Jurnal Rekayasa Elektrika

VOLUME 10 NOMOR 3 APRIL 2013

JRE	Vol. 10	No. 3	Hal 115–159	Banda Aceh, April 2013	ISSN. 1412-4785 e-ISSN. 2252-620x

A PostgreSQL/PostGIS Implementation for the Sightseeing Tour Planning Problem

Ardiansyah[1] dan Ruslan Rainis[2]
[1]Jurusan Teknik Elektro, Universitas Syiah Kuala
Jln. Tgk. Syech Abdurrauf No. 7, Banda Aceh 23111
[2]Geography Section, School of Humanities, Universiti Sains Malaysia
Pulau Pinang, Malaysia 11800
e-mail: razan_dad@yahoo.com

***Abstract*—This article discusses a procedure for finding the best multi stops route for sightseeing tour through a road network. The procedure involves building a database containing nodes and road network in PostgreSQL, calculating the shortest distance between a pair of nodes using pgDijkstra module, and solving the tour problem using a function written in PL/pgSQL. The function was developed based on the Nearest Insertion Algorithm for solving the Travelling Salesman Problem. The algorithm inserts a sightseeing attraction (node) at the best position in the existing route, which is between a pair of nodes that yields the minimum difference between the total tour time before and after the new node was inserted. The test result shows that the function can solve the problem within acceptable runtime for web application for total destination nodes of 22. It is concluded that the whole procedure was suitable for developing Web GIS application that solve the sightseeing tour planning problem.**

Keywords: ***tour planning problem, PL/pgSQL, PostgreSQL, Web GIS***

***Abstrak*—Artikel ini membahas prosedur untuk menemukan rute multi-hentian terbaik untuk wisata keliling melalui jaringan jalan raya. Prosedur ini melibatkan pembangunan sebuah database yang berisi titik tujuan dan jaringan jalan raya pada PostgreSQL, perhitungan jarak terpendek antara sepasang titik tujuan menggunakan modul pgDijkstra, dan memecahkan masalah wisata keliling menggunakan fungsi yang ditulis dalam PL/pgSQL. Fungsi ini dikembangkan berdasarkan Algoritma Penyisipan Terdekat untuk menyelesaikan Travelling Salesman Problem. Algoritma tersebut menyisipkan titik tujuan wisata (simpul) pada posisi terbaik dalam jalur yang ada, yaitu antara sepasang simpul yang menghasilkan perbedaan minimum antara waktu total wisata sebelum dan sesudah simpul baru dimasukkan. Hasil pengujian menunjukkan bahwa fungsi tersebut dapat memecahkan masalah dalam jangkauan waktu yang dapat diterima untuk aplikasi web dengan 22 simpul tujuan. Disimpulkan bahwa keseluruhan prosedur sesuai untuk pengembangan aplikasi Web GIS yang memecahkan masalah perencanaan wisata keliling.**

Kata kunci: ***masalah perencanaan wisata, PL/pgSQL, PostgreSQL, Web GIS***

I. INTRODUCTION

A web-based Geographical Information System (GIS), or web GIS, is GIS capabilities that are put on web sites [1]. In a Web GIS, a map is no longer a static image, but an interactive object that can be zoomed in or out and panned to a location of interest. Web GIS also makes it possible for web users to do few GIS analysis through the web without the need of GIS software.

Among the popular GIS analyses performed in the web is the shortest path analysis which searches the best route to travel from one place to another via a road network. This analysis helps people to find the best route to travel to a destination, but sometimes people need to arrange the best route to visit a number of places within specific period of time. An example when such a problem occurs is when a person wants to arrange a one day tour to visit tourist attraction within a city. Even though the shortest path analysis can be executed in Web GIS by using either commercial or free and open source software, a web GIS which provides function to solve the sightseeing tour planning problem is still difficult to find.

This article discusses a solution to the tour route planning problem as mentioned above. An implementation of the proposed solution using PostgreSQL is also discussed. Since PostgreSQL is free open source software that can serve data for web GIS, the solution will be very economical for web GIS development.

