Maciej Nazarczuk[1]
ORCID: 0009-0008-6559-7000

Artur Niewiadomski[2]
ORCID: 0000-0002-9652-5092

University of Siedlce
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

{[1]maciej.nazarczuk,[2]artur.niewiadomski}@uws.edu.pl

A Pathfinding Module for the Indoor Navigation System NaviSecure

DOI: 10.34739/si.2024.31.10

Abstract. NaviSecure is an indoor navigation system developed at University of Siedlce. It utilizes a dedicated hierarchical building map and an infrastructure of Bluetooth Low Energy transmitters, as well as hazard detectors such as smoke, flood, and gas sensors. The system facilitates daily navigation and provides emergency assistance while addressing the needs of individuals with disabilities. In this paper we present the pathfinding module of the NaviSecure system, implemented on top of the Neo4j graph database. We introduce the basic concepts behind our approach, we discuss the system architecture and our custom approach to deal with special needs of the users. Finally, we show the results of our experimental evaluation of several graph algorithms, modified to meet NaviSecure's specific requirements. The results confirm the efficiency of our approach.

Keywords: Indoor Navigation, Pathfinding, Accessibility, Dijkstra Algorithm, A* Algorithm, Graph Database, Neo4J.

1 Introduction

Due to the advancement of technology, with increasingly smaller and more powerful mobile devices, GPS satellite signals, and wireless communication, navigation systems have become ubiquitous and used daily by drivers, cyclists, pedestrians, and public transportation users. However, indoor navigation in large buildings - such as shopping centers, universities, or other public facilities - remains a challenge, as individuals may experience significant difficulties locating and accessing areas of interest. These challenges are even more pronounced for individuals with disabilities, posing serious obstacles that can limit their independence.

For new students, the transition to university life is accompanied by a range of logistical and educational challenges. They often experience a sense of disorientation as they navigate a novel environment, grapple with unfamiliar subjects and learning methodologies, adapt to different work organization than what they experienced in secondary school, and encounter physical obstacles. For example, visually impaired students encounter difficulties in independently locating lecture halls or navigating from their dormitories to classes.

Universities currently offer some assistance to disabled students through specialized offices, providing support during their initial steps and throughout their academic journey. However, these solutions heavily rely upon staff participation and time, which leads to increased costs. In addition, existing support models can inadvertently create discomfort and dependency for people with disabilities, who may find the constant need to ask for help burdensome and counterproductive to the promotion of an independent learning environment. To address some of these challenges more effectively, implementing automated solutions to support indoor navigation would prove highly beneficial.

Additionally, large facilities are exposed to various threats that require immediate evacuation, such as bomb attacks (including frequent false reports of such attacks), fires, gas leaks, etc. As a result, there has been a growing demand for the development of precise and reliable systems to support people with disabilities in indoor navigation and evacuation.

In contrast to the outdoor environment, where GPS-based navigation has become widespread, indoor spaces lack the appropriate satellite signals for accurate positioning, rendering traditional GPS methods inadequate. Some of the most common techniques for indoor localization [43] are: Wi-Fi-based [17], and exploiting Bluetooth Low Energy (BLE) beacons [7, 17] either using multilateration [15] or fingerprint [25, 36–38] approach.

NaviSecure [36–38] is an indoor navigation system under development at Siedlce University. It makes use of a dedicated hierarchical map and integrates BLE transmitters, and hazard detectors, like smoke, flood, and gas sensors. The system facilitates daily navigation and provides emergency assistance while addressing the needs of individuals with diverse disabilities

In this paper we present the concepts behind the NaviSecure pathfinding module. In particular, we focus on the algorithms and tools integrated and customized in order to calculate an optimal path, taking into account the user's special needs and preferences stored in the system database as the user's profile. The main idea is as follows. The building map is converted into a graph representation in which the vertices represent physical locations in the building and the edges, labelled with weights (e.g., distance), connect neighboring locations. By handling special user needs in the way presented later in this paper, we have reduced the

route finding problem to the classical problem of finding a path in a graph with minimal sum of weights.

Thus, the main contribution of the paper is a novel pathfinding approach for indoor navigation that dynamically adapts to user-specific needs and preferences, implemented on top of the graph database Neo4j [27]. We show how to adapt the well-known Dijkstra's [12] and A* [19] algorithms, and use the efficient implementation offered by the Graph Data Science (GDS) [2] library. The presented experimental results are scalable enough to use this approach as a pathfinding engine in the NaviSecure system.

The rest of the paper is organized as follows. First, we review related approaches and compare them with our own. The next section describes the main concepts of the NaviSecure system, especially the pathfinding routines. Then, we present the experimental results, followed by the conclusions and plans for future work.

2 Related Works

To contextualize the current state of the art in accessible navigation for indoor environments, we draw upon a comprehensive systematic mapping study [43]. This study offers valuable insights into the various technologies and solutions currently available, their effectiveness in supporting individuals with disabilities in navigating both indoor and outdoor spaces and the potential areas for improvement and innovation in this domain. The authors of [43] selected 111 papers out of a larger pool published between 2009 and 2020 and analyzed them exploring multiple dimensions of the proposed solutions, such as context of use, and target users.

