Researching open source WebGIS technology in building software to support optimal search and rescue equipment

Yen Phan Quoc*

Le Quy Don Technical University, 236 Hoang Quoc Viet Street, Co Nhue 1 Ward, Bac Tu Liem District, Hanoi, Vietnam

Received 2 March 2023; revised 11 April 2023; accepted 24 January 2024

Abstract:

Securing combat equipment for urgent special missions, such as rescue, fire protection, chemical prevention, and military operations, is crucial in technical logistics. Rapid preparation and mobilisation of forces in response to the reality of the tasks are essential. A software system that stores a complete database of equipment, human resources, and navigation maps is required. This article explores the application of Open Source WebGIS technologies, such as Geoserver, PostGIS, OpenLayer, GraphHopper, and Python, in experimentally building management software for searching equipment, units, and navigation. It supports decision-making by allowing users to select the optimal response plan based on digital maps. The software integrates spatial analysis and routing techniques to locate units with suitable equipment, either nearest or within a specified radius. Users can thereby determine the optimal route from the selected unit to the mobilisation site. Testing with fire prevention data in Hanoi city showed that the search results for the optimal route are precisely equivalent to those provided by current Google Maps. This research has initially mastered data and technology to proactively update and manage data in areas lacking industry-specific information that existing software does not cover.

Keywords: **decision support, equipment, search and rescue, spatial analysis, WebGIS.**

Classification numbers: **1.2, 1.3, 4.1**

1. Introduction

The Department of Defense of Canada defines search and rescue (SAR) as the search for and provision of assistance to persons or vehicles in or at risk of impending distress [1]. The logistics component of disaster preparedness is complex, encompassing the management of entire supply networks. Logistics assurance is defined as the process of planning, implementing, and controlling the storage and movement of goods, materials, personnel, and facilities to most effectively and cost-efficiently support search and rescue operations [2]. Various vehicles, such as helicopters, cars transporting relief goods, ambulances evacuating casualties, and specialised vehicles for different fields (chemical equipment, fire trucks, excavators, bulldozers), are categorised as rescue equipment. These facilities need management through software that links equipment data to the spatial data of the unit, enabling commanders to swiftly draft rescue plans, choose optimal manoeuvring routes, designate rescue vehicle parking areas, and identify suitable medical facilities.

A geographic information system (GIS) is a computer-based system that supports the collection, storage, retrieval, mapping, analysis, and distribution of geospatial data for decision support [3, 4]. Geospatial data represent natural earth objects like roads, fire stations, and parks. GIS applications visualise this data, enabling easy analysis of relationships and patterns [4]. Decisions on site selection, network routing, relief item location, and service coverage are based on this data [5]. The determination of optimal locations in GIS involves spatial analysis to compare attribute data across locations and identify the most suitable spatial position [4]. Recent advances have seen GIS-based systems widely applied across various fields, utilising spatial analysis techniques and road network analyses to optimise rescue mission responses. Notably, routing systems aid rescue crews in navigating efficiently from base stations to disaster sites, crucial in complex urban environments where factors like road length and speed impact route optimisation [6]. Currently, various algorithms, such as Dijkstra, A*, K-mean, Bellman-Ford, and Floyd-Warshall, are embedded in routing support tools like GraphHopper, Pgrouting, OSRM, Valhalla, and Openrouteservice to determine the optimal paths [7, 8].

Additionally, road network analysis to identify the optimal route is critical within a spatial decision support system, particularly for emergency response in rescue missions. A

*Email: phanquocyen@lqdtu.edu.vn

routing system that facilitates navigation between origin and destination points enables rescue teams to travel from base stations to disaster sites with maximum efficiency. In densely populated urban areas, complex road networks present specific challenges, where factors like road length and traffic speed significantly influence route optimisation [6]. Presently, several algorithms, including Dijkstra, A*, K-mean, Bellman-Ford, and Floyd-Warshall, have been integrated into routing support tools such as GraphHopper, Pgrouting, OSRM, Valhalla, and Openrouteservice, to compute the most efficient paths [7, 8].

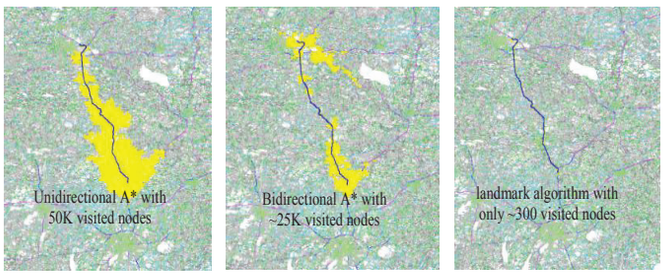

Fig. 1. GraphHopper's path finding algorithm improvement results [9].

GraphHopper stands out as an open-source, fast, and efficient routing library implementing multiple routing algorithms (unidirectional Dijkstra, one-to-many Dijkstra, and uni-and bidirectional A*, CH - Contraction Hierarchies) [9]. It offers options for routing, like the fastest or shortest route for pedestrians, motorbikes, and cars [10]. Particularly, GraphHopper boasts a faster and more accurate processing speed than other technologies. Its landmark algorithm, an enhancement using A* with a new heuristic based on landmarks and the triangle inequality for goal direction, exemplifies this [9]. Additionally, GraphHopper's ease of installation, configuration, and extension makes it highly convenient for developers [9, 11] (Fig. 1).

To build management software and ensure the equipment meets search and rescue work demands, various supporting technologies globally are considered. Since SDSS architecture is a distributed system [12], this study focuses on technologies for distributing and storing spatial data over the Internet. The open-source geospatial software family (FOSS/FOSS4G) has become foundational in the geospatial software ecosystem, providing core geospatial functionality and services, on par with commercial software [13]. FOSS4Gs are particularly vital for developing countries, allowing them to develop their technology rather than importing commercial software. This approach offers the closest path to digitalisation and is the right technology for creating a software system for unit management and equipment for SAR work in contexts of scarce development resources and system maintenance [14, 15].

