a computational point of view, the result can be determined in reasonable time, even if the effort is much higher than for the first dataset.

4.3 Examined properties

In order to evaluate the ability of our approach to solve the *Explorational Traveling Salesman Problem*, we use the benchmark datasets described above and compare the results of our approach to the ground truth. We gained these by applying an exhaustive search that checks all valid tours. We discuss the tour proposed by our approach with respect of both the tour optimality and quality. We also examine the impact of determining graph arcs on demand versus determining full graph connectivity in advance. Finally, we comment on the runtimes of our approach.

5. RESULTS AND DISCUSSION

5.0.1 Tour length The tour length of both our approach and the exhaustive search providing the ground truth is shown in Table 1. In order to generate the ground truth, 300 tours for

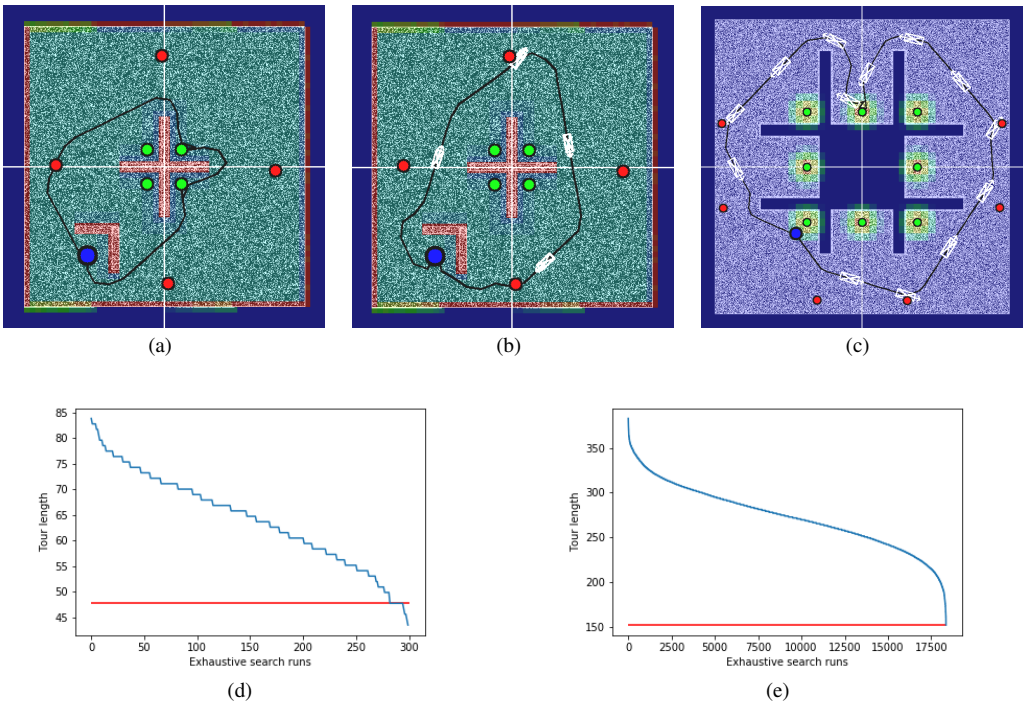

Figure 3. The optimal tour for the first dataset, determined by the exhaustive search (a) and the tour generated by our method (b). Both the optimal tour and the tour generated by our method for the second dataset (c). The start point is blue, points of interest are green and indirect viewpoints are red. The length of all tours examined by the exhaustive search for both datasets (e-f), the red lines show the tours determined by our method.