II. BACKGROUND

A. Tour Planning Problem

Generally, the problem of planning a tour route can be viewed as a variation of the Travelling Salesman Problem (TSP) because their objectives are similar, which is to find the best ordering of visiting several locations such that the cost of the route travelled is minimized. Vaughn et. al. categorized the problem as Travel Itinerary Planner Problem (TIPP) [2]. They considered the problem as

a Vehicle Routing and Scheduling Problem with Time Window (VRSPTW) which is a variation of the General Asymmetric Travelling Salesman Problem (TSP) in which the minimum tour has fixed origin and final destination node, and the tour should be completed within specified time constrained. The imposition of the time constraints cuts the total number of possible tours but adds the complexity of the problem at the same time [2].

A definition which is more specific to sightseeing tour planning was given by Godart [3]. He referred to the problem as the Trip Planning Problem (TPP) which seeks to find a route from an origin to destination with several stops at tourist attractions within a specified period of time such that the cost of activities (i.e. visiting sights), lodging possibilities and transportation is minimized and the attractiveness of the trip is maximized with respect to activities and lodging.

Joest and Stille defined the problem as the Enhanced Profitable Tour Problem (EPTP) which has the goal to find a route to visit nodes (tourist attractions) composing it only once within the given time while maximizing the prize of the nodes and arcs [4]. They simplified Godart's definition by not considering lodging possibilities. The prize of the nodes and arcs are assigned accordingly to reflect their position in user's preference. Besides, nodes are also weighted by the time required to visit them, and arcs are weighted by the distance between the nodes they interconnect [4]. The prize of nodes and arcs can combine the attractiveness, cost and other factor into one a single value for each of them.

B. Approaches to Solve Tour Planning Problem

Theoretically, an exact solution to the TSP can be found by picking the best tours after generating and evaluating all possible tours, but this method is computationally hard and inefficient for the TSP with many destinations [5]. If near-optimal solution is adequate, a heuristics algorithm can be used to significantly improved computational and hence a solution can be found faster.

The problem of planning tour route inherits the complexity of the TSP in addition to its extra constraints. Although there are various definitions of this problem, the approaches to their solution are similar in principal, which is making simplification to the problem and applying heuristics approach to obtain good solution in a reasonable amount of time. The simplification may be done by reducing the complexity of the problem or by relaxing some constraints.

Vaughn et. al. proposed a framework to solve the problem which comprises three basic algorithms to select a set of feasible destinations, provide the minimum path between each pair of the selected destinations, and find the solutions to the time constrained TSP [2].

The above framework requires integration between the algorithms and databases to support travel itinerary planning [2]. Vaughn further specified that the required database includes network related data such as road links and nodes, average speeds and speed limits on each road link; turn penalties. In addition to those data, the database should also contain destination related data and relevant maps which include detailed street level maps with addressable locations.

Another approach proposed by Joest and Stille implemented Vaughn's algorithm to reduce the graph size for solving the routing problem [4]. They used Dijkstra shortest path algorithm to find the shortest path between all pairs of places of interest and then used the result to replace the original distances between places of interest. Dijkstra algorithm is often used for searching the shortest path between each pair of destinations in commercial car navigation systems [7]. The nodes that do not represent places of interest are deleted from the graph and the real path is saved. The result is a complete graph consisting only of places of interest and user-optimal paths between them [4]. This reduction is important in facilitating solution finding of the traveling salesman problem (TSP) because the difficulty of the TSP increases as the number of nodes grows.

Joest and Stille solved the reduced graph for the tour route by applying an algorithm based on the insert/delete heuristic of Mittenthal and Noon [4]. They generated a tour by iteratively inserting the best possible node in each step while keeping the total time within the given constraint. They calculated the ratio of the change in cycle prize to the change in cycle time for each node, which is currently not in the tour, if the node is inserted between each pair of nodes in the tour.The largest ratio indicates the best node to be inserted at the optimal position between a pair of node in the tour at the next step. The iteration stops when there is no more feasible node to be inserted. The tour obtained is then improved by considering a set of nodes excluded from the tour while maintaining the tour structure in everyiteration. This step is performed until there is no further improvement gained [4].