Table 1. Summary of popular open source geospatial technologies [16-18].

Type	Software/libraries/programming language
Frontend libraries	**OpenLayers**, Leaflet, Cesium, Geomajas, Mapbender, GeoExt, GeoMoose, GeoNode, **Vue**, Angular, React...
WebGIS Server	**GeoServer**, MapServer, MapCache, Deegree, ncWMS, EOxServer, GeoNetwork, pycsw, PyWPS, MapProxy, QGIS Server, istSOS, 52 North SOS, 52 North WPS, Zoo Project, t-rex, Actinia...
Backend libraries	Libraries: GRASS GIS, **GeoDjango**, GDAL/OGR, GeoTools, GEOS, libLAS, JTS, PROJ4, CyberGIS Toolkit; Tools: GMT, Orfeo ToolBox, Mapnik, MapSlicer, R; Programming Language: PHP, Java, **Python**, NodeJS, **GraphHopper**...
Database management system	**PostGIS**, SpatiaLite, Rasdaman, pgRouting, MySQL Spatial, MongoDB, CouchDB, Rasdaman

Table 1 highlights various technology options within the open-source ecosystem for building a WebGIS system. These technologies are noted for their modernity, convenience, and flexibility, which facilitate development and adaptation [19]. This study aims to employ these technologies to develop a WebGIS software system for managing and searching unit data and spatial equipment data for SAR operations. The software not only provides functions for decentralized administration of users and management of equipment and common units but also incorporates advanced spatial analysis functions. These include searching for equipment and unit types by radius based on their location on the map and finding the shortest path from any unit to the incident location. The system is developed using the MVC model of the GeoDjango Framework, which is easy to inherit, adjust, and extend to other applications using a single database platform [20].

2. Methods

2.1. Back-end technology solutions

Programming languages and libraries: The backend utilises the Python programming language, a high-level programming language. To process geospatial data effectively, we integrated additional software packages into the Django Framework, including: 1. Django psycopg2 - is a PostgreSQL and PostGIS adapter for Python; 2. Geo Django - is a module for Django that turns it into a Web Framework for geospatial data. GeoDjango has the function of creating WebGIS applications. Its features include Extensions to Django's ORMs for querying and manipulating spatial data; High-level Python interfaces for GIS geometry operations and data manipulation in various formats; 3. Django REST-Framework - is a powerful and flexible toolkit for building Web APIs based on REST architecture. Django REST Framework provides robust model serialization, displaying data using standard function-based views or with views for more complex functions; 4. Django REST-framework-GIS - provides Django REST Framework geo plugins such as

GeometryField fields and GeoJSON serializers; 5. Django GeoJSON - is a toolkit for manipulating GeoJSON with Django for GeoDjango objects, query sets, and lists, etc.; base view to serve GeoJSON map layers from models; GeoJSON models and form fields to avoid spatial databases at the server; 6. Django Redis Cache - is a server cache software for the Django Web Framework, featuring a key-value data structure [20-23].

Django differentiates between an "application" and a "project". An application is a system designed for a specific function, while a project hosts multiple applications and their settings to form an entire website. Projects can host multiple applications, and applications can be migrated to multiple projects without issues. Table 2 presents the basic commands for creating software system projects and applications within Django.

Table 2. Basic commands to build a Django application [20].

No	Commands	Command Description
1	django-admin startproject CSDL_SDI	Create a project
2	python manage.py startapp GOIGIAOTHONG	Create a Goigiathong application in a project
3	python manage.py runserver	Start the test server
4	python manage.py runserver 0.0.0.0:8000	Start the test server to access the Internet

Map Server: The system's GIS server utilises GeoServer, and the web server employs Tomcat. GeoServer is a mature open-source map service publishing platform capable of easily publishing WCS, WFS, WMS, and other map services. Data imported into the PostgreSQL database through PostGIS is published, and the map service can be invoked in the frontend framework Vue using OpenLayers with the space name and namespace URL, allowing visual display in the frontend [24].

2.2. Front-end development environment

The client-side technology comprises two primary libraries: OpenLayer and Vue.js. Vue is a framework designed for building user interfaces, enabling and encouraging step-by-step application development. To integrate Vue.js, the user must embed the library using the following statement [25]:

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"> </script>

OpenLayers is an open-source client-side library for creating web-based spatial data object interactions, displaying maps in WMS/WFS format standards in most current Web browsers without requiring server-side software or installation. To use OpenLayers, execute the following import statement: *import {Map, View} from 'ol'* [26].

2.3. The solution to finding the unit location

The search for suitable units or equipment is carried out in six steps (Fig. 2). Steps 1 and 2 are built on the frontend using OpenLayers and Vue libraries; step 3 is constructed on the backend using Python to query data from the PostgreSQL database management system, with results returned in the form of GeoJSON files; step 4 involves checking the returned results-if a suitable unit is found, proceed to steps 5 and 6. If not, the process concludes; step 5 filters the space in the returned dataset by calculating the distance between the points in the returned data to the incident point. If determining the nearest point, the point with the smallest distance is selected. If within a certain radius, points whose calculated distance is less than or equal to that radius are chosen; step 6 displays the results on the map.

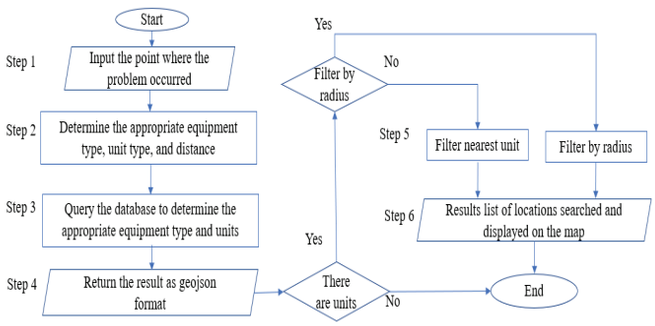

Fig. 2. Algorithm diagram for searching unit location with suitable equipment.