Only 19 out of 111 selected papers consider the needs of both fully able individuals and those with varying disabilities. Moreover, only 11 papers consider indoor environments, which have many unique constraints - such as the inability to exploit the GPS technology available on most of mobile devices or having to deal with areas stacked on top of each other, like multi-story buildings. A general conclusion can be drawn on the presently available solutions. Most of them exhibit at least one of the following limitations: either they concentrate solely on catering to the needs of a particular user group with specific impairments, or they restrict their functionality to supporting outdoor navigation exclusively.

For example NavCog [7] is a smartphone-based navigation system dedicated to blind users. It requires a pre-prepared map of the environment for navigation assistance. To build the map, a floor plan is uploaded to the map server and a web-based editor is used to mark, walkable areas, decision points, and points of interest. The map is then downloaded to the user's smartphone, where, unlike our proposed solution, all pathfinding computations are done locally. Similarly to our approach, it makes use of a network of BLE beacons to localize the user, but it uses an approach based on the K-nearest neighbor (KNN) algorithm, while we exploit a hybrid of multilateration and fingerprint-based approach [36, 38, 37].

Certain solutions are addressing the issue of swift evacuation during emergencies, however, at the expense of accommodating specific user's needs. GoFast [17] takes into consideration the user groups, i.e., all users who are near a specific BLE beacon. The system is focused on the comprehensive evacuation of the entire building, while not taking into consideration the special needs of individuals with disabilities. Both systems, NaviSecure and GoFast, offer evacuation capabilities, while GoFast lacks support for individualized navigation assistance.

InLoc [15] is based on raster maps of the building. Subsequently, using standard edge detection techniques, a polygonal map is generated. Based on this map, a grid is created, which is then used for pathfinding purposes. The system employs a method for fusing location data from phone IMU sensors and BLE beacons. Additionally, it introduces an approach to estimate the distance from BLE beacons using RSSI measurements. InLoc does not address the needs of people with disabilities.

The approach conceptually closest to ours is [44], since they also propose a solution involving a user profile in path calculations to address a variety of impairments, however, the paper offers only a conceptual overview and preliminary implementation insights. It is based on freely available Open Street Map (OSM) [5, 22] data provided by the community, while we are using manually prepared building model. The authors of [44] exploit and modify the GraphHopper [3] routing library (which uses OSM data), while we propose extending Neo4J[1, 24, 47] Graph Data Science (GDS) [2, 40] library.

Another solution, which utilizes OSM and community efforts is FootPath [31]. It uses extensively the accelerometer and compass available in modern smartphones. To deal with inaccuracies in step detection and heading estimation, the system utilizes a sequence of alignment algorithms from the field of bioinformatics. However, disadvantages of this approach include the limited OSM support for multi-story buildings with high room density. As presented, there are many competing approaches to the building modeling, user localization, or calculating optimal paths, each with its own upsides and drawbacks. We decided to employ a hierarchical building map representation created manually, or with CAD tools support.

Pathfinding primarily relies on well-established algorithms, notably Dijkstra's and A*, as highlighted in [8]. Additionally, algorithms such as Flexible Path Planning [30], Elastic algorithms [26], D* [35], and HCTNav [41] are used to find the shortest paths in indoor environments. Most research focuses on identifying the shortest path within indoor environments. However, a smaller body of work explores optimizing paths for low power consumption, user preferences, and minimizing the number of turns [8]. Some papers explore different approaches, such as the Ant Colony Optimization algorithm, which has been shown to outperform the well-known A* algorithm in certain scenarios [33]. Alternative algorithms are occasionally employed for specialized applications. For instance, GoFast [17], designed for rapid evacuation, uses algorithms from [16] to efficiently distribute evacuation loads among exits, optimizing load balance and accurately estimating evacuation time. Specific systems illustrate the varied applications of these algorithms. NavCog [7], for example, utilizes Dijkstra's algorithm to aid individuals who are blind. In contrast, systems like InLoc [15] are designed for general pathfinding purposes and do not account for disabilities.

The literature study shows that in static environments with well-defined maps the classical algorithms such as Dijkstra's and A* remain fundamental to pathfinding. Recent advancements include adaptive weight functions and bidirectional search techniques [34], making them a staple for global path planning. However, different approaches to finding the path are used for other environments. For example, sampling-based algorithms, such as the Probabilistic Roadmap (PRM) and Rapidly Exploring Random Tree (RRT), are used in high-dimensional and complex environments. PRM is effective for static settings with numerous obstacles by generating a graph of sampled points and connecting feasible paths [20]. RRT, on the other hand, is particularly suited for dynamic environments, as it explores space randomly to build

a tree connecting the start and goal configurations. While these methods are computationally efficient, they may produce suboptimal paths without additional refinement.

