Travel Itinerary Planning using Traveling Salesman Problem, K-Means Clustering, and Multithreading Approach

Muhammad Yasin, Sheila Nurul, Huda Septia Rani
Department of Informatics, Universitas Islam Indonesia
Yogyakarta, Indonesia
16523089@students.uii.ac.id, sheila@uii.ac.id, septia.rani@uii.ac.id

Abstract

This paper we proposed an algorithm for arranging travel itinerary using various approaches such as, traveling salesman problem with genetic algorithm, k-means clustering, and multithreading. The algorithm will be applied to develop a web based application which capable of making itinerary planning recommendation. This paper mainly focusing on how the proposed algorithm able to optimize the application in terms of computational proccessing time for the sake of efficiency. To make the itinerary recommendation, travelers must fill the input requirements such as number of days for vacation and list of destionations which they whish to visit. The destinations will first be clustered. Then find the TSP solution for the best route for each cluster. This TSP solution will be the itinerary recommendation.

Keywords— traveling salesman problem; Travel itinerary; k-means clustering; multithreading; genetic algorithm

I. INTRODUCTION

Travel itinerary can be determined as travel plans of tourists activity, which includes a detailed description of the attractions and activities of the destination visited, as well as the duration and specification of time and services provided [8]. The proccess of making travel plans becomes the next challengeing problem to be solved [1].

In real world situation some constrainsts of travel itinerary planning problem are unable to completely predifined [6]. However, we found that there are some constraints which can be defined such as, distance and time. By using these constraints we can provide optimal itinerary recommendation, assuming that the travelers have specified all the travel destinations they want to visit and how many days they will stay in the region.

The application needs to solve two sub-problems. These problems include grouping the travel destinations which travelers whish to visit for each day and create the travel route by arranging the destinations in a specific order so that it will have the least cost of travel distance.

Traveling Salesman Problem can be stated as if a salesman wishes to visit list of cities once for each city and then return to the home city, what is the route which has the least cost?[2]. In this study, a salesman can be interpreted as travelers and the visited list of cities are travel destinations.

In this study, we also apply paralel programing technique to maximize the application performance. The technique is simply running the application in multiple threads so that the minimum application proccessing time can be acquired. This technique is called multithreading.

II. PRELIMINARIES

A. Travelling Salesman Problem (TSP)

TSP is a well known problem in graph theory and combinatorics[1]. Mathematical problems related to TSP were discussed by the British mathematician Thomas Penyngton Kirkman and by the Irish mathematician Sir William Rowan Hamilton and in the 1800s [3]. In popular languages, traveling salesman problem can be described as a problem to find the least cost of travel distance in *n* cities, starting and ending in the same city and travel each city exactly once [4]. Many approaches and algorithms has been proposed to solve TSP such as Heuristic Algorithms, Mateheuristic Algorithms, Approximate Algorithms, and Exact Algorithms. The issue of TSP that processes no more than 20 cities can be solved optimally using the Exact method. The method which can provide reasonably high quality TSP solutions by processing large numbers of cities is the heuristic method [5].

B. Genetic Algorithm

Genetic algorithm as part of heuristic methode is an algorithm based on genetic proccess from within organism, that is the proccess of generation development within population, follows natural selection principle. This refer to evolution theory where the strongest one will survive. By imitating this evolution theory, genetic algorithm can be used to solve real world problems. This algorithm works with populations containing set of individuals or chromosomes. Each individual represents posible solution from existing problem. Also each individual has fitness value which will be used to find the best solution. For TSP, individual or chromosome can be interpreted as the route or set of cities. Each chromosome has fitness function where the shorter the distance the greater the fitness value. Based on the fitness

value, chromosome will be selected then do crossover and mutation to create the next generation. This proccess will be repeated until the solution is found.

In this study, we use genetic algorithm as a heuristic approach to solve traveling salesman problem in order to maximize the program efficiency. The proccess of genetic algorithm includes:

1. Create the population.

   In this case, the population can be defined as a collection of posible route. A single route is interpreted as the chromosome which contains list of travel destinations, and each destination is called genes. Each chromosome will have fitness value

2. Determine Fitness.

   After creating the population, we will determine the fitness for each chromosome. In traveling salesman problem fitness value can be interpreted as distance where the shorter the distance the greater the fitness value.

3. Select the parents that will be used to create next generations

   Parent is selected based on their fitness value. There are several method for how to select the parents. The most common are roulette wheel selection and tournament selection. For this study we use roullete wheel selection.

4. Crosover

   For TSP we use ordered crossover. In ordered crossover we select a subset of the first parent by random, then fill the rest of the route with the genes from the second parent.

5. Mutation

   Mutation can be defined as random changes to maintain the genetic diversity from within population, which usually applied in specified low probability. For TSP we use swap mutation. It means that the destinations or the genes within the chromosome will be swaped.