2.4. The solution to finding the optimal route

We consider the road system (edge-E) and the intersections of the roads (vertex-V) as a directed, non-negatively weighted graph G(E, V). The problem is to find the path from vertex x to vertex y that is the shortest or has the smallest weight. E.W. Dijktra's algorithm is as follows [27]:

dist[s] ←*0* (distance to source vertex is zero)

for all $v \in V-\{s\}$

do *dist[v]* ←∞ (set all other distances to infinity)

$S \leftarrow \emptyset$ (S, the set of visited vertices is initially empty)

$Q \leftarrow V$ (Q, the queue initially contains all vertices)

while $Q \neq \emptyset$ (while the queue is not empty)

do $u \leftarrow$ min_distance(Q,*dist*) (select the element of Q with the min. distance)

$S \leftarrow S \cup \{u\}$ (add u to the list of visited vertices)

for all $v \in$ *neighbors[u]*

do if *dist[v] > dist[u] + w(u, v)* (if new shortest path found)

then *d[v]* ←*d[u] + w(u, v)* (set new value of shortest path)

return *dist*

Dijkstra's algorithm has a running time of $O(|V|^2+|E|)$. For sparse graphs, which have few edges relative to the number of nodes, the algorithm can be implemented more efficiently using an adjacency list combined with a binary heap or priority queue, achieving a running time of $O((|E|+|V|)\ log\ |V|)$. This algorithm has been enhanced by scientists to increase processing speed through the use of appropriate heuristic evaluations, leading to optimal performance. This enhanced version is known as the A* algorithm and has been integrated into the GraphHopper library [28, 29].

To determine the optimal route in GIS, the process begins with the user entering the starting and destination points using OpenLayers and Vue libraries on the frontend. In this step, users click on the map background to select these points. The second step involves determining the type of transport-car, motorbike, or walking-based on the purpose of the journey. The third step, determining the optimal route, is conducted using the GraphHopper route search server. This step involves adjusting weights, determining priority criteria values, and sorting nodes by importance before establishing the optimal path. The formula for calculating the weight of line segments (edges) is as follows [29]:

edge_weight=edge_distance/(speed*priority)+edge_distance*distance_influence

where: edge_weight is the weight of the line segment; edge_distance is the length of the line segment; speed is the travel speed on that segment; priority is the priority of the segment, default is 1; distance_influence is the effect of distance, default is 0 [29].

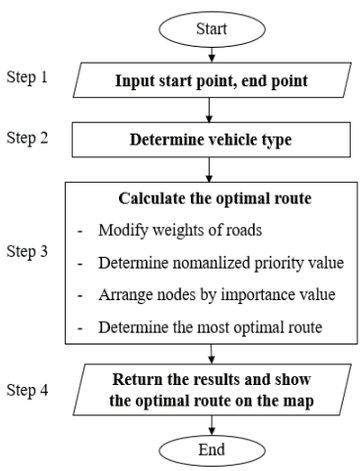

Fig. 3. Steps to find optimal path and A* algorithm.

The results are then processed using the improved algorithms on the GraphHopper server, typically the A* algorithm (Fig. 3). The final step involves returning the result to the client as GeoJSON and displaying the route on a map using OpenLayers and Vue libraries.

For GraphHopper, data processing and preparation play a particularly important role in the speed of returning results of the program. Initially, GraphHopper reads the input data, builds a graph structure for the data, and stores important feature information in a directory (graph-cache). This includes a set of information about edges, nodes, spatial relationships, and related configurations. Each time there is a request, the system combines the use of original data with data in the graph-cache to process and return results, thus optimising time without having to perform repetitive tasks.

Step 1. Set the distance to infinity for all points except for the starting point, which is set at 0.

Step 2. Mark all points, including the start point, as non-visited nodes.

Step 3. Select the non-visited node with the smallest current distance as the current node "C"

Step 4. For each neighbour "N" of "C": add the current distance of "C" with the weight of the edge connecting "C" to "N" and the weight to the destination point (heuristic). If it is smaller than the current distance of "N," set it as the new current distance of "N"

Step 5. Mark the current node "C" as visited.

Step 6. Repeat steps 3 to 5 until one of the neighbours "N" is the destination point.

3. Results and discussion

3.1. Data and experimental area

Basemap data: The base map is sourced from OpenStreetMap, open data released under the Open Data Commons Open Database License (ODbL) by the OpenStreetMap Foundation (OSMF).

Traffic class thematic data: Use GIS processing software to extract the Hanoi area to obtain the file HaNoi.osm.pbf. To prepare data, create a configuration file config2.yml with data input as Hanoi's pbf file, options for car (weighting: fastest), bike (weighting: fastest), and foot (weighting: shortest). Starting the GraphHopper server for the first time, the system builds a graph structure for the data and saves important feature information in the graph-cache (Fig. 4B). The structure of the data tables is stored to suit the optimal improvement of the applied algorithm (Figs. 4A, 4C).

