*International Journal of Research in Advent Technology, Vol.3, No.12, December2015*
*E-ISSN: 2321-9637*
*Available online at www.ijrat.org*

# MapBuddies: Web Application for the Travelling Salesman Problem

Kuo-pao Yang[1], Ranjan Poudel[2], Nishant Jha[3], Grace Chenevert[4]
*Department of Computer Science and Industrial Technology*[1, 2, 3, 4]
*Southeastern Louisiana University*[1, 2, 3, 4]
*Email: kyang@selu.edu*[1]

**Abstract-** This paper discusses a case study of the development of a web application for the Travelling Salesman Problem (TSP). The modern technologies and programming languages are used to solve the vehicle routing problem, implement a web site, and display a routed map for users.

**Index Terms-** Programming Languages, Travelling Salesman Problem, Google Maps.

## 1. Introduction

There are many applications developed for the Travelling Salesman Problem (TSP). The MapBuddies development team uses modern technologies and computer languages to implement various algorithms and then provides the solution for the TSP problem.

The Travelling Salesman Problem [8], which originates in the 1800's, is a very famous path problem. The concept for the TSP is to imagine a travelling salesman who must start from his current location and finish his journey at a given endpoint. He must visit multiple other destinations before he can reach the endpoint. The salesman's goal is to visit each location exactly once while minimizing his travel distance. There are many applications to find solutions for the Travelling Salesman Problem. One of the most common applications is to find the shortest and the most cost efficient path for driving a car.

There are many different methods used to solve the Travelling Salesman Problem [9]. It has multiple considerations to minimize [14] factors such as gas consumed, distance travelled, and time needed. The more objectives present in a TSP problem, the more complicated the algorithms to solve it. In the most common Travelling Salesman Problem, the path is minimized by a single constraint. In the case of driving a car, the path is typically found by minimizing only the distance.

This project uses modern technologies and programming languages to implement the Travelling Salesman Problem. The TSP [2] has been attempted and implemented many times in software. This project provides a solution for the Travelling Salesman Problem by coding in Ruby [1] with Ruby on Rails [12] framework, receiving a routed map in Google API calls, and then showing the shortest path on the map. Javascipt, CoffeeScript, and HTML5 are used to perform functions between user interaction and backend code.

The Travelling Salesman Problem has been widely studied. A variety of applications that have been shown to exhibit TSP problems include in computer wiring [11], wallpaper cutting [6], material handling [15], job sequencing [7], hole punching [16], dartboard designing [5], genome sequencing [3][4], and vehicle routing [17].

This paper discusses a case study of the implementation of a website that provides a solution of the vehicle routing problem for the TSP. The web application finds the shortest path by visiting all destinations, which are entered by a user. The problem is introduced, and the Brute Force Search and Dijkstra's algorithm are discussed. Full implementations for both algorithms are developed and detailed in this paper. Finally, conclusions are presented and issues for future research are discussed.

## 2. The Problem

This project is a web application to find the shortest route between multiple destinations. This web application is called MapBuddies. A user would input destinations to the web application. The first entered destination is view as the starting point and the last entered destination is the ending location. The backend system of the MapBuddies site would then find the coordinates for each destination from a local database source and Google Maps API [10] calls. After the coordinates are received, the MapBuddies site will process the information to find the solution to the TSP for the given destinations. After the path is found, it will be displayed on a map to give the user both visualization and textual representation of the path. This basic flow of the MapBuddies site is shown in Fig. 1.

The MapBuddies project has developed an operational website. It also includes extra functionality that allows the user to email the map showing the path to each destination. The MapBuddies would send an email that includes a link with the coordinates and display the results on the map. The MapBuddies development team has also developed multiple methods for entering destinations. Lastly, the destinations inputted by the user are tested for validity. The ultimate goal of the MapBuddies site is to provide users with an easy-to-use interface that finds the solution for the Travelling Salesman Problem, and presents it in an easy-to-understand and visually pleasing method.

