2014 International Conference on Indoor Positioning and Indoor Navigation, 27th-30th October 2014

Homotopy and Alternative Routes in Indoor Navigation Scenarios

Martin Werner
Mobile and Distributed Systems Group
Ludwig-Maximilians-University Munich
Munich, Germany
Email: martin.werner@ifi.lmu.de

Sebastian Feld
Mobile and Distributed Systems Group
Ludwig-Maximilians-University Munich
Munich, Germany
Email: sebastian.feld@ifi.lmu.de

***Abstract*—There are lots of innovative use cases possible that build on a shortest route between two locations together with a set of alternatives that are highly different yet short. Imagine, for example, a complicated building like an airport, where passengers can consult a computer terminal to get navigation advices to a desired goal. By scanning the boarding card the calculation of alternatives can be made context-sensitive. Based on gender, amount of time, or shopping preferences, for example, the terminal can display different routes regarding floors traversed, or shops and restaurants passed. Furthermore, the building operator can control the presentation of alternatives in order to influence visitor flows in real time. Therefore, we propose to use the topological concept of homotopy in order to decide if two routes should be considered equivalent or alternative. Basically, the homotopy relation identifies equivalence classes. We propose that a representative of an equivalence class is an alternative regarding another equivalence class. We concatenate the two routes in question and thus, create a polygon. If there is an obstacle inside, the routes are non-homotopic and we consider them as proper alternatives. For this situation, we propose two fundamentally different approaches that are able to find alternative routes with respect to homotopy. The input is a building plan in form of an occupancy grid. Bitmaps allow for fast calculation of the homotopy relation and can be generated from almost any type of environmental model. The first approach aims for enumerating routes that have to visit a special supporting point. This concatenation of two shortest paths leads to alternatives very fast. The second approach is orthogonal to that in the sense that it generates alternatives roughly ordered by their length. Finally, we evaluate and discuss the approaches' feasibility based on different metrics in several scenarios.**

I. Introduction

Finding shortest routes from a specific starting point to a goal has been the topic of research for long and can be handled equally in both outdoor and indoor scenarios by using Dijkstra's algorithm, A^* or optimized versions of these basic techniques. In many applications, however, a user has got more metrics to rate a given route than just the length of the path. If, for example, a path inside a building is slightly longer than the shortest path, but contains only one turn instead of two or more, it is likely that this path is preferred by the user. If, as another example, the shortest path contains two staircases while a slightly longer way does not change floors at all, this is clearly a favorable choice not only for wheelchair users. Thus, the shortest path is often actually not considered to be the best path by a certain user. In this situation, one could try to select a route that suits the user better by adding more metrics, additional constraints, or preferences of the user. The resulting task is then to optimize a multi-objective function in an integrated way. In general, this is very hard, since in many cases it is impossible to compare different paths with respect to the given objectives in a consistent way. In other words: One cannot clearly say whether the shortest or the fastest route is "the best".

For generic navigation, however, it is custom to define a primary objective function, try to maximize it, and integrate further objective functions later in the route finding process. In a first step, we find k *reasonably short paths* which are *sufficiently different* from each other. This is known as finding *alternative routes* or *alternative graphs*. At this point, however, one quickly realizes how hard it is to define *reasonably short* and *sufficiently different*. For vehicle navigation it is customary to achieve *sufficiently different* by exchanging the most important route segment, such as a highway, with a different one and *reasonably short* is then given by the shortest way that uses a different highway.

For indoor navigation scenarios, however, the definition of alternative routes is more difficult, as there is less variation in travel speeds, and navigation in free space leads to lots of different ways of similar length. In other words: If we employ unadapted alternative route algorithms from outdoor scenarios inside buildings, many ways found will be semantically equivalent regarding the fact that they cross the same doors, rooms and free spaces. There are several further differences with respect to outdoor scenarios. As already stated, on the first view all edge weights are roughly the same. But this will change if the system takes elevators or escalators into account. Another aspect is the missing of highways, which can easily be exchanged in outdoor scenarios leading to completely different routes.

Still, there is high utility in calculating and evaluating alternative routes in indoor scenarios as the selection of a path between two points can be personalized with items along the path such as shops or facilities: Imagine an airport and a computer terminal where a user can get navigation advices towards the gate of her flight by scanning the boarding pass. Based on context information like remaining time, gender, or other information the system proposes different routes to follow. Examples are a route that is short and thus fast, a route passing shops mainly targeting women, or a route that passes mostly restaurants. If few information on the person is known, the system could respond with a list of shop logos visible

978-1-4673-8054-6/14/$31.00 ©2014 IEEE

from each alternative route in order to let the person choose by herself. Another use case could be the regulation of person flows. The system can dynamically lead persons through less crowded, alternative ways, if there are bottlenecks.

This paper introduces a novel approach for generating alternative routes using the topological concept of homotopy. Homotopy is an equivalence relation on paths. Roughly speaking, two paths are homotopic if they can be changed into each other without crossing over geometry or being cut in parts. By using this concept, we do not only find alternative routes for a given indoor scenario but we also find different equivalence classes of alternative routes. Inside each of these classes of alternative routes we find a representative path. In order to facilitate both, fast computations of the homotopy relation and space-filling navigation graphs, the input of our implementations is an occupancy grid. Note, however, that triangulations and navigation meshes can be used in this setting as well, if the scalability with respect to the covered space has to be increased.

The main contributions of this paper are: (1) We define alternative routes in indoor scenarios as being non-homotopic with each other and provide a method of calculating this equivalence relation. (2) We implement an indoor version of the well-known *Penalty algorithm* [6], [11], [25] to find alternative routes. (3) We introduce our *One-Patching algorithm* which outperforms the *Penalty algorithm* by means of the number of iterations necessary to find a certain number of alternative routes. The *One-Patching algorithm* can be used in two modes: randomized or grid-based. In the grid-based approach, all equivalence classes of a certain kind can be detected by the cost of a lot of computational overhead. The randomized approach is better used in online scenarios where mutliple classes of alternative routes are needed fast.