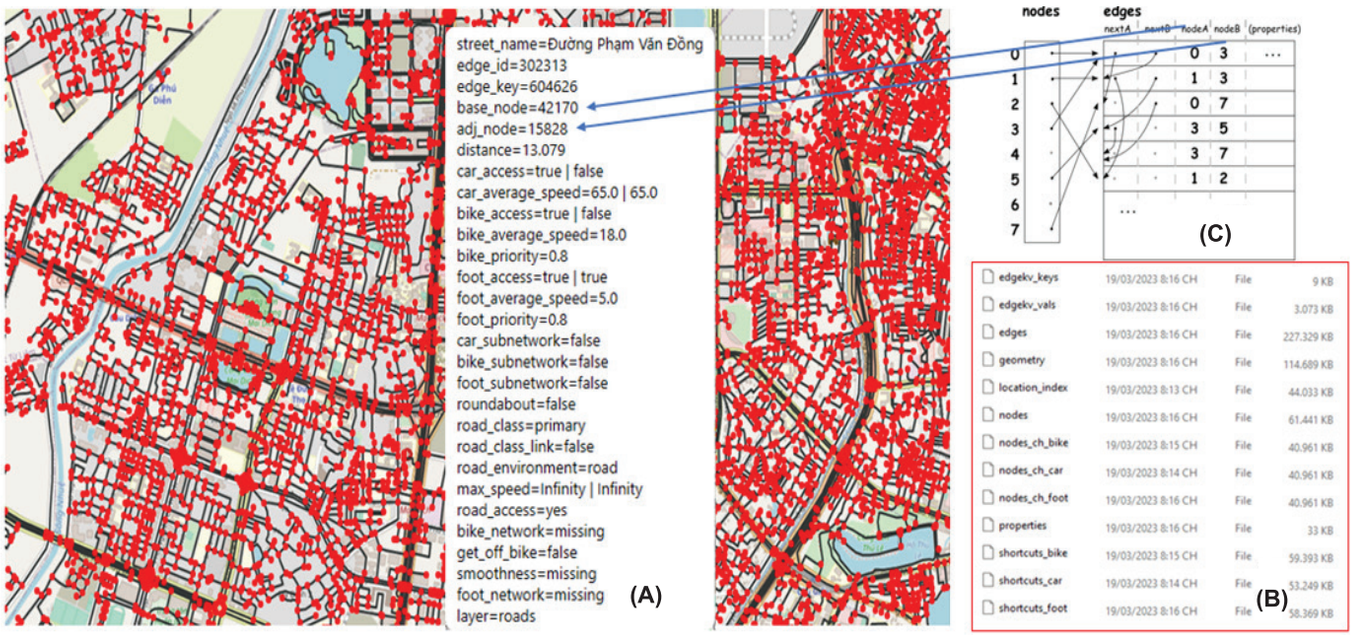

Fig. 4. Traffic thematic data prepared in Graphhopper.

Data of units and equipment: The structure of information fields and database tables are fully and wholly surveyed and designed at the centers, fire prevention, and fighting departments of the districts of Hanoi. Inside, some equipment related to the chemical defence of the Army.

3.2. Results and discussion

Architectural and functional model of the system: The system is designed with three layers: data tier, logic tier, and application tier (Fig. 5). The data layer includes spatial data (base map, unit map layer, traffic map layer) and attribute data (user data, equipment data). The logical layer comprises the Web application and the map server (Geoserver). The web application server hosts the web page, responsible for receiving HTTP requests from the client, parsing the request, and processing and responding to the HTTP client. The map server has map services according to OGC standards (WMS, WFS, WCS, etc.), provides map distribution services according to Raster and Vector standards, and performs querying, analysis, and filtering of spatial data, in addition to allowing services to add, edit, and delete spatial data. The application layer is a web browser, with accounts belonging to the system administration groups, equipment data management, and user accounts to look up, statistics equipment information, search for optimal routes, etc.

Functional diagram: The system has many functions, generally divided into four primary groups: account administration function group and user account group. This group has the function of creating/deleting/editing/authorizing accounts for a group or a single user, etc.; unit information management group, users can add/edit/delete/search/display units on the map/view, search for equipment from the map, etc.; a functional group of equipment management, including add/edit/delete/quick search/exact search/multi-criteria search/search on the map/export the list of equipment to the report form, etc.; and a group of spatial analysis functions, including network analysis to find the optimal path and buffer zone analysis to find the type of unit or type of equipment within a certain radius. Some typical functional models are shown in Fig. 6.

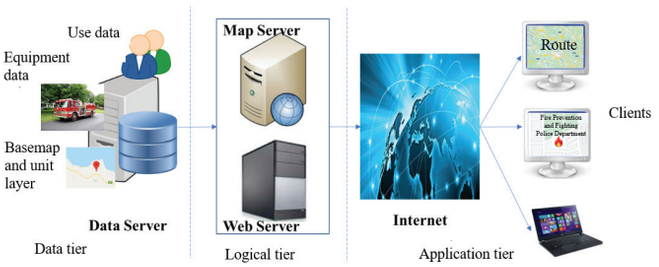

Fig. 5. Deployment architecture of the system.

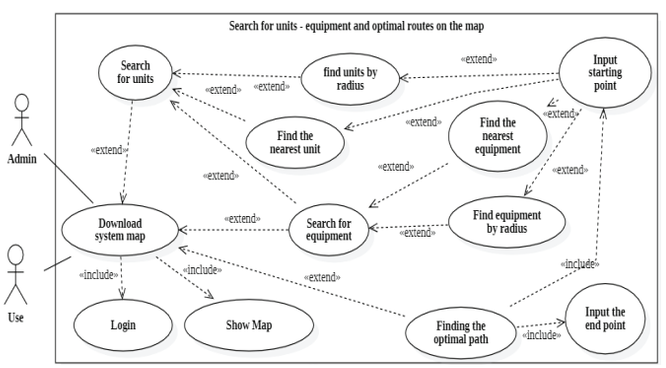

Fig. 6. Typical use case diagram of the system.

The main functional interface results of the system:

- The general layout interface of the system includes the following components: (1) Area describing information about the path, current user information, login/logout/change account information, and information about the number of device lists waiting for approval and viewing the user's operation history; (2) Working area with data tables or maps, including quick search with data, functions to add equipment data, accurately filter equipment by various criteria, output system charts list and output a report that prints the results after filtering; (3) The area where the equipment information list is displayed if the equipment menu is selected, and the selection operations to view, delete, and edit the equipment information. If you choose to enter the map menu, this area will display a map of units performing particular tasks. Users can perform operations to view unit information or analyse spatial problems to support decision support; (4) Menu area, to access specific functional groups such as Map function group, query execution, search for units and equipment on the map, optimal route search; The functional group views the equipment chart and validates the unit's equipment; Functional group of equipment management; Functional group to manage information describing the unit, including unit management, unit location management on the map; The system administration function group includes managing user accounts, managing user groups and adding data fields to the equipment table. In addition, financial equipment will be displayed in the list of equipment. You can click on the equipment name for detailed information, pictures, and attached equipment (Fig. 7).