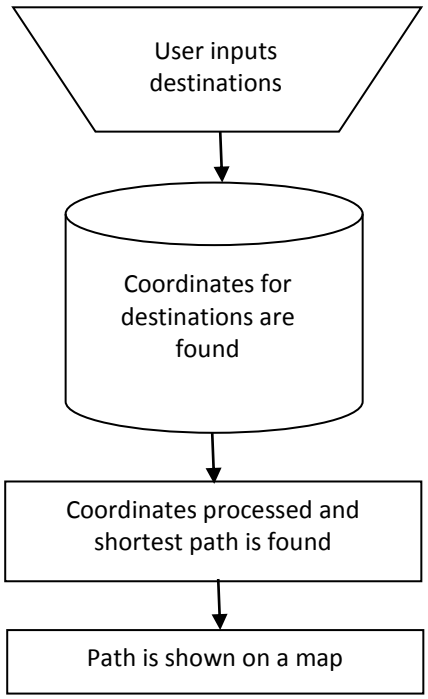


Fig. 1. The Basic Flow of the MapBuddies Site

The Travelling Salesman Problem belongs to the class of NP-complete problems [14]. Thus, it is possible that the worst case running time of any algorithms and number of destinations for the TSP increases exponentially. To find the solution to the Travelling Salesman Problem, the development team has researched multiple algorithms. There are two algorithms that have been studied and implemented: the Brute Force Search and Dijkstra's algorithm. For the purpose of the MapBuddies project, which will solve the TSP for a maximum of five destinations, simplicity of design and implementation are greater concerns than efficiency for many destinations.

## 3. Implementation

### 3.1. *Technologies*

The main languages used for development of the MapBuddies project are Ruby and Ruby on Rails framework. Ruby and Ruby on Rails are used for the project because of their ease of use and increasing popularity. As an interpreted language, Ruby is very dynamic, which means easier to read and makes programming simpler and faster [13]. Ruby has unique features that can be helpful for solving problems like the Travelling Salesman Problem. For example, looping in Ruby through arrays and hashes, as well as appending new items to an array, becomes very simple tasks. Ruby on Rails is an open source web framework and is also designed to be easy to use. With Ruby on Rails [18], new web pages or blogs can be easily created with just a few lines of code.

Ruby and Ruby on Rails, however, are not the only language and framework that are used in this project. The development team also uses JavaScript, CoffeeScript, and HTML5 to program various components of the project. JavaScript is a popular scripted language that is designed to resemble C language. CoffeeScript is a more condensed version of JavaScript that is designed to be similar to languages like Python. These languages are used to perform functions between user interaction and backend code. HTML5 is highly used in the MapBuddies site to improve the aesthetics of the site.

To develop the MapBuddies site, the development team researched different Interactive Development Environments, or IDEs. RubyMine, one of the Ruby and Ruby on Rails IDEs, was originally considered. After further research, Aptana Studio 3 was found to be more practical for the MapBuddies development team. Aptana is an open source development tool for open websites. It supports the major web browser technology and programming languages such as HTML5, JavaScript, Ruby, Ruby on Rails, PHP, and Python. For source control, the MapBuddies development team uses the free source control hosting tool, Bitbucket. Additionally, the development team chooses to use the Twitter Bootstrap to make the MapBuddies site responsive and cleaner.

Further research was conducted to find the appropriate database service and the best way to display a path on a map. Ruby on Rails framework comes with the default database SQLite. SQLite is a small scale database and the most useful for testing purposes. The development team uses MySQL as the database for MapBuddies. MySQL is a larger scale database that is more useful for actual implementation and larger projects. To find and display paths on a map, the MapBuddies development team uses Google

Maps API. It has useful functions for accepting and displaying route information. JavaScript and CoffeeScript are used to access the Google Maps API.

Each pair would be an API call to Google Maps. With up to five destinations, it means up to twenty API calls. With Google Maps API calls, it would automatically calculate the solution to the TSP with a single API call. The development team found Google Maps API has a limit on the number of calls that can be placed during a 24 hour period and per second. Due