The remainder of this paper is as follows: In Section II we review related work in the field of alternative routes, both in outdoor and indoor scenarios. Section III introduces the topological concept of homotopy that we use to tell if two given routes are alternatives to each other or equivalent. Section IV gives a formalized definition of our problem to find alternative routes in indoor scenarios. Our novel *One-Patching algorithm* in randomized and grid-based mode as well as the existing *Penalty algorithm* are explained in Section V, followed by a brief description of our concrete implementation in Section VI. We evaluate and discuss the algorithms with three different scenarios in Section VII. Section VIII concludes the paper and gives hints on future work.

II. Related Work

Finding the shortest path in navigation systems is a central problem to which a lot of solutions have been proposed. The basic algorithms of Dijkstra and Bellman-Ford [7], [12], [15] can be applied to find the shortest path in navigation graphs. It is reasonable to perform shortest path search in occupancy grids also by means of graph search algorithms using the implicit graph in which every free space pixel corresponds to a vertex and two vertices are connected if they are adjacent and unoccupied. Still, these graphs – and many other real-world networks – can become quite large. To handle such large graphs, a lot of optimizations and heuristics for faster search algorithms have been discussed including landmark-based methods [13], [19], graph compression methods [9], and hierarchical approaches [16], [18].

From a user perspective, it is very useful to get a small set of different paths in order to select a subjective best path based on information that are not available to the computer system. Towards the generation of such useful sets of alternatives, the first step is to efficiently find the k shortest paths [26]. The k shortest paths algorithm can be used to filter the set of possible paths ordered by the length in order to refine the result incorporating additional criteria afterwards. However, in street networks and even more in indoor scenarios, the k shortest paths often construct alternatives shortly leaving and re-entering the shortest path. Therefore, the number k has to be pretty large in order to find real alternatives which sufficiently differ from the shortest path rendering this approach impractiable.

Therefore, a lot of research has been devoted to finding metrics for measuring whether two routes are to be considered real alternatives or if they are too similar with respect to each other. Also, algorithms that quickly find short routes with a sufficiently different similarity metric have been proposed [4], [6]. However, these approaches are designed and evaluated in road networks only. There are several baseline approaches which have all been refined with additional heuristics and speedups. One classic example is trying to integrate multiple criteria into the graph search. Pareto paths are calculated by defining a domination relation in which a route dominates another route if it is "better" (e.g., larger or smaller) in all of the given criteria. This leads to multiple Pareto-maximal paths. Pareto-maximal paths are paths that are not dominated by other paths explored by the search algorithm. This can be incorporated into some graph search algorithm like Breadth-First-Search (BFS), Dijkstra's algorithm, or even A^* [20] which create shortest path trees from one vertex to a growing set of vertices [17], [23]. Another approach is given by the Plateau method which works by extracting paths with preferably high conformance out of the intersection of forward search and backward search [1], [22].

Another widely adopted method of finding alternative graphs is given by the Penalty method in which shortest paths are iteratively calculated and after each iteration, the edge weights of the shortest path are being increased such that for the next iteration, the shortest path might change [6], [11], [25]. Note, that defining a proper edge punishment is not a trivial task. A metric measuring dissimilarity of paths is then used in order to keep a list of the k most alternative paths found so far. One advantage of using the Penalty method is that it quickly generates alternative routes and that alternative routes are roughly ordered by their length. As a side-effect this method creates a lot of knowledge about the importance of vertices in the graph for specific pairs of vertices and areas. Thereby, landmark selection and other optimization methods can be guided by the result of searching in the graph. We employ a variant of the Penalty method as a reference algorithm to our new approach to generate alternative routes in indoor scenarios.

The previously described methods are related to our system from a shortest path and graph search perspective. Our geometric approach of deciding whether two paths are alternatives or to be considered equivalent has been used in problems

related to planning and autonomic robotics. One problem is to identify the shortest path that detours obstacles in the same way as a given reference route does [8], [14], [21]. In contrast, we employ similar definitions in order to detour obstacles in *another* way than the routes already found.

III. Homotopy

One of the main challenges in indoor scenarios is to give a definition in order to dedice if two paths r_1 and r_2 shall be considered equivalent, and an efficient method to calculate this relation for explicit instantiations of paths. We use the concept of homotopy, which defines an equivalence relation on the set of paths with equal start and end points. Roughly speaking, two paths are homotopic if they can be deformed into each other in a continuous way. That is, we can move all points of the paths without jumping over obstacles or dissecting the path.

Though homotopy is a very general relation on topological spaces, we restrict ourselves to paths in two-dimensional Euclidian space that are continuous maps

$$\alpha : [0,1] \to \mathbb{R}^2$$

These maps can be interpreted as paths in that the argument $t \in [0,1]$ is seen as a time and the value $\alpha(t)$ is seen as the point in which an object traveling along α resides in at time t. In this setting, two paths α and β having the same start and end points are homotopic, if there is a continuous map