Reinforcement Learning has gained prominence for its ability to adapt to dynamic and previously unseen environments. By leveraging deep neural networks, RL-based approaches can learn optimal navigation policies through trial-and-error interactions with the environment. Gao et al. [20] introduced a hybrid method combining PRM with RL using Twin Delayed Deep Deterministic Policy Gradients (TD3), achieving robust performance across diverse indoor scenarios. Simultaneous Localization and Mapping (SLAM) technologies complement pathfinding algorithms by providing real-time mapping and localization capabilities. SLAM enables robots to navigate in previously unmapped environments by combining sensory data to construct a map while localizing themselves within it. Classical algorithms like A* and RRT are often integrated with SLAM frameworks to enhance real-time navigation [20].

Recent works also shows growing attention to hierarchical or detailed 3D mapping for complex buildings. For instance, Singh et al. [45] introduced an approach that combines 3D LiDAR-based room modeling with dynamic path planning, useful for environments that undergo frequent changes. Additionally, Leitch et al. [29] demonstrate how combining Ultra-Wide Bandwidth Radio (UWB) or BLE-based localization with detailed building geometry can improve the accuracy of user tracking, thus enhancing real-time indoor navigation systems. Deng et al. [18] explored how real-time hazard prediction (e.g., fire spread) can update the cost of edges in a graph, providing rapid evacuation routes. These studies highlight the potential for real-time and adaptive route computation, which is especially relevant for emergency situations or supporting users with disabilities who may require personalized routes.

Pathfinding in indoor environments is an active area of research with diverse applications. Classical algorithms provide robust solutions for static scenarios, while sampling-based and reinforcement learning approaches offer flexibility and adaptability in dynamic environments. By combining these methodologies, hybrid approaches represent a promising direction for achieving efficient and reliable indoor navigation. While existing systems have unique design goals and varying levels of support for disabilities, they often focus on specific user groups. Our approach, however, provides a more generalized and adaptive solution.

3 Pathfinding in the NaviSecure System

Pathfinding is one of the key functions of indoor navigation applications, along with positioning. Supporting routes that take into account the special needs of the user is crucial to providing a safe and accessible navigation system. For example, consider an emergency evacuation scenario. While guiding able-bodied individuals to a safe exit is relatively straightforward, offering the same service to people with disabilities is significantly more challenging.

We have to take into account that users in wheelchairs may not be able to climb stairs, that the emergency may cause certain elevators to stop working, or that there may be debris that can be walked around but blocks the path of a person in a wheelchair. The system must be able to assist all individuals, regardless of their disability status, by guiding them to their destination and notifying security for assistance if necessary.

In this section, we show how to adapt the conventional pathfinding algorithms, such as Dijkstra [12] and A* [14, 19], in order to address the special needs of people with disabilities.

First of all, we need to represent the navigation-relevant building elements as a weighted graph. NaviSecure makes use of a hierarchical building map which is augmented with a grid of walkable locations, called places. The places become vertices of the building's navigation graph. We discuss this process in detail in the next two subsections.

A straightforward approach exploiting the off-the-shelf graph algorithms would require a substantial overhead in terms of the number of edges in the navigation graph. For every possible combination of user requirements affecting the navigation capabilities, it would be necessary to duplicate the edges between every neighboring pair of vertices and assign them different weights and labels. For example, one edge for able-bodied individuals, another for wheelchair users, an extra one for people with visual impairments, etc. Since NaviSecure is very flexible in configuring disability preferences, the number of possible combinations is huge what makes the direct use of the ready-made algorithms impractical.

This paper presents a solution to this problem: a modification of the A* algorithm that aims to address the challenges of indoor navigation for individuals with diverse special needs. By incorporating the user's profile, which encompasses specific information about their disabilities and requirements, the algorithm can dynamically adapt the pathfinding process to provide more personalized and accessible routes. The user's profile includes details such as mobility impairments, sensory limitations, and any other relevant factors that affect their navigation capabilities. By taking this information into account during the pathfinding process, the algorithm can avoid routes that may cause difficulties for the user and prioritize paths that meet their unique needs. By incorporating the user's profile, we can deliver a more inclusive solution that significantly enhances the accessibility and usability of the NaviSecure indoor navigation system for individuals with special needs.

3.1 Hierarchical Building Map

In our approach the hierarchical structure of a building relies on two basic concepts: ROOM, and LINK [36–38]. A ROOM instance corresponds to a cuboid space occupied by a fragment of the building. Its attributes include: coordinates, parent, children (rooms or links), and room type. We consider the following ROOM types corresponding to:

- Building – the whole building,
- Wing – a part of the building,
- Floor – one story, a level inside a wing,
- Room – a room, or its part on the particular floor. Room are indivisible pieces of the map connected by Links. They come in several kinds according to their purposes, like corridor, classroom, toilet, stairs, elevator, social, unavailable, etc.

Thus, the room hierarchy strictly follows the inclusion relation of represented cuboids.

Links connect adjoining rooms and also comes in different kinds, like, e.g, door, passage, stairs, elevator door, locked. It facilitates computing the optimal routes according to the user profile, for example, to avoid stairs by the wheelchair users, or take into account additional delay due to waiting for an elevator.