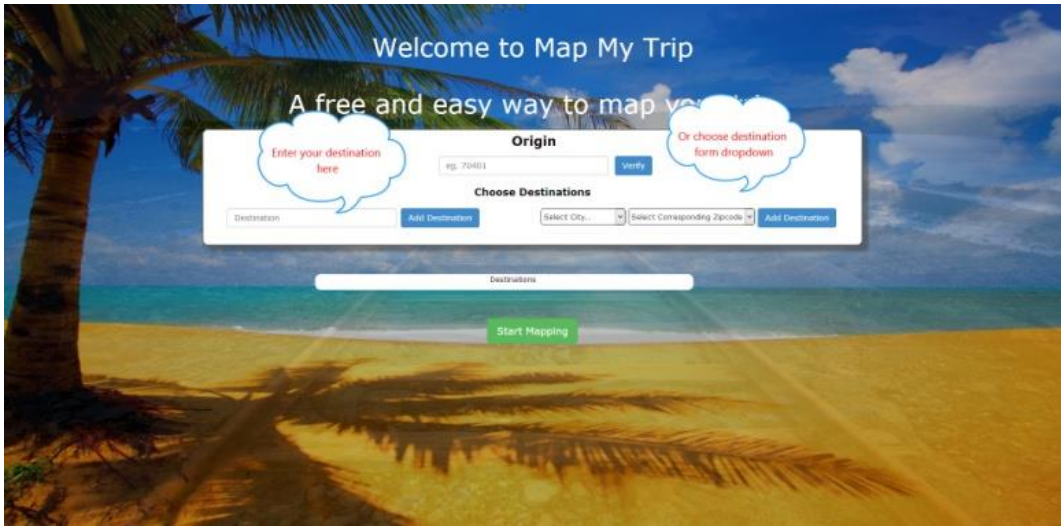


Fig. 2. Two Different Methods of Entering Destinations

### 3.2. *Development*

The Mapbuddies site allows the user to input destinations in two different ways. The first way would be to type in a city, address, ZIP code, or attraction. This method would send the text to Google Maps API to find the coordinates. The other method is to select a ZIP code by city for destinations in the state of Louisiana. This method uses a MySQL database that the development team created and populated with cities, ZIP codes, and coordinates. In Fig. 2, it shows the two different destination input options.

After the user inputs the destinations, the information is sent to the backend. At this point, the TSP is solved. Both the Brute Force Search and Djkstra's Algorithm were considered. The Brute Force Search finds every possible route. After it has found each path, it compares all of them and finds the shortest path. Dijkstra's Algorithm starts at a given location and travels to the closest destination to itself. It does this at each destination until it has visited all stops exactly once. After comparing performance, the development team for MapBuddies decided to emulate Dijkstra's Algorithm. Fig. 3 shows Ruby based pseudo code of this algorithm for a round trip. For round trips, the starting and ending points are the same.

To achieve an optimal display of this information using Google Maps API, the actual driving distance between every pair of destinations needs to be found.

```
distance_hash = {a: [[b, x],[c, x],[d, x],[e, x]], #hash contains distance
                 b: [[a, x],[c, x],[d, x],[e, x]],          #between every 2 points
                 c: [[a, x],[b, x],[d, x],[e, x]],
                 d: [[a, x],[b, x],[c, x],[e, x]],
                 e: [[a, x],[b, x],[c, x],[d, x]]
                 }
path = [a]       # 'a' is the first destination visited
total_dist = 0          #This will hold the total size of the path
next_city = a           #This sets the next key in the hash to be examined

while(number of elements in path is less than number of cities)
      distance_hash[next_city].each do |to, distance|       #increment
            min_dist = 9999999                              #through hash
            if(to not in path)
                  if(distance < min_dist){
                        min_dist = distance
                        next_city = to      #find closest
                  end
                  total_dist = total_dist + min_dist
                  path << next_city     #add visited destination to path
            end
      end
end
path << a       #return to original point (round trip)
distance_hash[next_city].each do |to, distance|   #add distance to total
      if(to == next_city)
            total_dist = total_dist + distance
      end
end
puts path
puts total_dist
```

Fig. 3. Dijkstra's Algorithm for the Travelling Salesman Problem

to the constraint on the number of API calls, the delay is caused by calling the API excess times. Therefore, the MapBuddies development team implements the TSP solution in house. Fig. 4 shows a mapped solution for the Travelling Salesman Problem.

To allow the user to email map information, the MapBuddies team develops the Simple Mail Transfer Protocol (SMTP) plugin for Ruby on Rails. Using this plugin, the development team is able to allow the user to enter an email address and then send the routed map information. After the user types an email address into the appropriate box, the MapBuddies site then verifies the email address. Once the address is verified, the information is sent in the form of a link via an email message. If the link is followed, the user will be directed to the MapBuddies site and the link will populate the map with the necessary information.

## 4. Conclusion

The team of MapBuddies has successfully developed a web application for the Travelling Salesman Problem. The modern technologies and programming languages are used to solve the vehicle routing problem, implement a web site, and display a map for users. The MapBuddies development team uses Ruby, Ruby on Rails, Google Maps API, CoffeeScript, JavaScript, and HTML5 for the vehicle routing problem. This project implements for the Travelling Salesman Problem by coding in Ruby with Ruby on Rails framework, receiving a routed map in Google Maps API calls, and then displaying a shortest path on a web browser. JavaScript and CoffeeScript are used to access the Google Maps API. HTML5 is used to improve the aesthetics of the site.

In the future, the MapBuddies development team will study and implement more algorithms for the TSP problems and provide more destinations for the NP-complete problems.

