Air-Aware Walking Routes
Air Quality Analysis and Path Finding on Modena

Giorgia Bertacchini
Graph Analytics course
Master in Artificial Intelligence Engineering
University of Modena and Reggio Emilia - UNIMORE
Modena, Italy
274372@studenti.unimore.it

***Abstract*—Traditional pedestrian route planning focuses mainly on distance, ignoring factors like air quality and green spaces that have an impact on well-being. This work introduces a path finding approach that integrates environmental considerations into urban navigation. Using data from PM10 sensors in Modena, an air quality raster is generated through interpolation and incorporated into a Neo4j road network, where paths can be optimized based on pollution levels, green areas, or a weighted combination of factors. The system, adaptable to other cities, is designed for easy data integration and accepts JSON-based routing requests, enabling seamless use in web applications and mobility services.**
My code is available at Graph Routing Air quality GitHub Repository.

I. Introduction

The planning of walking routes in urban areas traditionally prioritizes minimizing distance or walking time. However, in modern cities, additional factors such as air quality and green spaces play a crucial role in ensuring a healthier and more pleasant walking experience. Exposure to high levels of air pollution, particularly PM10 particles, has been linked to severe health issues, while green areas contribute to well-being by improving air quality and reducing stress levels.

This paper presents a novel path finding approach that incorporates air pollution and green area coverage in the pedestrian routes. The proposed system is based on a road network modeled in Neo4j, where nodes represent road junctions and edges (ROUTE) represent pedestrian streets enriched with attributes such as distance, air quality and green area coverage. To estimate air quality along routes, PM10 concentration values from sensors distributed across the city of Modena are processed using Inverse Distance Weighting (IDW) interpolation, generating a raster that extends pollution data beyond discrete sensor locations. The interpolated values are then used to compute an average PM10 concentration for each edge of the route network.

The system supports multiple path finding algorithms, including Dijkstra's algorithm, A* and Yen's k-shortest paths, enabling users to find optimal routes based on different criteria: minimizing exposure to pollution, maximizing the coverage of green areas or balancing these factors with distance travelled through a weighted scoring function. The current implementation is based on data from Modena, but the methodology is designed to be adaptable to pedestrian route networks in other cities.

Furthermore, the system architecture is structured for easy integration and usability. Users can enter their own datasets without having to make substantial changes to the code. Additionally, the system is designed to accept routing requests in JSON format, allowing seamless integration with web applications and other software solutions. This flexibility makes the approach suitable for various urban mobility applications, such as pedestrian navigation apps and smart city platforms.

II. Preliminaries

A. Graph Representation in Neo4j

Neo4j is a graph database that represents data through **nodes** (entities) and **edges** (relations). I recommend installing Neo4j Desktop from the following link: Download Neo4j Desktop, if you have not already done so. This will allow you to easily set up your graph database for pedestrian roads. However, be sure to also install the **Graph Data Science Library** for the subsequent operations.

As can be seen in Figure 1, in our model:

- **Nodes** represent road junctions and contain attributes such as latitude and longitude.
- **Edges** represent roads (ROUTES) and are characterized by attributes such as distance (expressed in meters) and green area coverage.

B. Air Quality Estimation with IDW Interpolation

To estimate air quality along streets, we use the **Inverse Distance Weighting (IDW)** interpolation method. IDW assigns estimated values based on nearby sensor readings, with weights inversely proportional to the distance. This method provides a continuous distribution of PM10 values across the urban area.

For the interpolation operation, the **GDAL** library was used, which offers a powerful set of tools for spatial data processing. Useful details can be found at GDAL Grid Tutorial - Inverse Distance to a Power.

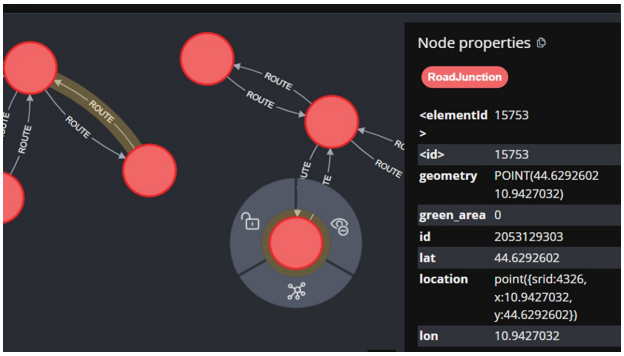

Fig. 1. Example of nodes and edges of my graph database.