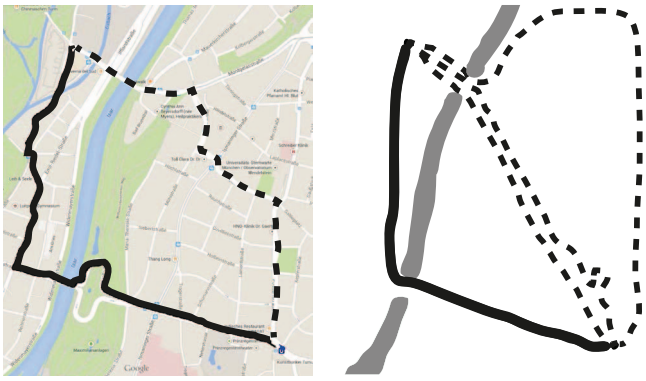
$$\gamma : [0,1] \times [0,1] \to \mathbb{R}^2$$

such that

$$\begin{aligned}
&\gamma(t,0) = \alpha(t),\\
&\gamma(t,1) = \beta(t),\\
&\gamma(0,t) = \alpha(0) = \beta(0), \text{ and}\\
&\gamma(1,t) = \alpha(1) = \beta(1)
\end{aligned}$$

This map is called a homotopy from α to β. In this map, the first argument encodes the time inside both paths and the second argument encodes the time in the process of deforming the first path into the second path fixing the start and endpoint. This definition creates an equivalence relation on the set of all paths connecting two fixed points and thereby a disjoint partitioning of the set into equivalence classes. Each path belongs to one and only one equivalence class. Further information on the concept and background can be found in textbooks on algebraic topology including [5], [10], [24].

Let us further motivate the definition above with an example that starts with an outdoor scenario. Figure 1(a) shows two routes which are intuitively seen as alternative routes. The solid route passes the river via another bridge than the dashed route. If the solid route would have gone underneath the monument close after the river instead of above, one would likely not define this route as an alternative even though both routes are not equal. Now we transform this setting into an indoor scenario where a user can walk freely in space, that means without considering any roads. The river represents an obstacle similar to walls or furniture indoors. Obviously, the three dashed tours in Figure 1(b) are homotopic with each other and therefore no alternative routes to each other while the solid one is an alternative route to all dashed lines.



(a) Outdoor alternative routes (b) Free space alternative routes

Fig. 1. Homotopy as an indicator of route equivalence.

Though it is still unknown how to compute the homotopy relation for even basic topological spaces including spheres of higher dimension, it is simple in two-dimensional Euclidian space for polygonal lines without self-intersection: In order to deform one polygonal line into another, we can consider the polygon spanned by the first path concatenated with the second path in reverse orientation. If the inner of this polygon is empty, i.e. no obstacle is included, then we can clearly deform one bounding line of the polygon into the other one. This is clear for polygons without self-intersections. For polygons having self-intersections, however, polygon filling algorithms fill the right triangles in order to deform one bounding line into the other one. Therefore, we calculate the homotopy relation by checking that the area which would be filled by a scanline polygon filling algorithm is actually empty.

In summary, we use the following formula to define whether two paths α and β with equal start and end are homotopic:

$$p \simeq q \Leftrightarrow \mathrm{Empty}(\mathrm{PolygonArea}(p * q^{-1}))$$

in which $*$ denotes the concatenation of compatible path segments and q^{-1} denotes the inverse path of q that starts with the endpoint and proceeds towards the starting point.

Let Figure 2 be an example for this. One can see that when cutting the concatenation of p and q^{-1} with the set of obstacles O the result is empty (left-hand side). Thus, p and q are homotopic ($p \simeq q$) and consequently no alternative routes to each other. The interior of the concatenation $p*q^{-1}$ of p and q on the right-hand side of Figure 2 is, however, not empty. There is an obstacle between both paths and thus they are non-homotopic and consequently provide alternatives to each other.

IV. Problem Statement

Given a floorplan $A \subset \mathbb{B}^{m \times n}$ in occupancy grid representation in which walkable space is white and obstructions are black – including walls as well as furniture, but excluding doors – and a pair (s,t) of points, we aim to find a set of k preferably short paths $R = \{r_1, ..., r_n\}$ from start s to goal t which are pairwise non-equivalent with respect to the

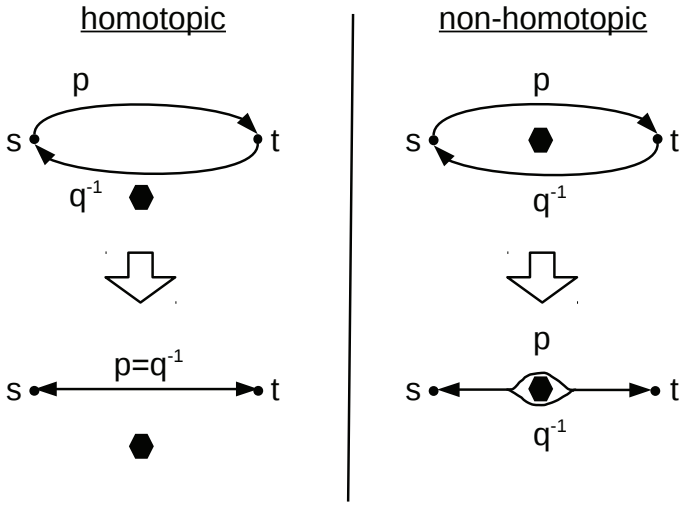


Fig. 2. Example showing our idea of using homotopy to distinguish alternative routes in indoor scenarios. Left-hand side: p and q are homotopic and thus no alternative routes. Right-hand side: p and q are not homotopic and thus represent alternative routes.

Require: $canvas, start, goal$
  $G \leftarrow \text{createGraph}(canvas)$
  $supportPoints \leftarrow \text{getSupportingPoints}(canvas)$
  $\text{performDijkstra}(G, start)$
  $\text{performDijkstra}(G, goal)$
  $path \leftarrow \text{ShortestPath}(start, goal)$
  $homotopyClasses \leftarrow path$
  for all $m \in supportPoints$ **do**
    $q_1 \leftarrow \text{ShortestPath}(start, m)$
    $q_2 \leftarrow \text{ShortestPath}(m, goal)$
    for all $p \in homotopyClasses$ **do**
      $polygon \leftarrow p * q_2^{-1} * q_1^{-1}$
      if $polygon \cap canvas \neq \emptyset$ **then**
        $homotopyClasses \leftarrow q_1 * q_2$
      end if
    end for
  end for

Fig. 3. One-Patching algorithm in pseudo code

equivalence classes induced by homotopy. Furthermore, we are interested in routes without selfintersection.