The process of creating a building map with rooms and links is supported by CAD software, which captures the layout and dimensions of rooms and spaces within (for more details, see [46]). This building model is then saved into a relational database. From there, the

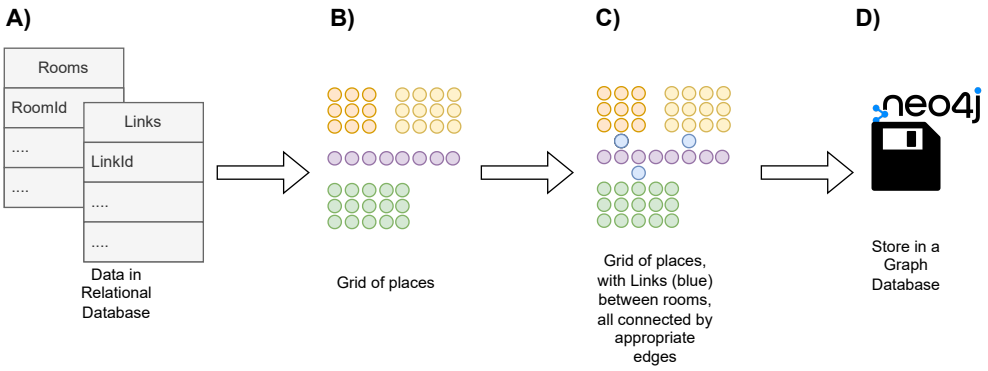

Figure 1. Building data processing scheme. Source: own study.

next steps of data preparation for pathfinding are shown in Figure 1. We start with rooms and links in a relational database (A). Next, we create a grid of places covering the rooms using the following algorithm.

1. **Generate Grid of Places**:
   - For each room:
     - Calculate the number of places within the room.
     - Determine the horizontal and vertical offsets needed for alignment.
     - Create a 2D array to represent the arrangement of places within the room.
     - Compute the places' coordinates.
   - For each place in a room:
     - Save its representation as a vertex in the graph database
2. **Build Navigation Graph**:
   - For each place in each room:
     - Create edges between adjacent places.
3. **Connect Rooms**:
   - For each link connecting rooms:
     - Calculate the arrangement of places within the link.
     - For each place in the link, store it as a vertex in the graph database.
     - Create undirected edges between each place in the link and the closest place in each connected room

Algorithm 1: Navigation graph construction

To generate the grid of places, we go through each room and first calculate the number of places based on the room's dimensions, coordinates, and how tightly the grid is arranged. We also consider how the room is rotated compared to its parent (see Figure 1, step B, Algorithm 1 step 1). We start by making a list of places for each room. Then, we connect the places that are

next to each other. These connections become the edges of the graph database. (Algorithm 1 step 2).

After creating all the nodes and edges for each room, the next essential step is to add links between rooms (see Figure 1, step C, Algorithm 1 step 3). The size, position and rotation of a link element is represented similarly to that of a room. Finally, the graph is stored in the Neo4j [27] database, saving additional navigation-relevant information as auxiliary attributes of edges and vertices (see Fig. 1, step D). For more details the interested reader is referred to [39].

3.2 Implementation

The GDS library [2, 40] offers highly optimized implementations of many popular graph algorithms, including pathfinding algorithms like Dijkstra's and A*. However, due to the requirements concerning support for individuals with disabilities, the provided implementation was found to be inadequate. Fortunately, both Neo4J and GDS are highly extensible, allowing for the creation of custom plugins and providing detailed documentation to facilitate this process. GDS library is licensed under the open GPL licence, allowing to view and modify the source code, if needed.

The classic A* algorithm [14, 19, 23] aims at finding a path in the graph with the smallest cost[1] from a specific starting vertex to a given goal vertex. The algorithm builds a tree of paths originating at the start vertex, incrementally extending these paths one edge at a time until it reaches the goal or satisfies its termination criterion[2]. During each iteration of its main loop, A* algorithm faces the decision of which path to extend further. It makes this choice based on two key factors: the current cost of the path and an estimate of the remaining cost required to reach the goal. To achieve this, A* evaluates and selects the path that minimizes the function given in Eq. 1, where n is the next considered vertex, $g(n)$ is the cost of the path from the start vertex to n and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal.

$$f(n) = g(n) + h(n) \tag{1}$$

To address the special needs of the impaired users we propose to replace it with a modified function (see Eq. 2) which takes into account also the user profile to dynamically adjust the cost and heuristic value in accordance with the requirements:

$$f(n, p) = g(n, p) + h(n) + h'(n, p) \tag{2}$$

where p is the user profile, $g(n, p)$ is the cost of the path from the start vertex to n taking into account the user's special needs[3], $h(n)$ has the same meaning as before, and the value of

[1] E.g., the least distance travelled, the shortest time

[2] The A* algorithm uses an "open list" to keep track of the nodes it still needs to explore. If the open list becomes empty, it means that there are no more nodes to explore, and the algorithm cannot find a path from the starting vertex to the goal vertex. In this case, the algorithm terminates without finding a solution.