C. PostgreSQL

PostgreSQL is an open-source object-relational database management system (ORDBMS) derived from POSTGRES Version 4.21. PostgreSQL supports most of the SQL standards and offers many modern features, such as complex queries, foreign keys, triggers, views, transactional integrity, and multiversion concurrency control. PostgreSQL gives flexibility for users to extend it by adding data types, functions, operators, aggregate functions, index methods, and procedural languages [8].

One important PostgreSQL feature is a loadable procedural language for PostgreSQL database system. The language, which is called PL/pgSQL, combines the power of procedural language with the ease of use of Structured Query Language (SQL), the standard language for accessing and manipulating data in relational database. PL/pgSQL can do anything that can be defined in C language functions except for input/output conversion and calculation functions for user-defined types. PL/

pgSQL gives significant advantage in performing a group of queries to the database. Every SQL statement must be executed individually by the database server. Normally, a series of query must be sent one by one from client application to the database server. Each of them requires certain amount of time due to interprocess communication. Additional time for network overhead is necessary if the server and clients located in different machine. A function that performs a block of computation and a series of queries inside the database server can be created by using PL/pgSQL. Client only needs to send a single SQL statement which invokes the function to perform a group of queries, and thus a lot of time can be saved.

A set of functions which are combined to accomplish certain task is called a module. An example of such module is Pgdijkstra, which was created by Sylvain Pasche for performing shortest path queries based on the Dijkstra algorithm. The core of this module is a function which computes a shortest path from a set of edges and vertices.

Capabilities of PostgreSQL can be exetended by installing extension to the DBMS software. One example is PostGIS, which is an extension to the PostgreSQL that allows GIS (Geographic Information Systems) objects to be stored in the database. It also provides supports for spatial indexes as well as functions for analysis and processing of GIS objects. PostGIS is necessary for PostgreSQL to supply spatial data for a Web GIS and requirement of PgDijkstra module.

III. METHOD

In general, the solution to the sightseeing or city tour planning problem discussed in this article follows the method of Joest and Stille [4], which starts by building street network dataset, simplifying the network, and solving the reduced network (graph). The difference is in the algorithm for solving the TSP in the reduced network. In this article, the solution to the TSP is based on the Nearest Insertion Algorithm for solving TSP [5] with some modification. The algorithm is summarized below:

1. Initialize set *R* with the start node *i* and set *T* to contain the candidates for destination nodes. The total time spent in the tour (t_{tot}) is equal to 0 (zero).
2. Remove a node from set T, insert it into set *R* and place it after the node *i*.
3. Remove a node *s* from set *T*. Then for each pair of nodes *u* and *v* in set *R*, calculate the change regarding the time (or cost), Δt_{suv}, if node s is inserted between that node. Δt_{suv} is defined as:

$$\Delta t_{suv} = t_{us} + t_{sv} - t_{uv} \quad (1)$$

where:
t_{us} is the time or cost of travelling from node *u* to node *s*
t_{sv} is the time or cost of travelling from node *s* to node *v*
t_{uv} is the time or cost of travelling from node *u* to node *v*
Record the minimum Δt_{suv} and the nodes *u* and *v* which give the minimum value.

4. Check whether or not the following constraint is satisfied:

$$\min(\Delta t_{suv}) + t_s \le t_{\max} - t_{tot} \quad (2)$$

where:
t_{us} is the time spent at node *s* (node cost)
$\min(\Delta t_{suv})$ is the minimum Δt_{suv}
t_{max} is the user specified maximum tour time
t_{tot} is the total time spent in the tour
5. If equation (2) is satisfied, the node s will be inserted between the pair of nodes u and v which gives the minimum Δt_{suv}.
Otherwise, the node s will be discarded and the process is repeated from Step 3.
6. After adding the node s, update the total tour time (t_{tot}) by adding $\min(\Delta t_{suv}) + t_s$ to the current ttot.
7. Repeat Step 3 to 7 until all the nodes in *T* are inserted into set *R*.