V. Methodology

In this section, we provide two approaches to the given problem of finding homotopy-alternative routes in indoor navigation scenarios. We start with our proposed One-Patching algorithm and compare this in detail with the Penalty algorithm, which has been widely used for vehicle alternative routing.

A. One-Patching Algorithm

In order to find alternative routes, we are interested in preferably short routes that are pairwise non-homotopic.

Given start s and destination t, we start with the idea of choosing a supporting point m_1 inside the navigable space. Now we calculate a shortest path from s to m_1 and from m_1 to t and concatenate the two routes, i.e.:

$$t_1 = \text{ShortestPath}(s, m_1) * \text{ShortestPath}(m_1, t)$$

The resulting tour t_1 is not a shortest path from s to t, unless m_1 is on a shortest path from s to t. Now we successively create other routes of two shortest path segments by using further supporting points m_k. For each new supporting point, a new path from s to t is generated and the homotopy relation is checked with all prior paths. If it is homotopic to one of the paths previously found, the path is dropped. Note that the homotopic shortest path problem has been solved in literature: It is possible to find the shortest path homotopic to a given reference path in reasonable time [8], [14], [21]. Therefore, the length quality of the paths generated by our algorithm is not a critical factor. We can keep any representative and cleanup our choices in postprocessing, for example, to the shortest path inside a given homotopy class.

We use two strategies in order to define the sequence of supporting points: For an offline analysis of the floorplan given fixed starting and end points, we use a grid-based approach in which each unoccupied point of a regular grid on top of the occupancy grid is used as a supporting point. This, however, amounts in numerous computations depending on the number of pixels skipped. Considering this aspect, we decided to also choose supporting points at random. This leads to quick detection of new homotopy classes, but without guarantees. When choosing two supporting points at random, these points are likely further apart than with the grid-based approach, what in turn increases the chance that the polygon spanned by the corresponding patched routes is not empty and we thus find two paths that are non-homotopic to each other.

Figure 3 summarizes the algorithm in detail. Please note that the two choices regarding supporting points are given behind $\text{getSupportingPoints}$. This can either return a regular grid on top of the canvas object or a set of random points.

A major weakness of the Patching algorithm with just a single supporting point is that it just finds routes that are patchings of two shortest paths. The upper half of Figure 4 gives an example of a "useful" homotopy class that is *not* found. One can see multiple snapshots of a canvas containing a starting point s on the top side, an ending point t on the bottom side as well as two labyrinths. It is obvious, that there are at least five meaningful homotopy classes, namely completely going outside the labyrinth (left and right), going through one labyrinth (top or bottom) as well as going straight through both labyrinths. If our algorithm chooses a supporting point lying inside one of the labyrinths, it calculates the shortest path from s to m lying inside, and then the shortest path from m to t. The labyrinth in which no supporting point resides in, however, will be avoided. Therefore, the homotopy class crossing both labyrinths is unavailable to our One-Patching algorithm as no representative of it is a patching of two shortest paths. In this example, the grid-based One-Patching algorithm just finds four instead of five classes (see the four snapshots in the upper part of Figure 4).

The next algorithm to be explained is capable to find five instead of four meaningful classes of alternative routes in the presented counterexample scenario. This happens, however, at the cost of (1) introducing additional noise to the alternative routes found resulting in the need for post-processing and (2)

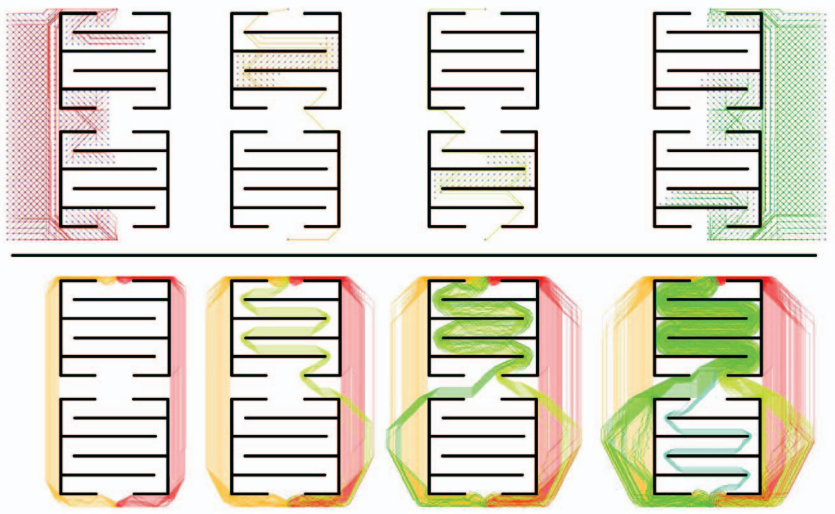


Fig. 4. Top: Counterexample showing that the Patching algorithm using just a single supporting point does not necessarily find all "useful" homotopy classes (in this case the way through both labyrinths). Bottom: Contrary to the One-Patching algorithm, the Penalty algorithm finds five instead of four meaningful homotopy classes.