- The function of spatial analysis is crucial in searching for units with equipment suitable for the task: it enables the search for units or equipment by radius at any location on the map, typically starting from where the disaster occurs. The search area is marked with a red teardrop, the centre of the circle on the map, and units within the circle radius are marked with a blue teardrop shape. This functionality is vital in mobilising the nearest response force within a certain radius, offering a rapid means to locate the appropriate equipment or units for the mission in the area to be mobilised, thereby assisting the commander in making prompt decisions (Fig. 8A). Currently, typical equipment management software does not include this function. If using standard spatial analysis, such as searching for firefighting units on map.google.com, the process involves mobilising all units that match keywords on the map (Fig. 8B).

- By applying the software as depicted in Fig. 8A, in the search area, there are four units, with only two matching units that can be filtered with the blue water drop icon. Next, the user will perform the second search phase to determine

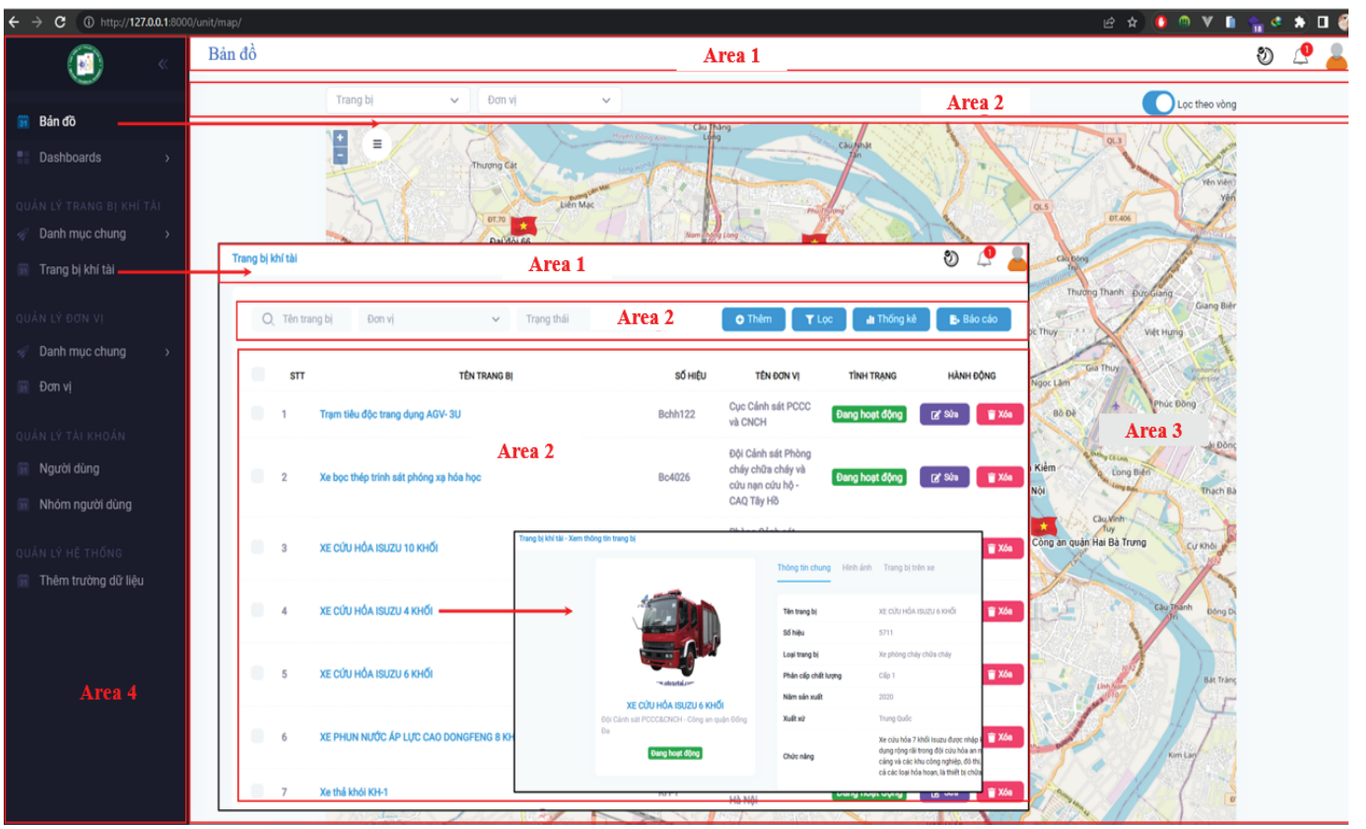

Fig. 7. The general interface of the software system.