C. Shortest Path Algorithms

The main algorithms used for shortest path computation in a graph are:

- **Dijkstra's Algorithm**: finds the shortest path from a source node to all other nodes in a weighted graph, guaranteeing optimality. To find the shortest path between a source node and a target node, the Dijkstra Source-Target algorithm can be applied.
- **A* Algorithm**: improves Dijkstra's algorithm by introducing a heuristic function to speed up path finding.
- **Yen's k-shortest paths**: computes the top k shortest paths between two nodes, useful for providing alternative routes.

Official documentation about the algorithms used: Neo4j - Path finding algorithms.

D. Visualization with QGIS

To visualize the results on a map, I used **QGIS**, an open-source geographic information system that allows easy visualization, editing and analysis of spatial data. The output files generated by the code are often structured in a way that facilitates integration as layers in QGIS, allowing for intuitive data exploration and visual representation of pathways and air quality.

It is possible to download the installation from QGIS Installers.

III. Methods

A. Air Quality Data Input

The air quality input data consists of two main components: the locations of the air quality sensors and the corresponding PM10 measurements.

The sensor locations were provided by my university's research group, and include a list of latitude and longitude coordinates for sensors previously installed in Modena. In addition, real-time PM10 values were obtained from two ARPAE sensors currently active in Modena, with daily measurements for the year 2024. These sensors are located in different areas of the city, providing a snapshot of the air quality for a limited number of locations.

Due to the lack of real world data provided by the ARPAE sensors, I generated synthetic air quality values to simulate a wider coverage. To determine the location of my sensors, I relied on historical sensor placement data but extended the coverage to key areas of the pedestrian road network, not represented by the original sensors.

For the synthetic PM10 values, I assigned plausible initial values based on geographic logic — low pollution in green spaces and high pollution near highways or urban roads — and used the average PM10 values derived from the data of the two real ARPAE sensors for 2024 as a baseline. To perform further tests, I generated two sets of synthetic PM10 data with different levels of variability: one with a standard deviation of about 10 (see Figure 2), and the other with a standard deviation of about 20 (see Figure 3), keeping the average PM10 levels consistent with the original dataset.

This data manipulation and the generation of synthetic PM10 values can be seen in the file `pre_analysis.py` in my code. My input data consists of two CSV files: the first contains static information about the sensors (ID_STATION, name, latitude and longitude), and the second includes the PM10 measurements with corresponding dates and sensor IDs.

B. Spatial Visualization of PM10

To create a visual representation on the map, I added layers to QGIS, with the OpenStreetMap layer preloaded.

In particular, I imported delimited text layers: one to display sensor locations with only the coordinates and names (represented as red dots in the images) and two to show PM10 values from the CSV files previously created.

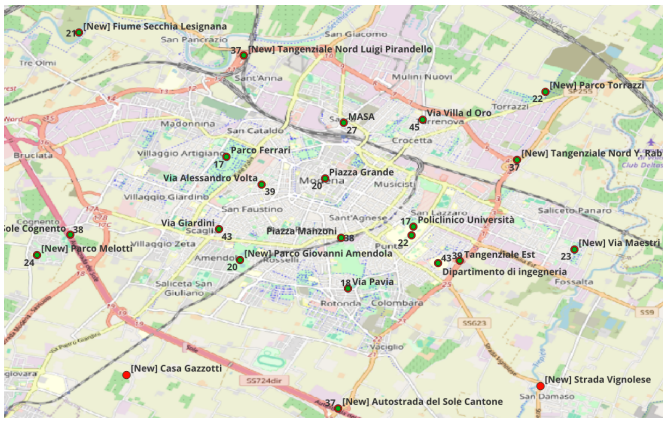

Fig. 2. Synthetic PM10 data with standard deviation of 10.