6. Repeat the proccess for the specified number of generations

C. K-Means

K-means is a simple clusterring method to cluster data set into k groups based on their smilarity. K-means assign each member of cluster by calculating the Euclidean distance. The goal of k-means is to minimize sum of square of distance between all data points with the center of each cluster. The proccess of k-means includes:

1. Specify the number of cluster k.
2. Initialize centroids by randomly selecting k data points from the dataset
3. Calculate the euclidean distance between all data points and all centroids, then assign each data point to the nearest cluster
4. Update centroids by calculating the means of all data points that belong to each cluster
5. Iterate until there are no more changes to the centroids

The advantage of using k-means is that it provides a simple, easy to use, and fast algorithm. For this study to avoid empty cluster or clusters having very few data points we use constrained k-means[7]. The algorithm is done by adding constraint to the clustering optimization problem requiring each cluster contains specified number of data points[7].

Constrained K-Means is an algorithm which basically consists of regular k-means that allows to execute k-means with defined minimum number of points belonging to given cluster. However, instead to just using clusters with smallest distance between data points to centers, the cluster assignment is done by solving Minimum Cost Flow problem. The data point are nodes with unitary supplies of flow, the centers are nodes with defined flow demands, one additional node contains demand balancing the global sum of supply and demands to zero. The data nodes are connected to center nodes with edge costs corresponding to Euclidean distances from data points to cluster centers. The center nodes are connected to the balance node with zero cost.

D. Multithreading

Multithreading is the ability of computer or central proccessing unit (CPU) to run application in multiple threads concurrently. In this study, we utilize multithreading to reduce the application proccessing time so that it can be more efficient. This technique is implemented in the developed application by simply create multiple threads with the same number of cluster, then run the program in each thread. More detailed explanation about how multithreading is implemented discused in section 3 of the paper.

III. METHODOLOGY

For this study we choose Yogyakarta Province as an example to demonstrate how the developed application with the proposed algorithm make travel itinerary recommendation.

There are some input requirements in the application which are number of days for vacation and the destinations the travelers want to visit. The destination can be filled with the address where the travelers want to visit.

The application consist of two main modules, which are clustering module using k-means and finding TSP solution

using genetic algorithm. Some proccesses that need to be done by the application are accessing the travel destinations that the travelers want to visit, making travel itinerary recomendaton, and displaying the itinerary result. The system model is presented in Figure 1.

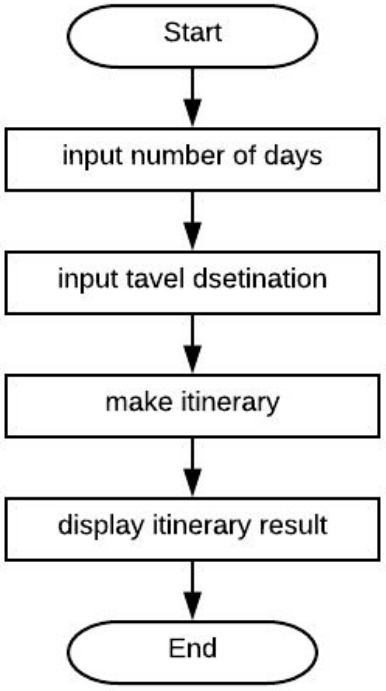

Figure 1. System Flowchart

Figure 1 explain the work flow of the application. The proccess begins by inputing the number of days and travel destinations. Then the system will make itinerary recommendation based on the inputed data. After that it will display the result. To be more specific figure 2 will explain the proccess of making the itinerary recommendation.