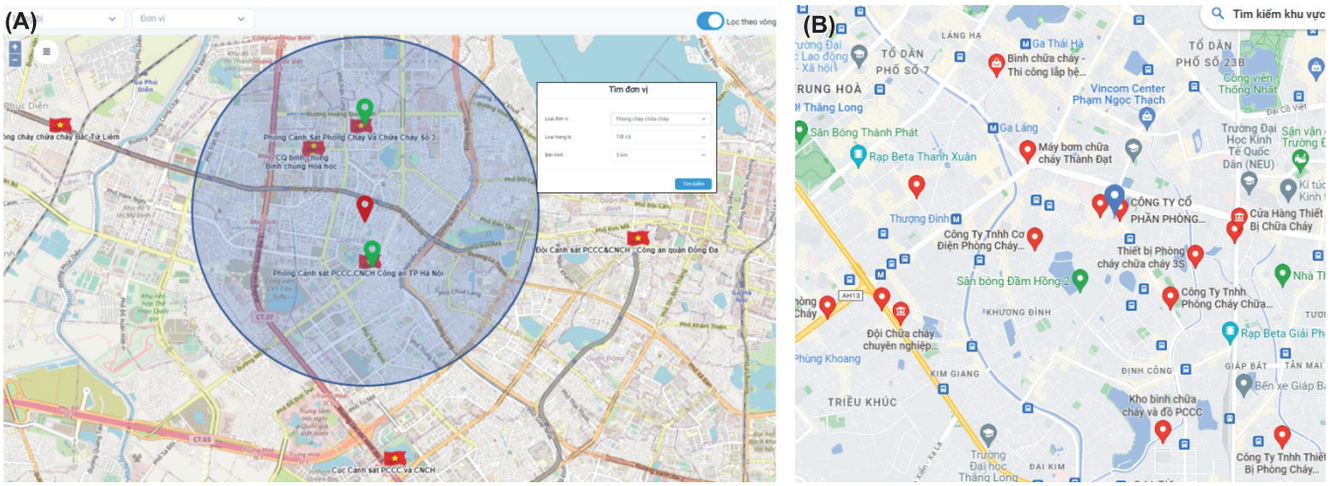

Fig. 8. Equipment type or unit type spatial query function by radius.

if the shortest path from the unit's location to the incident area is feasible. The results were tested with two modes of travel: by car and on foot. These results are compared and evaluated against the optimal route-finding results of Google Maps:

Option 1, from Fire Prevention and Fighting Police Team No. 3 to the rescue point (Fig. 9): Travelling by car, the software calculated a distance of 1.09 km, while Google Maps calculated 1.2 km. Generally, the overall routes are the same; the primary difference lies in the starting point location (circled area), which stems from the different granularities in the route datasets. Both algorithms selected the closest route from the starting position on the map. For walking, the shorter route is preferred: 943 m for the software compared to 1.1 km for Google Maps.

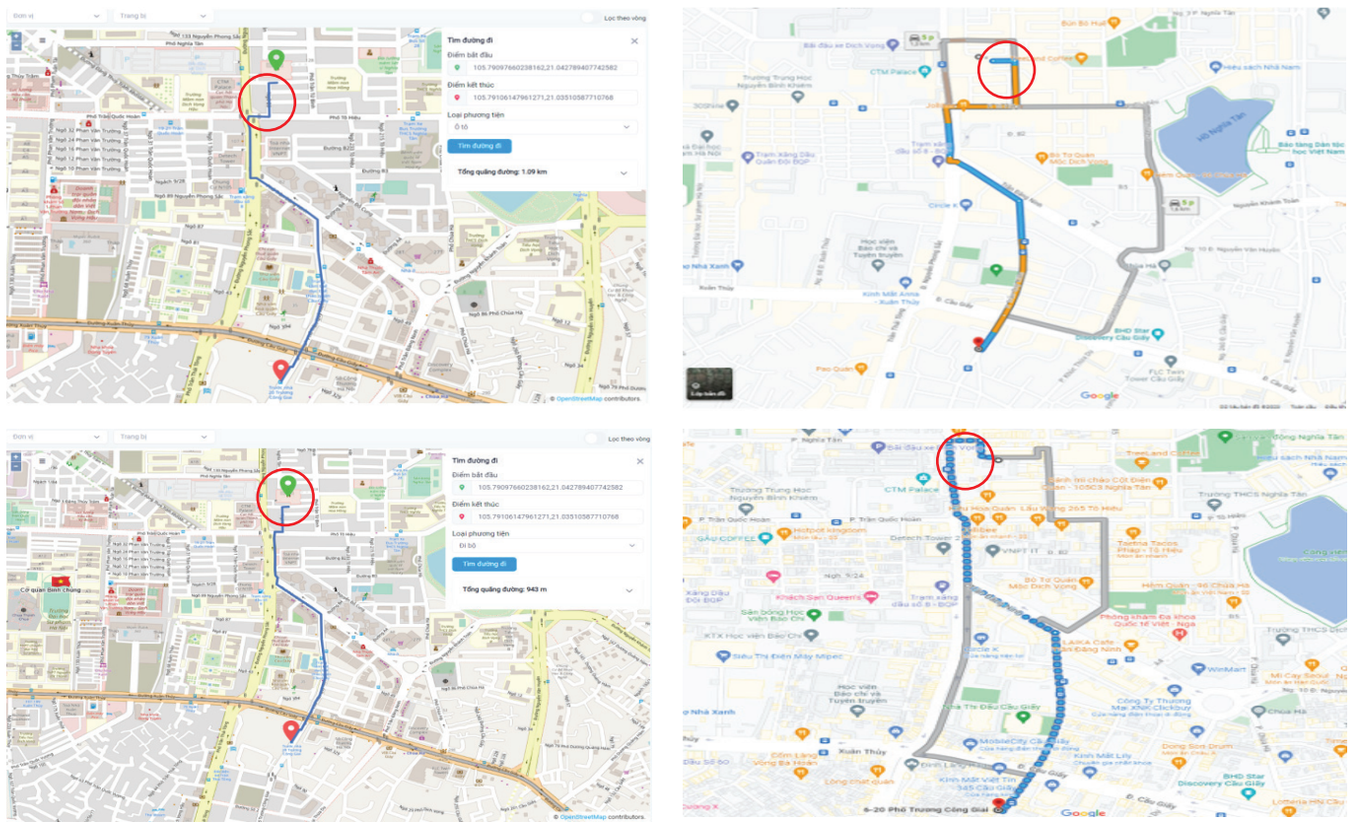

Fig. 9. Optimal path search results of option 1 (left side of software, right of Google Maps).