C. Raster Generation and IDW Interpolation

To obtain a complete representation of PM10 levels in the entire Modena area, it was necessary to generate a raster covering the city. The appropriate raster size was determined by accessing the Neo4j database to retrieve the minimum and maximum latitude and longitude values. To ensure complete coverage, a buffer of 5% was added to the raster boundaries, extending slightly beyond the city limits.

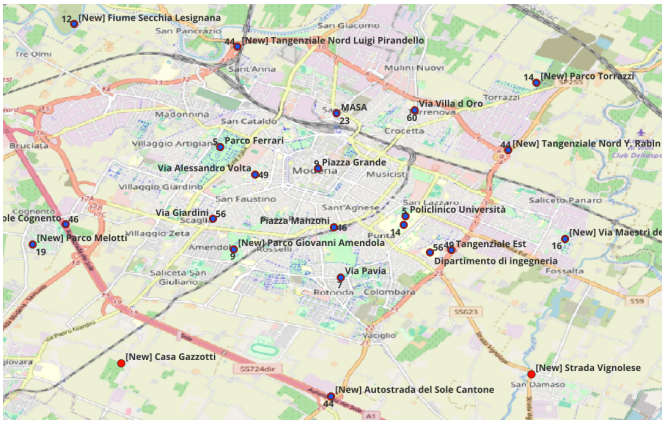

Fig. 3. Synthetic PM10 data with standard deviation of 20.

The GDAL library was used for interpolation, employing the Inverse Distance Weighting method. IDW is a spatial interpolation technique that estimates unknown values based on the proximity of known data points. The estimated Z-value at a given location is calculated as:

$$Z = \frac{\sum_{i=1}^{n} \frac{Z_i}{r_i^p}}{\sum_{i=1}^{n} \frac{1}{r_i^p}} \tag{1}$$

where Z_i represents the known values at neighboring locations. The weighting factor is computed as:

$$w = \frac{1}{r^p} \tag{2}$$

where r_i is the distance between the interpolation point and the known data point, and p is a power parameter controlling the influence of surrounding points, see Figure 4.

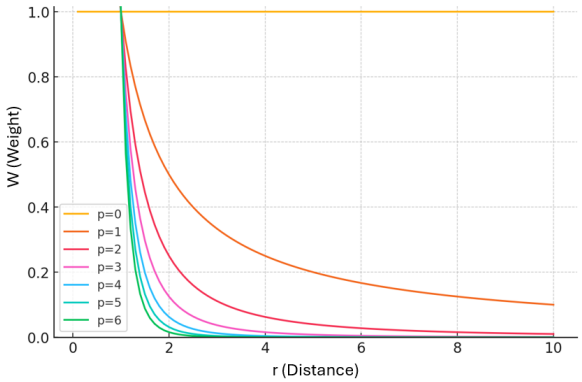

Fig. 4. Curves of w as a function of r for different values of the exponent p (from 0 to 6).

A raster file was generated using the following GDAL command:

```
gdal.Grid(raster_path, vrt_file_path,
    algorithm=f"invdist:power={power}
    :radius1={radius1}:radius2={radius2}",
    outputBounds=[x_min, y_min,
                  x_max, y_max])
```

The values of radius1 and radius2 act as parameters that define the search window (ellipse, see Figure 5.), specifying the area where the PM10 values are taken into account to calculate the raster at each location.

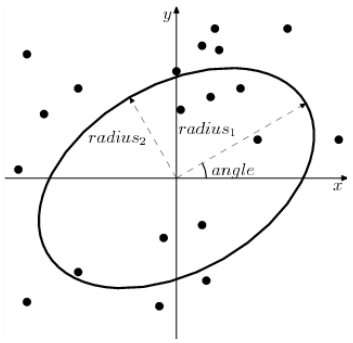

Fig. 5. Search Ellipse of IDW interpolation computation.

The command creates a GeoTIFF (.tif) file which can be easily imported into QGIS as a new layer for visualization. The generated raster, one for each level of variability, provides a continuous surface representation of PM10 concentration, enabling a better understanding of air pollution distribution in Modena, see Figure 6 and Figure 7.