The destination nodes were ranked based on their popularity. The algorithm ensures the most attractive destinations are included in the list as long as time permits by picking the highest ranking node available in each iteration.

The sightseeing or city tour problem has limited destinations (nodes). Therefore, it is possible to calculate the shortest time (or least cost) to travel from a node to another using the Dijkstra algorithm and stored the result in a table in the database. This approach actually simplifies the road network because there are only limited number of nodes and the shortest time for each pair of nodes which need to be considered for solving the TSP based on the above algorithm.

IV. IMPLEMENTATION DAN DISCUSSIONS

Before implementing steps explained in the Method section, a database which stores sightseeing attractions (nodes) and road network data was built. There were 22 sightseeing attractions used for testing the implementation, while the road network consisted of around 1000 line segments. Those spatial data are shown in Figure 1. Spatial data, such as points representing sightseeeing attractions and lines representing the road, were prepared using GIS software and then converted into PostgreSQL format. Storing of spatial data in a PostgreSQL database is possible when PostGIS extension is added to the DBMS software.

The ranking of each sightseeing attraction relative to the other and estimated time spent at the attraction (node cost) were stored along with the node spatial data in a table named node. The road spatial data of the road as well as the attribute data, including name of roads, class of roads, and time needed to travel through each road were stored in other table. The road network was built from the road data by using a function which is available in pgDijkstra module. The shortest time to travel from an attraction to another was then calculated by using another function in pgDijkstra module, and the results were stored in the

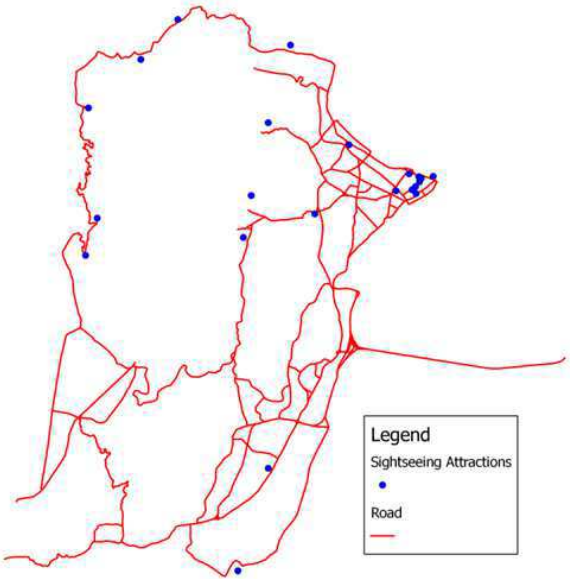

Figure 1. Sightseeing attractions and road data used for testing the implementation of the algorithm

database in a table named path.

Implementation of the algorithm described previously in the Method section required searching and retrieving appropriate data from node and path tables. Those tasks can be done quickly and easily by using SQL statement. Since determining the best position to insert a node requires an iterative process of finding the position that gives the minimum cost, SQL statements which retrieve appropriate data from the database must be executed repeatedly. As it is discussed previously, the best way to execute a block of computations and a series of queries in PostgreSQL is by creating a function using PL/pgSQL language. Such function can perform the entire algorithm efficiently within PostGIS/PostgreSQL. Therefore, a PL/pgSQL functions named *make_route()* was created inside the database. This function generates the best route based on user-specified maximum tour time. The function takes the starting node ID, the maximum tour time in minutes, and mode of tour (one way or round trip) as the inputs. The functions return a set of records that contains data of the resulting route, which are node ID, time spent at a node, and travel time between from a node to the next node.

Function *make_route()* checks whether or not the resulting route violates the given maximum tour time. If inserting a node in its best position will cause the total tour time exceeds the maximum time, the node will not be inserted. The function will find another node which can be inserted into the route without violating the given maximum time. This function will always test all nodes in the node table because it tries to include as many nodes as possible into the route for the given maximum tour time. Nodes are retrieved from the database according to their rankings, starting from the highest ranking. No matter what is the value of the maximum time, function *make_route()* will do the iteration until all the nodes are retrieved and tested.