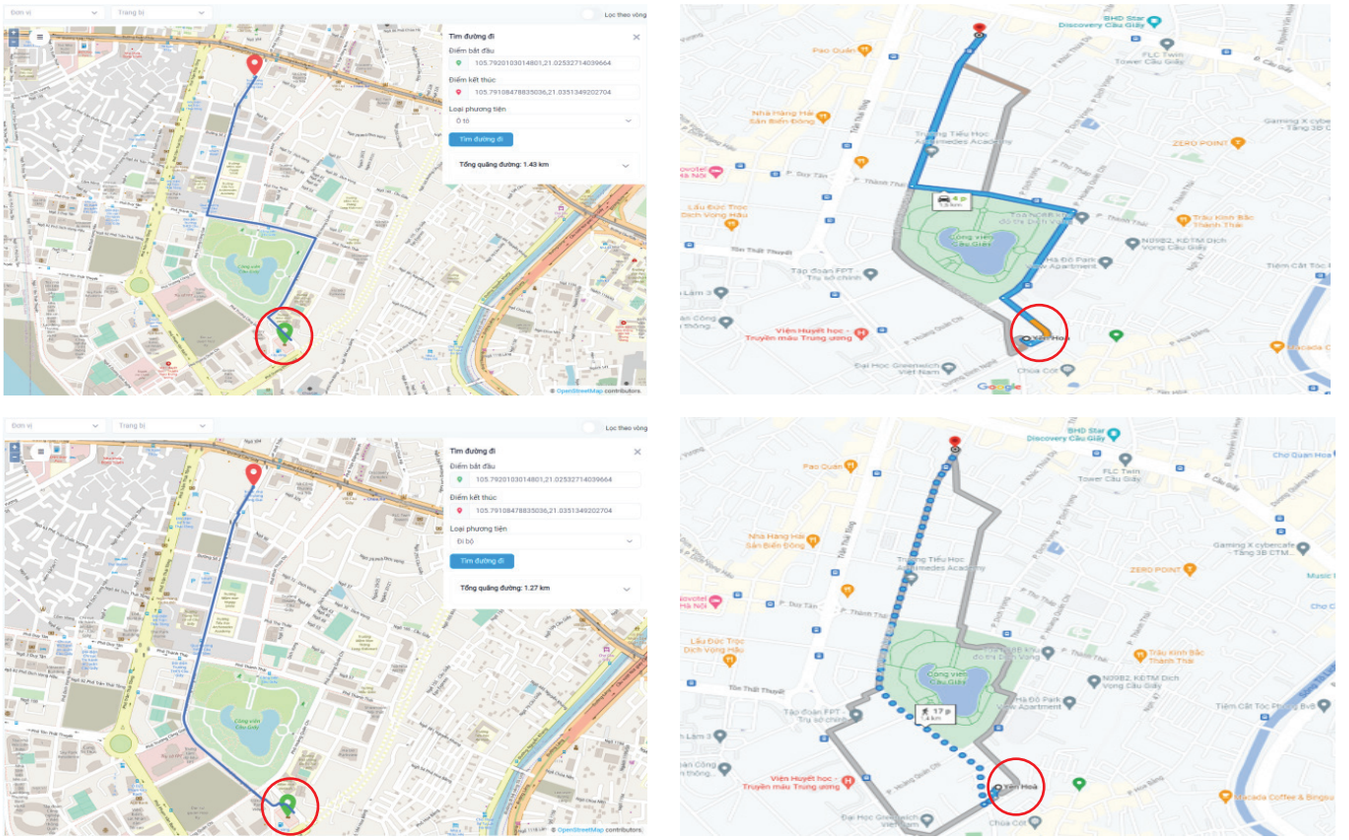

Fig. 10. Optimal path search results of option 2 (left side of software, right of Google Maps).

Option 2, from Hanoi City Fire and Rescue Police Department to the rescue point (Fig. 10): The shortest road length when travelling by car is 1.43 km for the software and 1.4 km for Google Maps. The optimal walking length is 1.27 km for the software and 1.4 km for Google Maps. This discrepancy, similar to the previous case, arises due to the differences in the route data near the starting points and the algorithms' nearest point capture mechanism. The optimal solution requires that the spatial data for the area be complete, adhering to the correct geographic database standards. This involves using spatial analysis methods to determine the connection point of the starting and destination areas with the connecting route to it, necessitating sufficient data collection between the area boundary layer and the traffic layer to ensure the system operates accurately.

Based on these test results, the software can select the most suitable option, achieving accuracy equivalent to the widely-used Google Maps navigation system. This is a crucial step in mastering the technology for building data management and decision support software for specialised fields involving sensitive and highly confidential data. The system operates efficiently and synchronously. In addition to the described functions, the system also includes a group of general administrative functions (monitoring user activity history, equipment statistics by chart, approving new or edited equipment by the administrator), user account modifications, and system login and logout functions; a functional group for user administration and a functional group for device-unit administration.

4. Conclusions

Rescue missions or operations are often urgent and require swift action to be effective. Thus, tools for managing information about human and material resources to support prompt and optimal planning are increasingly vital. The development of open-source geospatial technology (including maps, GIS, WebGIS, and Industry 4.0 technology) provides a favourable condition for developing a spatial decision support system. This study has investigated, designed, and programmed a software system for managing the spatial data of units and technical equipment to ensure specific combat tasks such as rescue and firefighting, chemical assurance, and engineer assurance are met. The paper focuses on spatial analysis to locate units with suitable firefighting equipment and the optimal manoeuvring route in the Hanoi area. The results have been evaluated to possess the same accuracy as Google Maps, marking an essential first step as a foundation to tackle more

complex spatial decision-making and management issues, such as spatial analysis using AHP techniques to find the optimal path based on multi-criteria, overlap analysis, etc. This will further develop systems for drafting spatial plans based on shared maps, approving plans, monitoring plan implementation, and gradually mastering specific systems.

COMPETING INTERESTS

The author declares that there is no conflict of interest regarding the publication of this article.

REFERENCES

[1] M. Wysokiński, R. Marcjan, J. Dajda (2014), "Decision support software for search & rescue operations", *Procedia Computer Science*, **35**, pp.776-785, DOI: 10.7494/csci.2015.16.3.281.