Require: $canvas, start, goal, iterations$
  $G \leftarrow \text{createGraph}(canvas)$
  $homotopyClasses \leftarrow \emptyset$
  for $i = 1$ to $iterations$ **do**
    $\text{performDijkstra}(G, goal)$
    $t \leftarrow \text{ShortestPath}(start, goal)$
    $\text{edgePenalty}(G, t)$
    if $i == 1$ **then**
      $homotopyClasses \leftarrow t$
    end if
    for all $p \in homotopyClasses$ **do**
      $polygon \leftarrow p * t^{-1}$
      if $polygon \cap canvas \neq \emptyset$ **then**
        $homotopyClasses \leftarrow t$
      end if
    end for
  end for

Fig. 5. Penalty algorithm in pseudo code

much more shortest path calculations, see Section VII.

B. Penalty Algorithm

The Penalty approach for finding alternative routes is quite simple and has been widely used for finding alternative paths in road networks. In order to find routes different from the shortest path, but yet short, one can iterate the shortest path algorithm several times and each time increase the edge weights of the shortest path such that the shortest path gets longer and longer, until the originally second-shortest path becomes the new shortest path. This approach is able to find the five sought-after homotopy classes in the labyrinth example when all shorter paths have been penaltied often enough such that the shortest path finally traverses both labyrinths (see Figure 4 bottom). We sketch the details of the Penalty algorithm in pseudo code in Figure 5.

The most important configuration choice when applying the Penalty algorithm is the amount of penalty and the way in which it is applied to the graph. For a smaller amount of penalty, the number of iterations until a specific alternative route will been found is larger. In extreme cases, the shortest path does not even change for several iterations. For a large amount of penalty, a part of the graph might be penaltied too much and useful hotspots (e.g., bridges, etc.) are overpenaltied and alternatives using the same hotspot cannot be found anymore. For the purpose of our indoor routing applications, we employed a simple strategy of doubling the edge weights. Furthermore, the successful application of this iterative algorithm needs a reasonable criterion to stop the computation. This can be done by the number of alternative routes found, by the area (e.g., number of edges) of the graph which have been explored during search, or the number of homotopy classes found. One drawback of this approach is that each iteration performs a relatively complicated operation: Finding a shortest path. In comparison to the previous approach of patching two shortest paths, however, this approach was able to find the fifth homotopy class crossing both labyrinths (see the four snapshots in the bottom part of Figure 4).

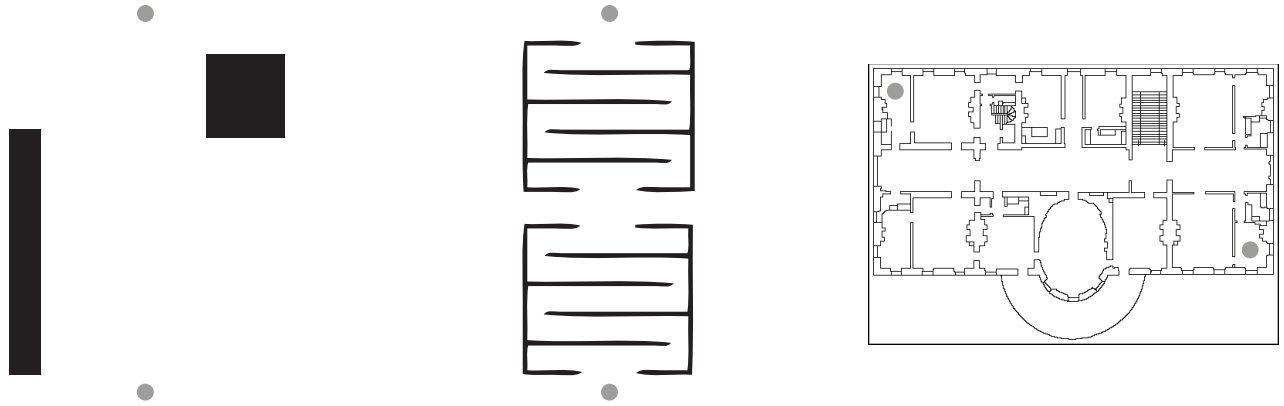
VI. Notes on the Implementation

We implemented both, our One-Patching algorithm and the Penalty algorithm, in order to perform experimental evaluations. We use GNU Octave as a prototyping and evaluation environment. The input to our implementation is a monochrome bitmap containing the building floorplan where the color white represents walkable area and the color black represents walls and obstacles. We also give the starting point and the goal as pixel coordinates.

For performing graph operations, we provide a native extension to Octave built on top of the Boost Graph Library [3] which offers well-established, reviewed, template-based algorithms for graph operations including fast Dijkstra and A^* implementations. The first subtle decision to be made is, whether each pixel shall be a graph vertex or only free-space, white pixel. In the former case, a lot of additional connected components are generated, one for each black pixel.

Scenario	Size [px]	Vertices	Edges	Penalty Iterations	Grid Skip ($\simeq$ Iterations)	Random Tries (Iterations)
Simple	150x150	22,649	163,556	100	15 (100), 20 (49) , 25 (36)}	100 (100)
Labyrinth	390x390	152,489	1,102,252	250	25 (225), 35 (121), 45 (64)	100 (250)
The White House	529x361	191,329	1,372,258	250	26 (260), 36 (140), 46 (77)	100 (250)

TABLE I. DATASETS



(a) Scenario "Simple" (b) Scenario "Labyrinth" (c) Scenario "White House"

Fig. 6. Evaluation Scenarios

In the latter case, the notion of connected components reflects connected components of walkable space and graph algorithms are faster since a lot of vertices can be skipped, for example, for initialization loops. However, graph operations can only be made on modelled vertices. Performing a Dijkstra search starting from a black pixel will typically result in an error, exception or segmentation fault. For adding edges to the graph, we considered all four neighbors above, left, right, and below each white pixel as well as the four diagonals. Edge weights for the horizontal and vertical neighbors have been set to 1 and, consequently, $\sqrt{2}$ for the diagonals. The Octave bindings for these basic graph operations and for creating a graph out of an occupancy grid are available online [2].