dataset A and 18354 tours for dataset B have been checked. Figure 3(a)-3(c) illustrate the tours generated by our approach as well as the corresponding ground truth. The length of all possible tours for both datasets can be seen in Figure 3(d) and 3(e), as well as the red line marking the length of the tour provided by our algorithm. In case of the first dataset, our approach determined a tour close to the optimal route, with a length of 47.73 units in comparison to 43.49 units. However, it can be argued that said route is more like the route one would actually take, since it includes more straight forward driving maneuvers. In case of the second dataset, the optimal tour with a length of 151.67 units was found. It should be noted that tour optimization swapped the last two points in the tour in order to get said result. This shows that the heuristic used by our approach does not work in all conceivable cases, but this doesn't differ from many other procedures in the literature. As a side note it should be mentioned that there are two outstanding routes, one clockwise and one counter-clockwise. Due to the vehicle orientation at the start, however, the latter one is actually shorter, since turning the vehicle adds additional length to the path.

The benchmark used here is only intended to prove the functionality of the algorithm, since evaluating a solver for the Traveling Salesman problem is not a trivial task. As stated in (Johnson and McGeoch, 2007), there is no such thing as an optimal benchmark. For a more detailed answer regarding the optimality of the tours generated by our approach, a detailed study of the theoretical properties are required, which is outside the scope of this paper.

5.0.2 Tour quality As can be seen in Figure 3, the tour quality is appropriate with regard to the path planning method used. Both paths are relatively smooth, despite the naive path planning algorithm applied. The vehicle also only drives backwards if it is really necessary, because this behavior is penalized. The paths chosen are not very organic, but that can easily be corrected by applying postprocessing steps such as the ones proposed in (Kurzer, 2016). Collisions are avoided due to checking the collision model.

5.0.3 Impact of path planning on demand The number of arcs in a fully connected graph is $n(n-1)/2$, where n is the number of nodes. In case of the second synthetic dataset with 15 nodes, there are 105 arcs. Starting from an average duration of 74 seconds required for determining the path along two nodes, a total time span of about 130 minutes would be required. Our approach required to calculate 20 paths in order to solve the problem, with an accumulated duration of 24 minutes. Even with a relatively small graph like this one, our method already saves more than 80 % of runtime. And here it has not yet been taken into account that paths would have to be calculated multiple times due to variations in vehicle orientation.

Dataset	Our approach	Ground truth
A	47.73 units (4 points)	43.49 units (4 points)
B	151.67 units (6 points)	151.67 units (6 points)

Table 1. The tour length of both our approach and the ground truth generated by brute forcing all valid paths.

Dataset	Our approach	Ground truth
A	3 min	71 min
B	27 min	12509 min (∼208 h)

Table 2. The runtimes of both our approach and the ground truth generated by an exhaustive search of all valid paths.

5.0.4 Runtimes The runtimes for both our approach and the one using an exhaustive search can be found in Table 2. Path planning is the most decisive factor with regard to runtime. This is especially the case if there is no route and the hybrid A-star algorithm has to do an exhaustive search of its search space. There are methods that are able to carry out path planning in real time (Frazzoli et al., 2002). From the authors' point of view, this is the area in which the most runtime can be saved. If one looks beyond the effort required for the path planning, the runtime for the iterative approach can be compared with that from the literature (Johnson and McGeoch, 2007). Other operations such as selecting the Next Best View and sampling the density functions are neglectable. Visibility calculations also require noticeable runtimes, since they are computationally expensive due to the use of raycasting.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new variation of a well-known problem from computer science we refer to as the **Explorational Traveling Salesman Problem**. We have shown how this theoretical problem can be solved in order to find a suitably short path for a measuring vehicle. In doing so, we demonstrated the usefulness of density functions for performing the spatial calculations required to solve spatial problems. The evaluation of our approach on a synthetic benchmark dataset generated for this purpose shows that it is able to find tours with good length and quality. To apply the approach to large real-world data sets, all that is required is faster path planning. Many optimizations are possible in the context of ongoing work, such as taking drive-by situations into account. The most important aspect, however, is a detailed study of the theoretical properties of the Explorational Traveling Salesman Problem.