[2] J. Malczewski, P. Jankowski (2020), "Emerging trends and research frontiers in spatial multi-criteria analysis", *International Journal of Geographical Information Science*, **34(7)**, pp.1257-1282, DOI: 10.1080/13658816.2020.1712403.

[3] R. Tierno, B. Puig, B. Vera, et al. (2013), "The retail site location decision process using GIS and the analytical hierarchy process", *Applied Geography*, **40**, pp.191-198, DOI:10.1016/j.apgeog.2013.03.005.

[4] A. Mardoukhi, N. Kordzadeh (2016), "Site location determination using geographic information systems: The Process and a case study, *The Proceedings of The Southern Association for Information Systems Conference*, USA, 5pp.

[5] P.B. Keenan, P. Jankowski (2019), "Spatial decision support systems: Three decades on", *Decision Support Systems*, **116**, pp.64-76, DOI: 10.1016/j.dss.2018.10.010.

[6] B.A. Widyantoro, P.B. Santosa (2021), "Network analysis to determine the optimal route for firefighters in Makassar city", *IOP Conference Series: Earth and Environmental Science*, **936(1)**.

[7] O. Khaing, H. Wai, E. Myat (2018), "Using Dijkstra's algorithm for public transportation system in Yangon based on GIS", *Int. J. Sci. Eng. Appl.*, **7(11)**, pp.442-447, DOI: 10.7753/IJSEA0711.1008.

[8] D. Das, A.K. Ojha, H. Kramsapi, et al. (2019), "Road network analysis of Guwahati city using GIS", *SN Applied Sciences*, **1(8)**, pp.1-11, DOI: 10.1007/s42452-019-0907-4.

[9] N. Nolde (2020), "Open source routing engines and algorithms - An overview", https://gis-ops.com/open-source-routing-engines-and-algorithms-an-overview/, accessed 5 May 2023.

[10] N. Krismer, D. Silbernagl, M. Malfertheiner, et al. (2016), "Elevation enabled bicycle router supporting user-profiles", *28th GI-Workshop on Foundations of Databases*, **2016**, pp.74-79.

[11] K Samah, S Ibrahim, N Ghazali, et al. (2020), "Mapping a hospital using OpenStreetMap and GraphHopper: A navigation system", *Bulletin of Electrical Engineering and Informatics*, **9(2)**, pp.661-668, DOI: 10.11591/eei.v9i2.2082.

[12] R. Sugumaran, J. Degroote (2010), *Spatial Decision Support Systems: Principles And Practices*, CRC Press, 174pp.

[13] R. Can, S. Kocaman, A.O. Ok (2021), "A WEBGIS framework for semi-automated geodatabase updating assisted by deep learning", *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, **43**, pp.13-19, DOI: 10.5194/isprs-archives-XLIII-B5-2021-13-2021.

[14] R.M. Sanchez (2012) "Free and open source software for Geospatial Applications (FOSS4G): A mature alternative in the Geospatial technologies arena", *Transactions in GIS*, **16(2)**, pp.81-88, DOI: 10.1111/j.1467-9671.2012.01314.x.

[15] M. Molina, S. Bayarri (2011). "A multinational SDI-based system to facilitate disaster risk management in the Andean community", *Computers & Geosciences*, **37(9)**, pp.1501-1510, DOI: 10.1016/j.cageo.2011.01.015.

[16] OpenSource for GeoSpatial Fosters (2019), *Welcome to OSGeoLive 13.0-OSGeoLive 13.0 Documentation*, https://live.osgeo.org/en/index.html, accessed 19 October 2019.

[17] S. Coetzee, I. Ivánová, H. Mitasova, et al. (2020), "Open geospatial software and data: A review of the current state and a perspective into the future", *ISPRS International Journal of Geo-Information*, **9(2)**, DOI: 10.3390/ijgi9020090.

[18] M. Li, D. Peak, W. Zeng, et al. (2017), *An Open Source Webgis Framework For Collaborative Research In West Africa*, Spatial Knowledge Initiative Conference, Banff, AB, Canada, pp.1-9.

[19] M.A. Brovelli, M. Minghini, R.M. Sanchez, et al. (2017), "Free and open source software for geospatial applications (FOSS4G) to support future Earth", *International Journal of Digital Earth*, **10(4)**, pp.386-404, DOI: 10.1080/17538947.2016.1196505.

[20] Django (2023a), *Django Documentation Contents*, https://docs.djangoproject.com/en/4.1/contents/, accessed 5 May 2023.

[21] Django (2023b), *GeoDjango*, https://docs.djangoproject.com/en/4.1/ref/contrib/gis/, accessed 5 May 2023.

[22] Python Package Index (2023a), *Geographic add-ons for Django Rest Framework*, https://pypi.org/project/djangorestframework-gis/, accessed 5 May 2023.

[23] Python Package Index (2023b), *Django-GeoJson 4.1.0*, https://pypi.org/project/django-geojson/, accessed 5 May 2023.

[24] L. Cui, W. Li, X. Wang, et al. (2022), "Design and implementation of open-source WebGIS system for orchard land management", *Advances in Computer, Signals and Systems*, **6(3)**, pp.26-34, DOI: 10.23977/acss.2022.060304.

[25] Vue.js (2023), *What is Vue.js?*, https://vi.vuejs.org/v2/guide/, accessed 5 May 2023.

[26] OpenLayers (2023), *Tutorials*, https://openlayers.org/doc/tutorials/, accessed 5 May 2023.

[27] E.W. Dijkstra (1959), "A note on two problems in connection with graphs", *Numerische Mathematik*, **1**, pp.269-271.

[28] P.E. Hart (1968), "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics SSC4*, **2**, pp.100-107.

[29] GraphHopper (2023), *GraphHopper Directions API*, https://docs.graphhopper.com/#section/Custom-Model, accessed 5 May 2023.