[3] It can be infinite if no path accommodating the user exists.

$h'(n, p)$ estimates the difficulty of the route considering the user's profile:

$$h'(n,p) = \begin{cases} 0 & \text{for perfectly suitable path,} \\ Penalty & \text{for paths violating the user preferences.} \end{cases} \tag{3}$$

where $Penalty$ stands for a punishment value for inadequate paths. We recommend to set it several orders of magnitude greater than costs of ordinary paths [4].

This approach allows us, for example, to guide a wheelchair user to an elevator even if the staircase path is technically shorter because the second path violates the user's preferences. Figure 2 shows two alternative paths starting on the ground floor and leading to an upper floor: the red path takes the stairs, while the blue path uses an elevator.

To implement our user-centric pathfinding approach, we developed a custom plugin for Neo4j. This strategy not only extends the powerful capabilities of the Graph Data Science library, but also leverages its robust and efficient graph algorithm infrastructure while integrating enhancements specifically designed to address the unique user requirements discussed above. We took advantage of the inherent extensibility of the GDS library by introducing our own implementations for key interfaces, specifically HeuristicFunction and RelationshipFilter. The HeuristicFunction is instrumental in dictating the order in which vertices are processed. Conversely, RelationshipFilter is used to cut the algorithm's search space. This optimization is achieved by cleverly filtering relationships within the graph, thereby deliberately excluding parts of the building that are definitely outside the optimal path. Our solution is seamlessly integrated with Neo4j, making the implemented functionalities accessible through the Cypher query language. For more information and examples please refer to [39].

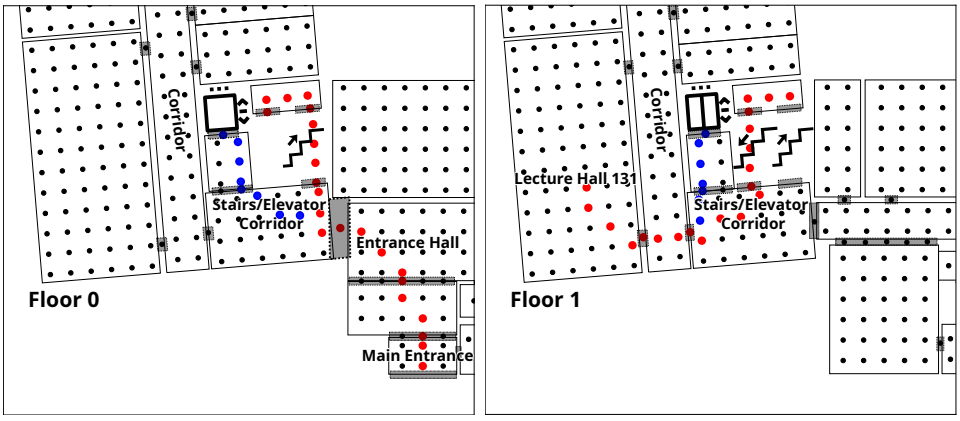

Figure 2. Two alternative paths from the Main Entrance to the Lecture Hall 131, based on user's profile - red stairs allowed, blue - stairs not allowed. Links between Rooms are marked in gray. Source: own study.

[4] In our experiments we used $Penalty = 10^{10}$. The value was set basing on initial experiments.

4 Evaluation

We tested our solution using a real model of the Faculty of Exact Sciences, University of Siedlce building. We compared the performance of the pathfinding service using 3 different heuristics for A* based on Euclidean distance, squared Euclidean distance, and Manhattan Distance[5] [42], and, for the reference, without any heuristics.

To begin the evaluation, we generated a dataset of 200 random pairs of rooms within the building, which are connected by a path no shorter than 10 meters. We plotted this information on a histogram (Figure 3) to visually depict the distribution of pathway lengths. The largest is the set of paths between 78 and 95 meters long. The average length is 59.54m, the median equals 63.68 m, while the standard deviation is 28.2 meters.

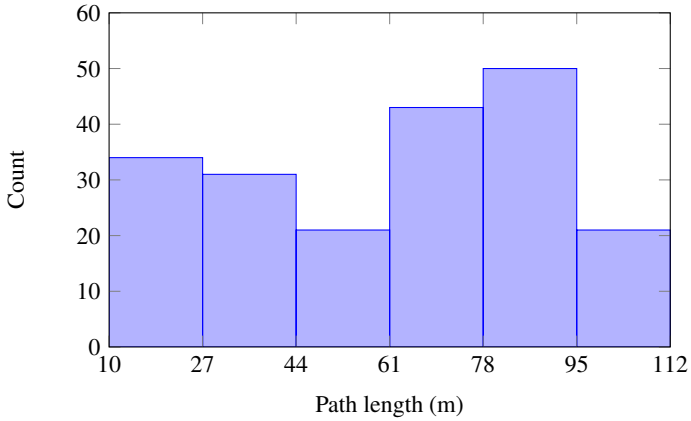

Figure 3. The distribution of path lengths used in the experiments. Source: own study.