REFERENCES

A. J. Hoffman, P. W., 1985. *The Traveling Salesman Problem: A Guide Tour of Combinatorial Optimization*. John Wiley & Sons, 1–15.

Banta, J., Abidi, M., 1996. Autonomous placement of a range sensor for acquisition of optimal 3d models. *Proc. IEEE 22nd Int. Conf. on Industrial Electronics, Control and Instrumentation*, 1583–1588.

Beardwood, J., Halton, J. H., Hammersley, J. M., 1959. The shortest path through many points. *Proceedings of the Cambridge Philosophical Society*, 55(4), 299.

Bentley, J. L., 1992. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(4), 387-411.

Biggs, N., Lloyd, E. K., Wilson, R. J., 1986. *Graph Theory, 1736-1936*. Clarendon Press, USA.

Borůvka, O., 1926. *O jistém problému minimálním (About a certain minimal problem)*. Práce Moravské přírodovědecké společnosti, 3, Mor. přírodovědecká společnost.

Bowyer, K. W., Dyer, C. R., 1990. Aspect graphs: an introduction and survey of recent results. *ISPRS International Conference on Computer Vision and Remote Sensing*.

Clarke, G., Wright, J. W., 1964. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4), 568-581.

David S. Johnson, L. M., 1995. Local optimization and the traveling salesman problem. *Automata, Languages and Programming*, Springer Berlin Heidelberg, 446–461.

Frazzoli, E., Dahleh, M. A., Feron, E., 2002. Realtime motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 116–129.

Gehrung, J., Hebel, M., Arens, M., Stilla, U., 2019. A Representation of MLS Data as a Basis for Terrain Navigability Analysis and Sensor Deployment Planning. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W7, 39–46.

Helsgaun, K., 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1), 106 - 130.

J. Kahn, M. Klawe, D. K., 1980. Traditional Galleries Require Fewer Watchmen. *SIAM. J. on Algebraic and Discrete Methods*, 194-206.

Johnson, D., Gutin, G., McGeoch, L., Yeo, A., Zhang, W., Zverovitch, A., 2007. *The Traveling Salesman Problem and its Variations*. Combinatorial Optimization, 12, Springer, chapter Experimental Analysis of Heuristics for the ATSP, 445–487.

Johnson, D., McGeoch, L., 2007. *The Traveling Salesman Problem and its Variations*. Springer, chapter Experimental Analysis of Heuristics for the STSP, 369–443.

Kurzer, K., 2016. Path planning in unstructured environments : A real-time hybrid a* implementation for fast and deterministic path generation for the kth research concept vehicle. Master's thesis, KTH, Integrated Transport Research Lab, ITRL.

Matai, R., Singh, S., Mittal, M. L., 2010. Traveling salesman problem: an overview of applications, formulations, and solution approaches. D. Davendra (ed.), *Traveling Salesman Problem*, IntechOpen, Rijeka, chapter 1.

Maver, J., Bajcsy, R., 1990. How to decide from the first view where to look next. *Proceedings of the DARPA Image Understandwing Workshop*.

Pudney, C. D., 1994. Surface modelling and surface following for robots equipped with range sensors. PhD thesis, University of Western Australia.

Scott, W. R., Roth, G., Rivest, J.-F., 2003. View Planning for Automated Three-Dimensional Object Reconstruction and Inspection. *ACM Computing Surveys*, 35(1), 64–96.

Tarbox, G., Gottschlich, S., 1995. Planning for Complete Sensor Coverage in Inspection. *Computer Vision and Image Understanding*, 61(1), 84 - 111.

Valenzuela, C. L., Jones, A. J., 1997. Estimating the Held-Karp lower bound for the geometric TSP. *European Journal of Operational Research*, 102(1), 157 - 175.

Whaite, P., Ferrie, F. P., 1991. From uncertainty to visual exploration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10), 1038-1049.

Yuan, X., 1995. A Mechanism of Automatic 3D Object Modeling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(3), 307–311.