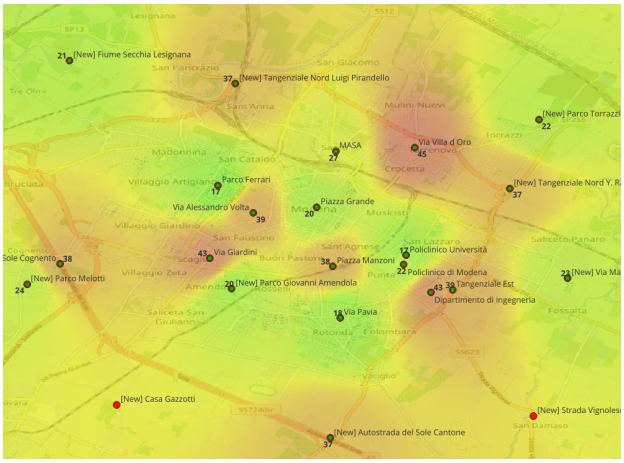

Fig. 6. Raster from PM10 data with standard deviation of 10.

D. Integration of Pollution Data into the Graph Database

At this stage, we aim to incorporate pollution data into our graph database. To begin, we need to configure the JSON settings file by specifying the correct Neo4j URL, username and password.

Since the road network consists of straight-line routes on the map, we sample PM10 values from the raster at the segments (lines) with endpoints at the locations of the graph nodes. This is done by extracting the coordinates of the node pairs that define each edge, using the `map_coordinates` function of Scipy library to interpolate values within a multidimensional array. It is possible find the manual at

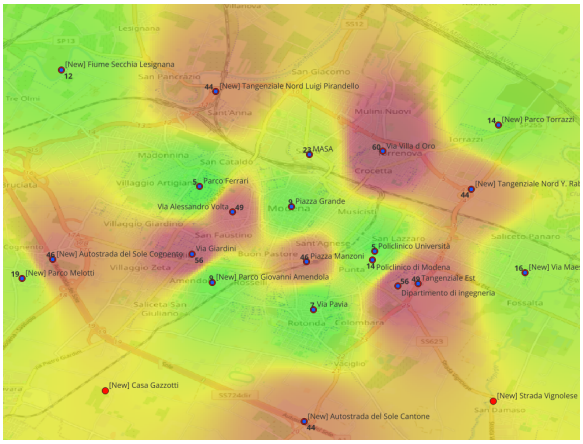

Fig. 7. Synthetic PM10 data with standard deviation of 20.

scipy.ndimage.map_coordinates. If a broader sampling area is required, the JSON configuration file allows setting a buffer value, expressed in raster pixels. In my specific case, each pixel represents approximately 0.000543058828125012 meters by -0.00038998480468752095 meters. The sampled PM10 values are then averaged to obtain a single representative value for each edge, which is stored as a property in micrograms per cubic meter (µg/m^3).

To efficiently load this property into the database and minimize execution time, we use a single Cypher query that performs a bulk insert of all the computed PM10 values, associating them with the corresponding node coordinate pairs. Given that roads are bidirectional, the query also assigns the calculated PM10 value to the reverse direction, further optimizing the process.

For visualization purposes, we exported road junction nodes to be loaded into QGIS as a delimited text layer. Additionally, we exported the route edges — including their associated properties — as a CSV file. These were then imported into QGIS using the *geometry by expression* method to render road segments as lines. The expression used in the tool is:

```
make_line(
  make_point("source_lon", "source_lat"),
  make_point("target_lon", "target_lat")
)
```

Fig. 8. Modena routes with PM10 level value.

This approach enables an intuitive spatial representation of PM10 pollution levels along the road network, see Figure 8.

E. Optimization Factors for Route Calculation

Since the goal is to allow route to be found according to different optimization criteria, we begin by analyzing the possible graph parameters that can be used. The simplest approach is to find the shortest path based on route distance. However, we can also allow the search for paths that minimize pollution exposure. For this purpose, the property `pm10` needs to be modified slightly.