To measure the performance we used Java Microbenchmark Harness (JMH) [6, 28] which is a framework for building, running, and analyzing benchmarks written in Java and other languages targeting the JVM. All the tests were performed on a 16-thread AMD Ryzen 7 3700X CPU with 32GB of RAM, setting maximum heap size for Neo4J to 4GB. Our tests consisted of choosing subsequent pairs of rooms from the pre-generated pool of and calculating the path between them. We tested each combination of heuristics and grid size for 20 minutes (plus 3 minutes for JVM warm-up [32]).

The results are presented in Table 1. The column meanings, from left to right, are as follows: the experiment label, the heuristics used by the A* algorithm, the density of places in the rooms (lower values mean a higher density), the total number of vertices and edges in the navigation graph, and the average time needed to calculate a single path.

The results have been also summarized in Figure 4. The most important observation is that commonly used A* heuristics not only do not improve the response time, but make it worse. The reason for such behaviour is twofold.

[5]Manhattan Distance, sometimes referred to as Taxicab Distance, is a metric used to determine the distance between two points in a grid-like path, i.e., it is the sum of the absolute differences between the coordinates. For the points (x_1, y_1), (x_2, y_2) the Manhattan distance is $|x_1 - x_2| + |y_1 - y_2|$.

Table 1. Average path computation times depending on the grid size and the heuristics used.

Label	A* Heuristics	Grid Size	Vertex Count	Edge Count	Avg. Time [ms]
1.1	None	1x1m	5691	15649	**3.58**
1.2	Euclidean Distance				5.44
1.3	Euclidean Distance Squared				5.59
1.4	Manhattan Distance				5.53
2.1	None	0.5x0.5m	25730	87276	**12.55**
2.2	Euclidean Distance				26.05
2.3	Euclidean Distance Squared				27.31
2.4	Manhattan Distance				23.20
3.1	None	0.25x0.25m	108607	401762	**45.08**
3.2	Euclidean Distance				84.77
3.3	Euclidean Distance Squared				85.20
3.4	Manhattan Distance				65.81

The first is the computational overhead of frequently calling the heuristic function, even when its value is trivial to compute. The second is that the distance measurements can be misleading for the algorithm when it needs to calculate transitions between floors, and we performed the experiments using a 4-story building model. This indicates a need to develop a custom heuristic function that prioritize reaching the destination floor, or applying Dijkstra algorithm, as the most efficient one.

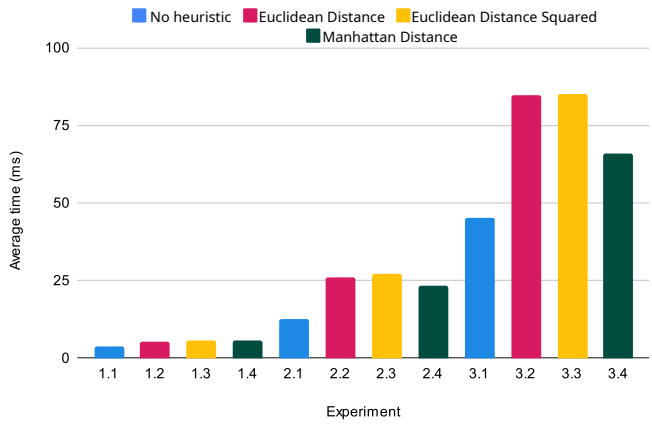

Figure 4. Average computation times during subsequent experiments.

Analyzing the remaining cases it is easy to observe that the heuristics based on the Manhattan distance performs better than the others with the difference being more pronounced with denser grid sizes. Memory consumption turned out to not be an issue, with total time spent in garbage collection (GC) being around 0.1% - testing on grid size 0.25m took one

hour 35 minutes and of that time GC was only 5.8 seconds with 1099 collections. Smaller grid sizes resulted in even less time spent in GC.

5 Conclusions and Future Work

We presented the motivation and basic concepts behind the NaviSecure system focusing on the pathfinding module and algorithms and tools used. To store the navigation graph structure we use Neo4j database because of its excellent performance, a powerful query language Cypher [4, 9], and the extensibility of the Graph Data Science library. The main goal - to provide a computationally efficient pathfinding routine that takes user preferences into account - has been achieved.

We also investigated two-phase planning approach, where the first search returns abstract paths of rooms and links, and these are then used to cut the search space by projecting onto the grid of places. As an alternative, we also considered using symbolic methods based on translation to SAT [11, 21], SMT [10], or BDD [13], however the conducted experiments have shown that the direct approach is sufficiently scalable and effective for the needs of the NaviSecure system. Tests conducted on graphs representing real building models showed that explicit algorithms running on top of Neo4j demonstrate sufficient performance and have the potential to serve as a pathfinding engine. As future work, we plan to develop a dedicated floor-aware heuristic function that is simple and efficient enough to improve the current results.