The function uses three arrays to hold information of the generated route, which is referred to as set R in the algorithm. The first array keeps the node ID of all nodes that makes up the route. The second array keeps the node cost of each node in the first array. The node cost of the i-th element in the first array is the i-th element in the second array. The third array stores the cost of travelling from a node to the next node. The i¬-th element of this array is a cost of travelling from the i-th node to the (i+1)-th node in the first array. The last element of the third array will be 0 (zero) because the tour ends at that node (sightseeing attraction).

In the case of one-way trip mode, the first array is initialized with a start node. The best position to insert a node is determined by calculating the change regarding the time (or cost), Δt, if the node is inserted between the i-th and the (i+1)-th element of the array. The calculation follows the steps given in the algorithm. Initially, the cost of travelling from the last element of the array to the new node is assigned as the minimum Δt, andthe index of the last element is stored as the best position. Then, nodes are retrieved from the node table one-by-one, starting from the highest ranked node. Each time a node is retrieved, two values of shortest time are retrieve from path table. One is for traveling from the i-th element to the node (or, to-node cost), and the other is for travelling from from the node to the (i+1)-th element (or from-node cost). For each pair of the i-th and (i+1)-th element of the array, the Δt is calculated and then the result is compared to the current minimum Δt. If the new Δt is smaller, then it replaces the current minimum Δt. The index i is stored as the best index (*min_idx*). The cost of traveling from this i-th element to the new node is stored as the minimum to-node cost, whereas cost of traveling from the new element to the corresponding (i+1)-th element is stored as the minimum from-node cost. After all the elements in the first array are checked, minimum Δt is used to check whether Eq. (2) is satisfied. If so, the new node is inserted into the first array between the *min_idx*-th and (*min_idx*+1)-th elements. The second array is updated by inserting the node cost at the same position as its corresponding node. The third array is updated by replacing the *min_idx*-th element with the minimum to-node cost and inserting the minimum from-node cost after that element. Those updated arrays are then used to determine the best position to insert the next node.

For the round trip mode, the first array is initialized with two elements and both of them are the start node. This is to indicate that the tour starts and ends at the same place. The new nodes will be inserted between these two nodes. Initially, the minimum Δt is set to 999999. The procedure of determining the best position and inserting the new node is the same as in the case of one-way trip described above.

The function was tested by executing it through an SQL SELECT query statement in pgAdmin tools of PostgreSQL. The query statement invokes the function to generate one-way tour route from a selected starting point and specified maximum tour time. The tests were conducted by increasing the maximum tour time gradually while keeping the starting point the same throughout the

test. The test was repeated three times for each selected maximum tour time, and the average execution time of the query was calculated for each case. The tests were conducted using an NEC PowerMate PC Series with 3.0 GHz Intel Pentium® 4 CPU and 512 MB of RAM. The test results are summarized in Table 1.

The number of available sightseeing attractions (nodes) in the database used in the test was 22. Thus, the test stopped at the maximum tour time of 14.5 hours because all the nodes were visited when this values was input. Table 1 shows that the maximum average run time of *make_route()* function was 82.8 ms.

A study on Quality of Service for Web conducted by Bouchet. al. found that the threshold where QoS as 'Low' is around 11 seconds [9]. Nielsen stated that users start to feel the web is slow if the response time is longer than 1 second, while delays greater than 10 seconds cause users to lose their focus to the operation and switch to other task [10]. Therefore, the average run time of 82.8 ms is acceptable for Web application because it does not add much to the web response time. The function can be used for finding the best sightseeing or city tour route in a Web GIS application as long as the number of sightseeing attractions is not too many. The result shows the runtime is still acceptable (much less than 0.1 seconds) when the number of nodes is 22.

Even though the *make_route()* function always checks all the available nodes in the database, its running time still varies according to given maximum tour time. This variation occurs because *make_route()* function traverses two loops. The outer loop depends on the number of nodes available in the database, while the inner loop depends on the number of nodes which are already inserted in the array. Less tour time allows less nodes inserted into the array, and thus, less iteration needed for checking whether a new node can be inserted.