Since PM10 values are measured in micrograms per cubic meter (µg/m^3), we must take into account the total pollution exposure along a route by multiplying the PM10 concentration by the segment length. This new value will serve as the cost metric for optimizing the route selection based on air pollution.

$$PM10_{\text{route}} = PM10 \cdot d \tag{3}$$

Another important factor to consider is the **green area** along the routes. Given that path finding algorithms are designed to minimize a chosen cost function, we need to *invert* the `green_area` parameter, as our goal is to prioritize routes with more green rather than minimise it.

Analysing this parameter, we observed that most routes have no green areas (i.e., `green_area` = 0), while only a few segments have values of 50 or 100. Since these values are relatively high, we normalize them by dividing them by 100 to prevent an unbalanced influence on the presence of green.

Additionally, since some values are zero, we must prevent division by zero by introducing a small constant ϵ, which we set to 1. To ensure that green areas are also correlated with distance - since longer green roads should be prioritized - we multiply the adjusted `green_area` value by the segment length.

$$GA_{\text{route}} = \frac{d}{\frac{greenArea}{100} + \epsilon} \tag{4}$$

This results in the following behavior:

- If no green area is present, the cost remains equal to the distance (which we always want to be relatively high in this case).
- If a green area is present, its contribution reduces the overall cost, making greener and longer routes more favorable in the optimization process.

To better understand the behavior of these three factors, we can analyze the distributions of their values represented in Figure 9.

To allow path finding that considers both green area and pollution factors simultaneously, an additional weight has been introduced. This weight is calculated as the weighted average of the normalized values of the green area along the route and the PM10 levels along the route. The calculation is performed for each edge of the route and added as a new property.

The normalization was necessary because these parameters have different distribution ranges. Without normalization, one

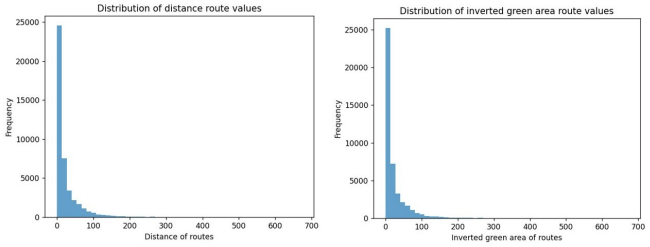

Fig. 9. Distribution of route distances and inverted green area.

TABLE I
MAXIMUM AND MINIMUM VALUES FOR ROUTE PM10 AND ROUTE GREEN AREA

	Route PM10 (µg)	**Inverted Route Green Area**
Max Value	20712.293	672.219
Min Value	1.900	0.084

parameter could have a disproportionately higher influence over the other due to its range. By normalizing the values, we ensure that both factors contribute equally to the calculation.

$$PM10'_{\text{route ij}} = \frac{PM10_{\text{route ij}} - PM10_{\text{route min}}}{PM10_{\text{route max}} - PM10_{\text{route min}}} \quad (5)$$

$$GA'_{\text{route ij}} = \frac{GA_{\text{route ij}} - GA_{\text{route min}}}{GA_{\text{route max}} - GA_{\text{route min}}} \quad (6)$$

Furthermore, to allow the user to customise the routing request, it is possible to adjust the weights assigned to the green area and PM10. These user-defined weights are applied as multipliers to the respective parameters, enabling users to prioritize one factor over the other according to their preferences.

$$\text{weight}_{ij} = w_{\text{PM10}} \cdot PM10'_{\text{route ij}} + w_{\text{ga}} \cdot GA'_{\text{route ij}} \quad (7)$$

This newly computed combined value will act as the new weight that the path finding algorithms can use to determine a route that achieves a more balanced trade-off between the two factors.