References

1. Extending Neo4j. https://neo4j.com/docs/java-reference/current/extending-neo4j/, [Accessed 19-07-2023]
2. Graph Data Science library repository. https://github.com/neo4j/graph-data-science/, [Accessed 19-07-2023]
3. GraphHopper Routing Engine repository. https://github.com/graphhopper/graphhopper, [Accessed 10-07-2023]
4. Neo4j - cypher query language. https://neo4j.com/developer/cypher/, [Accessed 19-07-2023]
5. Open street map - about page. https://www.openstreetmap.org/about, [Accessed 12-07-2023]
6. OpenJDK Code Tools: JMH. https://openjdk.org/projects/code-tools/jmh/, [Accessed 12-07-2023]
7. Ahmetovic, D., Gleason, C., Ruan, C., Kitani, K., Takagi, H., Asakawa, C.: Navcog: A navigational cognitive assistant for the blind. In: Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services. p. 90–99. MobileHCI '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2935334.2935361, https://doi.org/10.1145/2935334.2935361
8. Alqahtani, E., Alshamrani, F., Syed, H., Al-Haidari, F.: Survey on algorithms and techniques for indoor navigation systems pp. 1–9 (04 2018). https://doi.org/10.1109/NCG.2018.8593096
9. Anthapu, R.: Graph Data Processing with Cypher: A practical guide to building graph traversal queries using the Cypher syntax on Neo4j. Packt Publishing (2022), https://books.google.pl/books?id=NR2gEAAAQBAJ

10. Barrett, C., Tinelli, C.: Satisfiability modulo theories. Handbook of model checking pp. 305–343 (2018)
11. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
12. Bondy, J.A., Murty, U.: Graph Theory With Applications. Elsevier Science Ltd/North-Holland, hardcover edn. (1984)
13. Bryant, R.E.: Binary decision diagrams. Handbook of model checking pp. 191–217 (2018)
14. Burdick, J.W.: Summary of the A* algorithm. http://robotics.caltech.edu/wiki/images/e/e0/Astar.pdf, [Accessed 12-07-2023]
15. Chandel, V., Ahmed, N., Arora, S., Ghose, A.: Inloc: An end-to-end robust indoor localization and routing solution using mobile phones and ble beacons. In: 2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN). pp. 1–8 (2016). https://doi.org/10.1109/IPIN.2016.7743592
16. Chen, L.W., Cheng, J.H., Tseng, Y.C.: Optimal path planning with spatial-temporal mobility modeling for individual-based emergency guiding. vol. 45, pp. 1–1 (12 2015). https://doi.org/10.1109/TSMC.2015.2445875
17. Chen, L.W., Chung, J.J., Liu, J.X.: Gofast: A group-based emergency guiding system with dedicated path planning for mobile users using smartphones. In: 2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems. pp. 467–468 (2015). https://doi.org/10.1109/MASS.2015.35
18. Deng, K., Li, M., Zhong, W.: Optimal emergency evacuation route planning model based on fire prediction data. Mathematics **10**(2), 379–392 (2022). https://doi.org/10.3390/math10020379
19. Foead, D., Ghifari, A., Kusuma, M.B., Hanafiah, N., Gunawan, E.: A systematic literature review of A* pathfinding. Procedia Computer Science **179**, 507–514 (2021). https://doi.org/https://doi.org/10.1016/j.procs.2021.01.034, 5th International Conference on Computer Science and Computational Intelligence 2020
20. Gao, J., Ye, W., Guo, J., Li, Z.: Deep reinforcement learning for indoor mobile robot path planning. Sensors **20**(19), 5493 (2020). https://doi.org/10.3390/s20195493, https://www.mdpi.com/1424-8220/20/19/5493
21. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. Foundations of Artificial Intelligence **3**, 89–134 (2008)
22. Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4), 12–18 (2008). https://doi.org/10.1109/MPRV.2008.80
23. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968). https://doi.org/10.1109/TSSC.1968.300136
24. Hodler, A.E., Needham, M.: Graph data science using Neo4j. In: Massive Graph Analytics, pp. 433–457. Chapman and Hall/CRC (2022)
25. Jiang, J.R., Subakti, H., Liang, H.S.: Fingerprint Feature Extraction for Indoor Localization. Sensors **21**, 5434 (Aug 2021). https://doi.org/10.3390/s21165434
26. Ko, N.Y., Noh, S.W., Moon, Y.S.: Implementing indoor navigation of a mobile robot pp. 198–200 (2013). https://doi.org/10.1109/ICCAS.2013.6703892
27. Kumar, A., Singh, A.: Graph Database Modeling with Neo4j. Amazon Digital Services LLC - KDP Print US (2020), https://books.google.pl/books?id=Z9PZzQEACAAJ
28. Laaber, C., Leitner, P.: An evaluation of open-source software microbenchmark suites for continuous performance assessment. In: Proceedings of the 15th International Conference on Mining Software Repositories. pp. 119–130 (2018)
29. Leitch, S.G., Ahmed, Q.Z., Abbas, W.B., Hafeez, M., Laziridis, P.I., Sureephong, P., Alade, T.: On indoor localization using wifi, ble, uwb, and imu technologies. Sensors **23**(20) (2023). https://doi.org/10.3390/s23208598