As described in Section III, we use a polygon filling approach to test the homotopy relation of two paths p and q with the same start and end point. In order to get a flexible and fast homotopy test based on polygon filling algorithms, we decided to use a generic approach in which a polygon mask is generated by drawing the complete polygon $p * q^{-1}$ into a matrix using 0 for free space and 1 for the inner of the polygon. For testing homotopy, then, the polygonal mask is multiplied pointwise with the inverted floorplan, which contains a 1 for each occluded location and a 0 for walkable space. The pointwise multiplication results in a matrix consisting entirely of zeros unless the inner of the polygon hits non-walkable space. If there are nonzero entries in the result, we conclude $p \not\simeq q$, otherwise $p \simeq q$.

Note that it would have been slightly faster to integrate the testing into the polygon filling algorithm and interrupt the polygon filling once there is an occluded pixel inside the polygon. In this generic way, however, the sum of the entries of the result gives a hint on the number of occupied pixels lying inside the polygon, a possible ingredient to a metric for measuring the dissimilarity of alternative routes.

VII. EXPERIMENTAL RESULTS

We used three scenarios in order to explain the behavior of the algorithms. The first scenario is very simple and shows a basic understanding of how the algorithms perform in comparison to each other. This scenario named "Simple" consists of two blocks leading to three different meaningful homotopy classes for a start point at the top center and an end point at the bottom center of the image given in Figure 6(a). Please note that we consider routes containing loops as not meaningful, since they do not provide real alternatives and are unlikely desired by a user of a navigation system. The second scenario "Labyrinth" – depicted in Figure 6(b) – is designed to show that the One-Patching algorithm's constraint of only finding routes that are made up of two shortest paths is a weakness. The algorithm does not find the route through both labyrinths. Finally, for a realistic scenario, we chose the historical floorplan of the White House depicted in Figure 6(c).

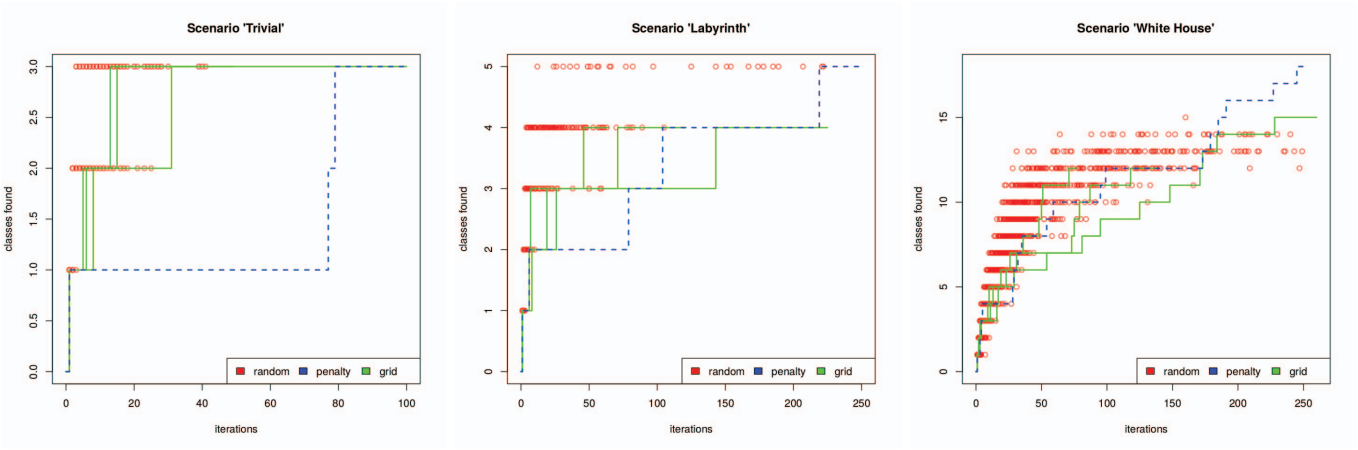
Table I subsumes numbers on the datasets and the algorithms' configuration for the three scenarios which are depicted in Figure 6.

A. Scenario "Simple"

We found that random One-Patching is a very good strategy in simple scenarios like the one depicted in Figure 6(a). But all other algorithms were also able to find the three meaningful homotopy classes (e.g., "left", "middle", and "right") in reasonable time.

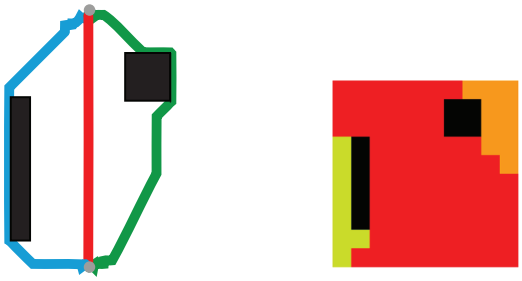
However, the Penalty algorithm has got a high computational overhead and produces artefacts near the beginning and the end of a route where vertices have been penaltied too often leading to a noisy path. On the contrary, the patchings can be unnatural near the chosen supporting point.

A comparison of the computational demand of all three approaches for the simple scenario is depicted in Figure 7(a).



(a) Scenario "Simple" (b) Scenario "Labyrinth" (c) Scenario "White House"

Fig. 7. Performance for different scenarios.



(a) Result of the Penalty-Algorithm (b) Homotopy Map for Grid-based Patchings with step 15

Fig. 8. Results for the Scenario simple

The number of iterations of the algorithms is given on the horizontal axis compared to the number of classes found on the vertical axis.