F. Path finding Algorithms

In this study, we used three path finding algorithms provided by Neo4j: Dijkstra, A* (A-Star) and Yen. These algorithms were used to identify optimal paths between two nodes, taking into account previously described and adapted factors, such as distance, pollution and green areas.

a) Dijkstra Algorithm: The Dijkstra algorithm is a classic technique for determining the shortest path between two nodes in a graph with non-negative weights. It works by iteratively exploring adjacent, unvisited nodes and selecting the node with the minimum cumulative distance from the starting node until it reaches the destination node. In Neo4j, the Dijkstra algorithm can be implemented using the `gds.shortestPath.dijkstra` function.

b) A Algorithm:* The A* (A-Star) algorithm is an informed variant of Dijkstra's algorithm, using a heuristic function to guide the search towards the destination node more efficiently. In Neo4j, the A* algorithm employs the Haversine formula as its heuristic function, which calculates the distance between two points on the Earth's surface based on their geographic coordinates. This approach is particularly useful when working with spatial data and aiming to find optimal paths considering the geolocation of nodes.

c) Yen's Algorithm: Yen's algorithm, also known as the K-shortest paths algorithm, not only finds the shortest path between two nodes but also identifies the next k-1 shortest paths, while avoiding cycles. This is especially useful for analyzing alternative routes between two points. In Neo4j, Yen's algorithm can be implemented to explore multiple paths between nodes, providing a more comprehensive overview of possible routes.

To request the optimal path or the best multiple paths along with their respective details, the system expects these parameters to be read from a JSON configuration file. Specifically, the file contains the ID of the starting road junction node, the ID of the destination road junction, the preferred path finding algorithm (and, in the case of Yen's algorithm, the number of top paths to be returned), as well as the weight to be used for the path search.

G. Exporting and Visualizing Optimal Routes in GeoJSON Format

Once the optimal path(s) have been calculated using Neo4j algorithms, they are exported into a GeoJSON file for visualization on a map. The file generates a geometry feature of type `LineString`, which draws a line for each pair of road junction nodes along the path.

Additionally, various properties are included, such as the total distance, total green area, total PM10 along the route and the total inverse green area along the route. These properties can be displayed as labels in QGIS above the visualized path.

IV. RESULTS AND DISCUSSION

To evaluate the effectiveness of the implemented path finding algorithms, several routing tests were conducted using different parameters and weights. The tests generated multiple paths based on various criteria, such as shortest distance, lowest pollution levels, highest green area coverage and a balanced combination of both factors.

The results showed that the selected algorithms successfully computed optimal paths according to the specified objectives. When prioritizing pollution reduction, the generated routes tended to avoid high PM10 concentration areas, whereas paths optimizing green area coverage favored roads surrounded by vegetation. Additionally, when using the newly introduced weighted combination of both factors, the resulting paths provided a more balanced trade-off between minimizing pollution and maximizing green area coverage.

These computed paths were then exported in GeoJSON format for visualization in QGIS, where they could be analyzed

with additional contextual data. The extracted route properties, such as total distance, total green area, and PM10 levels, confirmed that the algorithm behaved as expected based on the assigned weights.

A. Comprehensive Routing Example

A complete routing example was conducted, to show the results of the project. A specific starting node and a destination node were selected, and the optimal path was calculated for each factor individually. The tests included:

- The **shortest path** based on physical distance.
- The **path minimizing PM10 pollution levels**.
- The **path maximizing green area coverage**.
- The **path balancing both pollution and green area** using the weighted combination approach.

The results of these computations are visually represented in Figure 10, where each generated path is displayed on a map. This comparison highlights how different criteria influence the chosen route and demonstrates the flexibility of the approach to adapting to user preferences.

Fig. 10. Comparison of optimal paths based on different criteria.

TABLE II
COMPARISON OF COMPUTED PATHS BASED ON DIFFERENT ROUTING CRITERIA.

Routing Criteria	Distance (m)	Tot PM10(µg)	Tot Green Area
Shortest Distance	9235	310325	0
Lowest PM10	9810	294817	0
Highest Green Area	9540	316246	6400
Balanced (PM10 + Green)	9784	297955	3800

B. Comparison of Path finding Results with Different PM10 Variability

To analyze the impact of pollution variability on the computed paths, a comparison was conducted using two different scenarios: one where PM10 values had a standard deviation of 10 and another where the standard deviation was 20. This variation in PM10 distribution affects how the path finding algorithms evaluate and prioritize routes based on pollution levels.