30. Li, Y., Shin, B.S.: Internal topology based flexible shortest path planning method for indoor navigation pp. 171–176 (2015)
31. Link, J.A.B., Smith, P., Viol, N., Wehrle, K.: Footpath: Accurate map-based indoor navigation using smartphones. In: 2011 International Conference on Indoor Positioning and Indoor Navigation. pp. 1–8 (2011). https://doi.org/10.1109/IPIN.2011.6071934
32. Lion, D., Chiu, A., Sun, H., Zhuang, X., Grcevski, N., Yuan, D.: Don't get caught in the cold, warm-up your jvm. In: Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI'16) (2016)
33. Liu, K., Motta, G., Ma, T., Guo, T.: Multi-floor indoor navigation with geomagnetic field positioning and ant colony optimization algorithm pp. 314–323 (2016). https://doi.org/10.1109/SOSE.2016.18
34. Liu, Q., Yang, J., Huang, D.: Research on path planning for intelligent mobile robots based on improved a* algorithm. Symmetry **16**(10), 1311 (2024). https://doi.org/10.3390/sym16101311, https://www.mdpi.com/2073-8994/16/10/1311
35. Manlises, C., Yumang, A., Marcelo, M., Adriano, A., Reyes, J.: Indoor navigation system based on computer vision using camshift and d* algorithm for visually impaired pp. 481–484 (2016). https://doi.org/10.1109/ICCSCE.2016.7893623
36. Mikulowski, D., Salamonczyk, A., Niewiadomski, A., Pilski, M., Terlikowski, G., Switalski, P.: An approach to provide a universal structural map to aid navigation inside campus buildings using beacons and mobile devices. In: Bastiaens, T. (ed.) Proceedings of EdMedia + Innovate Learning 2023. pp. 1198–1203. Association for the Advancement of Computing in Education (AACE), Vienna, Austria (July 2023), https://www.learntechlib.org/p/222636
37. Mikułowski, D., Niewiadomski, A., Salamończyk, A., Pilski, M., Świtalski, P., Terlikowski, G.: Supporting independent navigation of disabled students in university campus using beacons and ontology map. In: Bastiaens, T. (ed.) Proceedings of EdMedia + Innovate Learning 2022. pp. 1005–1010. Association for the Advancement of Computing in Education (AACE), Online (November 2022)
38. Mikułowski, D., Salamończyk, A., Świtalski, P., Niewiadomski, A., Terlikowski, G., Pilski, M.: Supporting user positioning and navigation inside campus buildings using ble transmitters and geometric methods. In: Bastiaens, T. (ed.) Proceedings of EdMedia + Innovate Learning 2024. pp. 93–102. Association for the Advancement of Computing in Education (AACE), Brussels, Belgium (July 2024)
39. Nazarczuk, M.: Moduł wyznaczania ścieżki dla systemu NaviSecure (in Polish). Master's thesis, Uniwersytet Przyrodniczo-Humanistyczny w Siedlcach, Siedlce (September 2023)
40. Needham, M., Hodler, A.E.: Graph algorithms: practical examples in Apache Spark and Neo4j. O'Reilly Media (2019)
41. Pala, M., Osati Eraghi, N., López-Colino, F., Sánchez, A., De Castro, A., Garrido, J.: Hctnav: A path planning algorithm for low-cost autonomous robot navigation in indoor environments. ISPRS International Journal of Geo-Information **2**, 729–748 (08 2013). https://doi.org/10.3390/ijgi2030729
42. Patel, A.: A*'s use of the heuristic. http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html, [Accessed 12-07-2023]
43. Prandi, C., Barricelli, B., Mirri, S., Fogli, D.: Accessible wayfinding and navigation: a systematic mapping study. Universal Access in the Information Society **22** (09 2021). https://doi.org/10.1007/s10209-021-00843-x
44. Richter, J., Lorenz, J., Costantino, M., Traubinger, V., Tauchmann, N., Graichen, T., Heinkel, U.: Dynamic indoor navigation and orientation system for people with impairments. pp. 473–477 (09 2020). https://doi.org/10.1145/3404983.3410000

45. Singh, J., Kaur, H., Johnson, M.T., et al.: 3d indoor modeling and game theory based navigation for pre and post covid-19 situation. Frontiers in Public Health **11**, 123–132 (2023). https://doi.org/10.3389/fpubh.2023.012345
46. Switalski, P., Salamonczyk, A.: Universal structural map for indoor navigation in university campus. Studia Informatica. System and information technology **29**(2), 69–79 (2023)
47. Zaniewicz, N., Salamończyk, A.: Comparison of MongoDB, Neo4j and ArangoDB databases using the developed data generator for NoSQL databases. Studia Informatica. System and information technology **26**(1), 61–72 (Nov 2022). https://doi.org/10.34739/si.2022.26.04