The Penalty algorithm finds the three meaningful homotopy classes within the given setup. The first class is found, of course, with the first iteration. It is the route right through the two obstacles. The seconds class is found after 77 iterations, shortly followed by the third class after 79 iterations. The long time between class one and two is due to the fact that the map has got a large free space in it's center. Class two and three are found shortly after each other since the length of both alternatives are quite similar. This shows one disadvantage of the Penalty algorithm: The Penalty algorithm has got problems when the length of the next alternative route differs much from the length of the current shortest path.

In order to compare the results just mentioned with the grid-based One-Patching algorithm, we chose to use roughly the same number of iterations reflected in choosing the skip values as given in Table I. Since the grid-based One-Patching algorithm outperforms the Penalty algorithm in most cases, we decided to make additional test runs with fewer iterations. In all three setups the algorithm finds the second class extremely fast, namely after 8, 6, and 5 iterations. The third class is found fast as well, due to the obstacles that appear after few lines of the grid. See Figure 8(b) for a visualization of the classes found with a skip of 15. In this figure, the supporting points of the One-Patching algorithm are given the same color if and only if they lead to homotopic patchings of shortest paths.

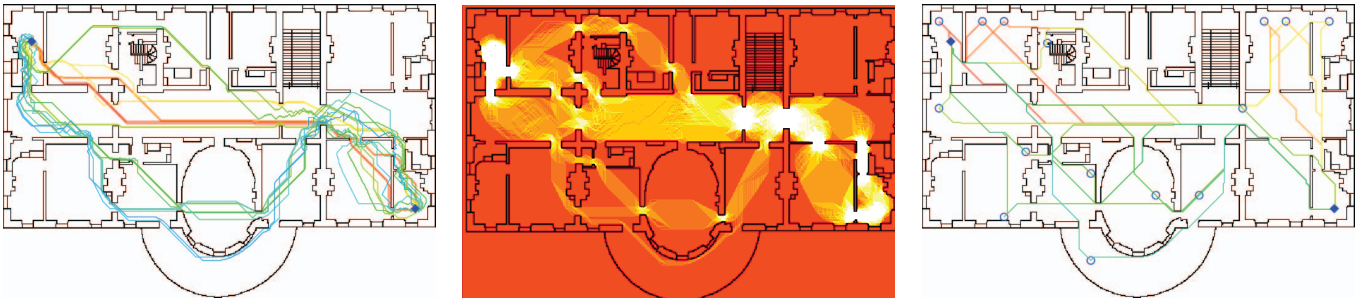
Finally, we analyzed the behaviour of the randomized One-Patching algorithm. We performed the random mode 100 times with 100 iterations and found three classes in every run. It took 13.7 iterations on average for finding three classes ($\text{median} = 12, \min = 3, \max = 41$). In contrast to the Penalty algorithm, the second class is found extremely fast after 5.8 iterations on average ($\text{median} = 4, \min = 2, \max = 25$). Note that for the random case, the first iteration does not always produce a homotopy class since the chosen supporting point can be a black pixel (i.e. non-walkable) leading to no route at all.

B. *Scenario "Labyrinth"*

The scenario "Labyrinth" depicted in Figure 6(b) has been designed as a counterexample to the One-Patching approach and contains at least the following nine homotopy classes:

- C1: Detour both labyrinths on the left
- C2: Detour both labyrinths on the right
- C3: Traverse the 1^{st}, detour the 2^{nd} on the left
- C4: Traverse the 1^{st}, detour the 2^{nd} on the right
- C5: Detour the 1^{st} on the left, traverse the 2^{nd}
- C6: Detour the 1^{st} on the right, traverse the 2^{nd}
- C7: Traverse both labyrinths
- C8: Detour the 1^{st} on the left, detour the 2^{nd} on the right
- C9: Detour the 1^{st} on the right, detour the 2^{nd} on the left

The Penalty algorithm finds five of the named homotopy classes within the given iteration bounds of 250: The first class found is C1 after one iteration closely followed by class C2 after six iterations. The next classes are C3 and C4 after 79 and 104 iterations, respectively. This means, that the routes traverse the first labyrinth but then detour the second labyrinth



(a) Penalty (250 iterations) (b) Penalty Weightmap (250 iterations) (c) Grid-based One-patching (260 iterations)

Fig. 9. Results of the scenario "White House"

on the left/right. The symmetric classes C5 and C6 are not found (detouring the first labyrinth and traverse the second one). After 219 iterations the Penalty algorithm finds class C7 which goes through both labyrinths. This is remarkable, since the One-Patching algorithm is not able to find this homotopy class: It is not a patching of two shortest paths. However, the Penalty algorithm does not find the S-shaped classes C8 and C9.

With respect to the performance, Figure 7(b) gives the results and is similar to the findings before: The grid-based One-Patching approach finds homotopy classes with slightly less iterations than the Penalty algorithm. However, it does only find four classes (C1, C2, C4, C5). The randomized One-Patching algorithm finds different classes very quick, again. However, in 72% of the cases, it did only find four classes (C1, C2, C4, C5) and five classes for the remaining 28% of runs (besides the already stated classes one of the S-shaped classes C8 or C9). It took 104.8 iterations on average for finding five classes ($\text{median} = 79.5, \min = 12, \max = 222$). However, as can be clearly seen, the random One-Patching algorithm was very fast in finding the first classes. Namely, four classes in 31.2 iterations on average ($\text{median} = 24, \min = 4, \max = 105$), three classes in 11.2 iterations on average ($\text{median} = 7$), and two classes after 2.8 iterations on average ($\text{median} = 2$).