The results indicate that when the standard deviation is lower ($\sigma = 10$), the calculateded paths tend to be more stable, with fewer drastic deviations from the shortest distance. However, in the scenario with a higher standard deviation ($\sigma = 20$), the algorithms exhibit a greater sensitivity to pollution hotspots, leading to significantly different route choices. In some cases, the paths diverge substantially from those computed in the lower variability scenario, as the algorithms attempt to avoid highly polluted areas more aggressively. Figures 11 and 12 illustrate the two cases.

Fig. 11. Results of the least polluted route. Light-blue: case of standard deviation of 10. Red: case of standard deviation of 20.

Fig. 12. Results of the least polluted route. Light-blue: case of standard deviation of 10. Red: case of standard deviation of 20.

This comparison highlights the influence of data distribution on path finding outcomes and underscores the importance of considering pollution variability when designing routing strategies.

C. Comparison of Dijkstra and A Path finding Results*

To further evaluate the behavior of the path finding algorithms, additional tests were conducted to compare the results obtained using Dijkstra's algorithm and the A* algorithm. The goal was to determine whether the two approaches yield different optimal paths under the same conditions.

The tests were performed in multiple scenarios with varying weights and path constraints. The results showed that in all tested cases, the paths computed by both algorithms were identical.

Figures 13 and 14 illustrate two example cases where the computed paths are visually identical, confirming that both algorithms arrive at the same solution. This consistency demonstrates that A* does not alter the final path selection, but improves computational efficiency by reducing the number of explored nodes.

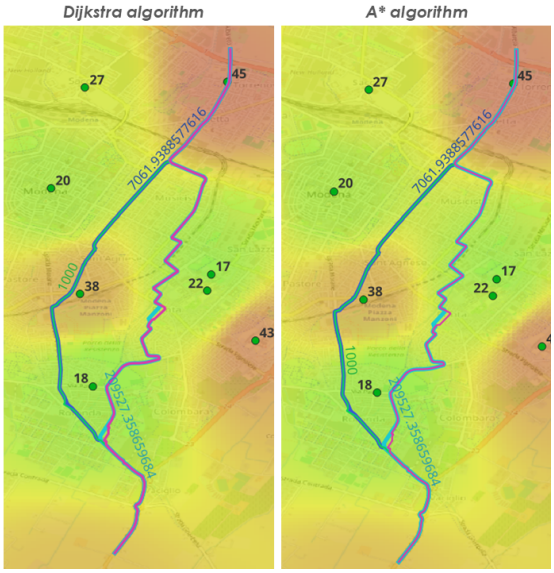

Fig. 13. An example to compare Dijkstra's algorithm and A* algorithm.

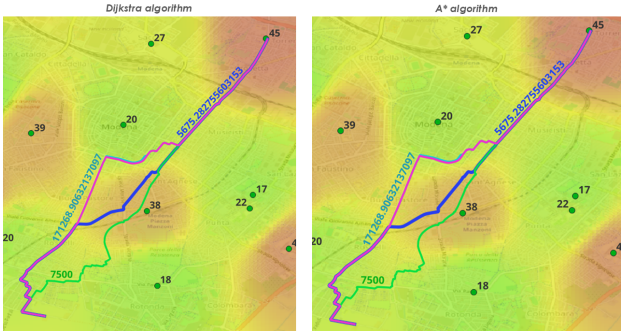

Fig. 14. Another example to compare Dijkstra's algorithm and A* algorithm.

V. Conclusion

The findings highlight the potential of this approach to improve urban mobility by incorporating health and environmental factors into pedestrian route planning. By offering a solution that encourages the selection of healthier routes, this work aims to promote better well-being for urban residents.

Although the research is focused on Modena, the methodology is scalable and can be applied to other cities with similar environmental data. Future work could extend this approach by integrating additional environmental factors, such as noise pollution and altitude differences, as well as optimizing routes based on diverse user preferences and mobility needs. Furthermore, the system provides a fast and simple solution for integration into future applications, enabling seamless adoption into web-based services.

Ultimately, this work lays the foundation for more sustainable, health-conscious urban navigation systems, balancing efficient walking with environmental well-being.