Table 1. Results of executing *make_route()* function with different maximum tour time

No	Maximum Tour Time (hours)	Average run time (ms)	Number of places visited in the route
1	1.0	21.8	2
2	1.5	35.8	3
3	3.0	40.8	5
4	5.0	49.7	7
5	8.0	67.2	12
6	10.0	76.8	15
7	12.0	77.9	18
8	14.0	82.9	21
9	14.5	82.8	22

V. CONCLUSIONS

This article discussed a procedure for finding the best multi stops route for sightseeing tour through a road network. The procedure was developed based on the Nearest Insertion Algorithm. The algorithm inserts a sightseeing attraction (node) at the best position in the existing route, which is between a pair of places that yields the minimum difference between the total tour time before and after the new node is inserted. A node is inserted into an existing route only if the resulting tou time does not violate the given maximum tour time. In this case the time spent at each node is also added up to the tour time.

The algorithm was implemented as a function written in PL/pgSQL language ofPostgreSQL DBMS software. The function was tested to solve sightseeing tour planning problem with different maximum tour time, and was able to give solution in much less than 0.1 seconds for 22 destination nodes. Since PostgreSQL can provide data for Web GIS, adding the function, along with PostGIS extension and pgDijkstra module, to PostgreSQL will make it possible to develop a Web GIS application which can provide solution to the sightseeing tour planning problem.

REFERENCES

[1] K. Zheng, T. R. Soomro, and Y. Pan, "Web GIS: Implementation issues," *Chinese Geographical Science*, vol. 10, no. 1, pp. 74–79, Beijing, China: Science Press, 2000.

[2] K. M. Vaughn, M. A. Abdel-Aty, and R. Kitamura, "A framework for developing a daily activity and multimodal travel planner," *International Transactions in Operational Research*, vol. 6, pp. 107–121, 1999.

[3] J. M. Godart, "Using the trip planning problem for computer-assisted customization of sightseeing tours," in *Information and Communication Technologies in Tourism*, P. J. Sheldon, K. W. Wober, and D. R. Fesenmaier, Eds., Montreal, Canada: Springer, 2001.

[4] M. Joest and W. Stille, "A user-aware tour proposal framework using a hybrid optimization approach," *Int. Proc. of 10th ACM Int'l Symp. Advances in GIS*, 2003.

[5] D. S. Johnson and C. H. Papadimitriou, "Computational complexity," in *The Traveling Salesman Problem*, E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan and D. B. Shmoys, Eds. New York, NY: John Wiley & Sons, 1985.

[6] D. S. Johnson and C. H. Papadimitriou, "Performance guarantees for heuristics," in *The Traveling Salesman Problem*, E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan and D. B. Shmoys, Eds. New York, NY: John Wiley & Sons, 1985.

[7] A. Maruyama, N. Shibata, Y. Murata, K. Yasumoto, and M. Ito, "A personal tourism navigation system to support traveling multiple destinations with time restrictions," *18th International Conference on Advanced Information Networking and Applications (AINA'04)*, 2004.

[8] The PostgreSQL Global Development Group. (Oct. 11, 2006). PostgreSQL 8.1.4 Documentation [Online]. Avalaible: http://www.postgresql.org/docs/manuals/.

[9] A.Bouch, A. Kuchinsky, and N. Bhatti, "Quality is in the eye of the beholder: Meeting Users' Requirements for Internet Quality of Service," *Proceedings of CHI2000 Conference on Human Factors in Computing System*, 2000.

[10] J. Nielson, *Usability engineering*, Boston , MA:AP Professional Press, 1994.

Penerbit:
Jurusan Teknik Elektro, Fakultas Teknik, Universitas Syiah Kuala
Jl. Tgk. Syech Abdurrauf No. 7, Banda Aceh 23111
website: http://jurnal.unsyiah.ac.id/JRE
email: rekayasa.elektrika@unsyiah.net
Telp/Fax: (0651) 7554336