C. Scenario "White House"

Finally, we provide a realworld scenario using a simplified version of a historical floor plan of the White House, see Figure 6(c). Starting point is the room in the top left corner, goal is the bottom right room. The Penalty algorithm finds 18 different homotopy classes in the given iteration bounds of 250. The first 12 classes are found within 100 iterations. These are the routes going through the upper rooms and the horizontal hallway, see Figure 6(c). After a gap of about 70 iterations the algorithm finds six more homotopy classes. These are the routes going through the oval office and the balcony. It is quite interesting to look at the weight map the algorithm created containing the penalized edge weights given in Figure 9(b). One can easily see that there exist more alternatives to go in the left part of the building than on the right side. In fact, there is also a door that absolutely has to be passed. With respect to the aforementioned noise problems in alternative routes generated with the Penalty algorithm, Figure 9(a) depicts the results of the White House example in which especially the right side of the building has got "noisy" routes.

The grid-based One-Patching approach was able to find 15 homotopy classes using approximately the same number of iterations and still 12 classes for the other configurations of Table I. The random One-Patching algorithm was performed 100 times each time with 250 iterations. It was able to find 12 classes with only 116.6 iterations on average ($\text{median} = 109, \min = 35, \max = 247$). These performance information is also depicted in Figure 7(c), which shows how much the random One-Patching algorithm outperforms the Penalty algorithm in finding alternative routes in this scenario.

VIII. Conclusion & Future Work

In this paper, we have proposed a novel approach to find alternative routes especially for indoor navigation scenarios. Our approach exploits the fact that in indoor scenarios many ways exist which differ only slightly from each other. In order to define when two different paths should be considered equivalent or alternative, we proposed to use the concept of homotopy and defined an efficient way to approximate the homotopy relation in our setting. With this definition in place, we implemented the Penalty algorithm as a reference algorithm from the domain of navigation in road networks and showed that our approach of guiding the search of alternative routes with a single supporting point outperforms the classical approach in the amount of graph search operations needed to find reasonable alternatives. In performance-critical scenarios, the information for patching shortest paths can be calculated beforehand by solving an all pair shortest path problem, possibly after simplifying the graph. Then, the calculation of new candidates would become very easy.

With respect to the results of both, our new approach as well as the Penalty algorithm, we find the need for post-processing. The Penalty algorithm destroys the homogenity of indoor graphs and produces noisy paths after some iterations, the One-Patching algorithm constructs senseless walks towards a supporting point and back towards the main track of the route. In both cases, we propose to use the homotopic shortest path as a representative without these artefacts which can be calculated efficiently [14], [21]. The One-Patching approach, however, generates new candidates for homotopy classes which consist of patchings of two shortest ways. In this case, it is also possible to keep the shortest representative found for this path during search. For the Penalty algorithm this is not possible, since the changes made to the weighting lead to the case that all instances generated later in the iteration suffer from noisy movement.

In summary, we have shown how to define alternative ways in indoor navigation scenarios, how to find alternative ways quickly, and how to index a map using the grid-based One-Patching algorithm in order to tell which supporting points create the same or different homotopy class. For future work, we envision to integrate the concept of alternative graphs towards higher and lower layers: We think about integrating the concept of homotopic shortest paths and the One-Patching algorithm into the position filtering stage of indoor navigation systems as it provides additional and more profound knowledge compared to simple map matching. Towards upper layers, we would like to provide the k most probable ways or the k most sensible choices for indoor navigation in order to allow multi-objective optimization and personalized indoor location-based services. Another area of improvement is given by refining the definition of alternative ways in a metric way. Measuring the amount of difference of alternatives might become essential in advanced location-based services and flexible rankings of the alternative routes found based on their geometry (e.g. number of turns, smoothness, etc.) should be provided.

REFERENCES

[1] Cambridge Vehicle Information Technology Ltd. - Choice Routing. Online, 2013. http://www.camvit.com/camvit-technical-english/Camvit-Choice-Routing-Explanation-english.pdf.

[2] Graph Search Using Boost Graph Library in GNU Octave. Online, 2014. http://trajectorycomputing.com/software/graph-search-boost-graph-library-gnu-octave/.

[3] The Boost Graph Library (BGL). Online, 2014. http://www.boost.org/doc/libs/1_55_0/libs/graph/doc/index.html.

[4] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Alternative routes in road networks. *Journal of Experimental Algorithmics (JEA)*, 18(1):1–3, 2013.

[5] Mark Anthony Armstrong. *Basic topology*. Springer, 1983.

[6] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 21–32. Springer, 2011.

[7] Richard Bellman. On a routing problem. Technical report, DTIC Document, 1956.

[8] Sergei Bespamyatnikh. Computing homotopic shortest paths in the plane. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 609–617. Society for Industrial and Applied Mathematics, 2003.

[9] Adi Botea and Daniel Harabor. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2013.

[10] Glen E Bredon. *Topology and geometry*, volume 139. Springer, 1993.

[11] Yanyan Chen, Michael GH Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *Intelligent Transportation Systems, IEEE Transactions on*, 8(1):14–20, 2007.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[13] Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, page 25. ACM, 2013.

[14] Alon Efrat, Stephen G Kobourov, and Anna Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry*, 35(3):162–172, 2006.

[15] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton Princeton University Press, 1962.

[16] Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments (ALENEX13)*, pages 31–54. SIAM, 2013.

[17] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *ALENEX*, volume 10, pages 124–137, 2010.

[18] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[19] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[20] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[21] John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational geometry*, 4(2):63–97, 1994.

[22] Dennis Luxen and Dennis Schieferdecker. Candidate sets for alternative routes in road networks. In *Experimental Algorithms*, pages 260–270. Springer, 2012.

[23] Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

[24] James R Munkres. *Topology: a first course*, volume 23. Prentice-Hall Englewood Cliffs, NJ, 1975.

[25] Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *Experimental Algorithms*, pages 66–79. Springer, 2007.

[26] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.