## REFERENCES

[1] Baas, B., "Ruby in the CS Curriculum," Journal of Computing Sciences in Colleges 17(5): 95-103, April, 2002.
[2] Bellman, R., "Dynamic Programming Treatment of the Travelling Salesman Problem," Journal of the ACM 9(1): 61-63, 1962.
[3] Ben-Dor, A. and Chor, B., "On Constructing Radiation Hybrid Maps," Journal of Computational Biology 4, 517-533, 1997.
[4] Ben-Dor, A., Chor, B., and Pelleg, D., "RHO-Radiation Hybrid Ordering," Genome Research 10, 365-378, 2000.
[5] Eiselt, H.A., and Laporte, G., "A Combinatorial Optimization Problem Arising in Dartboard Design," Journal of the Operational Research Society 42, 113-118, 1991.
[6] Garfinkel, R.S., "Minimizing wallpaper waste, Part I: A class of traveling salesman problems," Operations Research 25, 741-751, 1977.
[7] Gilmore, P.C., and Gomory, R.E., "Sequencing a One State-Variable Machine: A solvable case of the traveling salesman problem," Operations Research 12, 655-679, 1964.
[8] Hoffman, K., Padberg, M., and Rinaldi, G., "Traveling Salesman Problem," Encyclopedia of Operations Research and Management Science, 1573-1578, 2013.
[9] Kai, A. and Mingrui, X., "A Simple Algorithm for

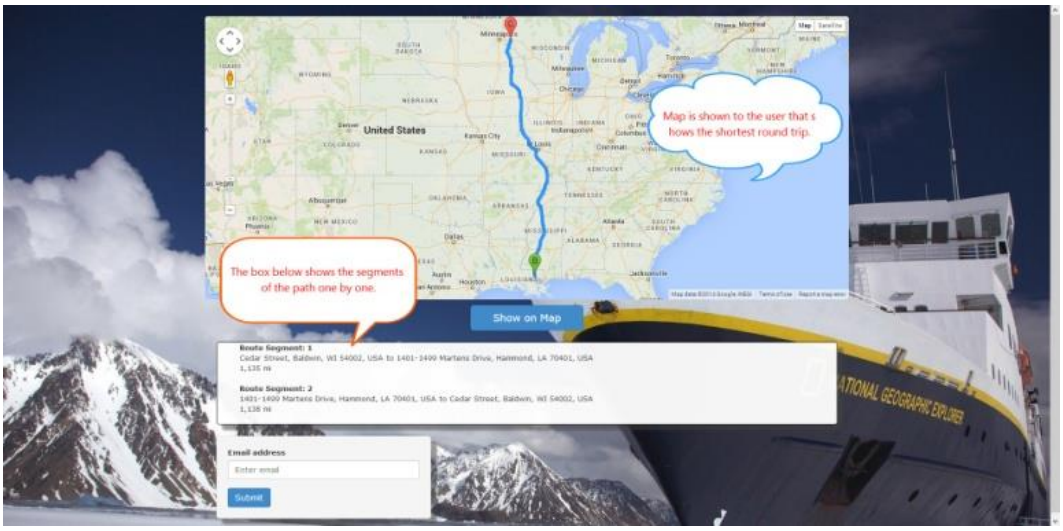


Fig. 4. A Mapped Solution for the Travelling Salesman Problem

Solving Travelling Salesman Problem," Proceeding of Instrumentation, Measurement, Computer, Communication and Control (IMCCC), Harbin, China 931-35, December, 2012.

[10] Konarski, M. and Zabierowski, W., "Using Google Maps API along with Technology .NET," Proceeding of Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), Lviv-Slavske, Ukraine, 180-82, February, 2010.

[11] Lenstra, J.K., Rinnooy Kan, A.H.G., "Some Simple Applications of the Travelling Salesman Problem," Operational Research Quarterly 26, 717-733, 1975.

[12] Lerner, R., "At the Forge: Ruby on Rails," Linux Journal, Issue#138, October 2005. DOA: http://www.linuxjournal.com/article/8433

[13] Paulson, L., "Developers Shift to Dynamic Programming Languages," Computer 40(2): 12-15, February, 2007.

[14] Perez, D., Prowley, E., Whitehouse, D., Samothrakis, S., Lucas, S., and Cowling, P. I., "The 2013 Multi-objective Physical Travelling Salesman Problem Competition," 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 2314-2321, July, 2014.

[15] Ratliff, H.D. and Rosenthal, A.S., "Order-Picking in a Rectangular Warehouse: A Solvable Case for the Traveling Salesman Problem," PDRC Report Series No. 81-10. Georgia Institute of Technology, Atlanta, Georgia, 1981.

[16] Reinelt, G., "The Traveling Salesman: Computational Solutions for TSP Applications," Springer-Verlag, Berlin, 1994.

[17] Toth, P. and Vigo, D., "The Vehicle Routing Problem," SIAM, Philadelphia, USA, 2001.

[18] Viswanathan, V., "Rapid Web Application Development: A Ruby on Rails Tutorial," IEEE Software, 25(6): 98-106, November, 2008.