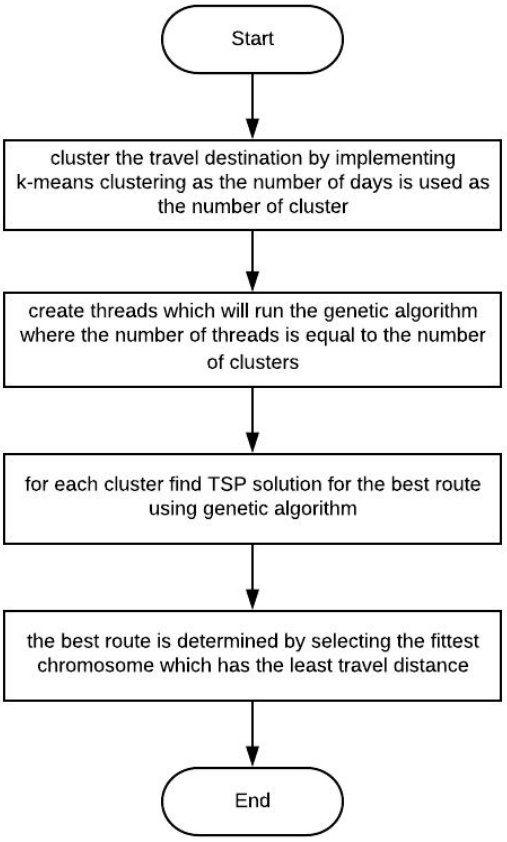

Figure 2. Detail proccess of making itinerary

Figure 2 explain the detail proccess of making itinerary. Start by clustering the travel destinations using k-means, where the number of days is used as k number of cluster. Next is creating threads where the number of threads is equal to the number of clusters. Each thread will run the genetic algorithm containing list of destinations from each cluster. After that, finding the TSP solution for the best route using genetic algorithm. Then select the fittest chromosome which has the least travel distance. The distance is calculated by using Google Maps Distance Matrix API. This chromosome can be ilustrated as the travel route which contains travel destinations. The fittest chromosome or route considered as the itinerary recommendation.

The proccess of making itinerary recommendation will also be executed with multithreading technique. The application will run in multiple threads. To implement this technique first we need to create the threads, to create threads we need to determine the number of threads. In this scenario, the number of threads is equal to the number of cluster. Then we execute the genetic algorithm function which will solve the TSP solution in each thread. For example if the number of cluster is 2 then we create two threads and execute the genetic algorithm in thread 1 and thread 2. Thread 1 run the algorithm which consists travel destinations from cluster 1 and thread 2 run the algorithm which consists travel destinations from

cluster 2. The example of the multithreading implementation can be seen in Figure 3.

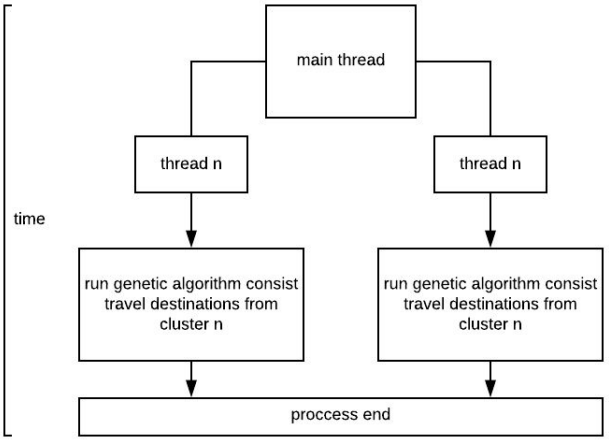

Figure 3. Multithreading Implementation

IV. RESULTS

The developed application consists two pages, first page is used to get the input from user and the second page is used for displaying the itinerary result. In the first page user can specify the number of days of vacation and desired tavel destinations. Since the application is on development phase it still performs in very simple interface. Figure 4 shows the example of implementation in the first page.

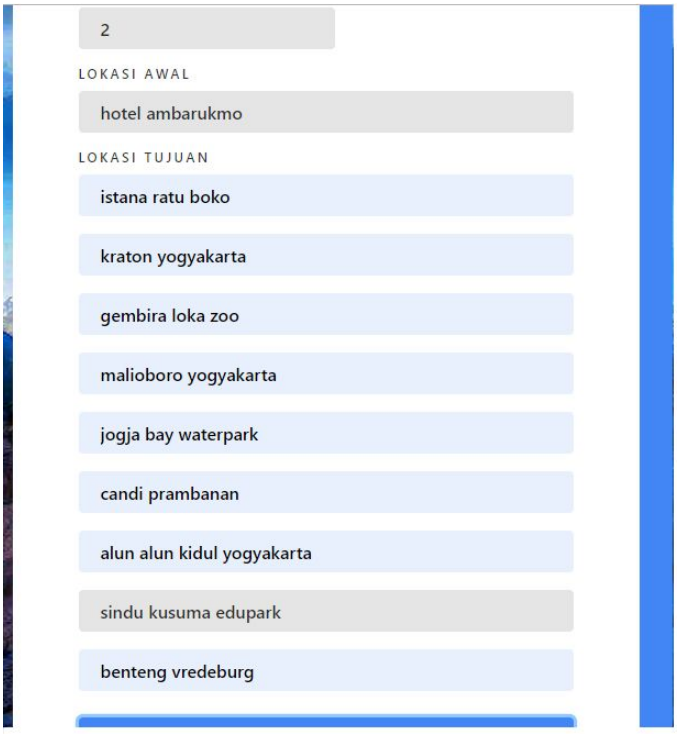

Figure 4. First Page

As we can see from figure 4, user inputs 2 days as the number of vacation days, and inputs hotel ambarukmo as the origin location, also istana ratu boko, kraton yogyakarta, gembira loka zoo, malioboro yogyakarta, jogja bay waterpark, candi prambanan, alun alun kidul yogyakarta, sindu kusuma edupark, and benteng vredeburg as travel destinations by filling the form field. After user submit the form the application will proccess the data using the proposed algorithm as described in section 3 of this paper. The itinerary result can be seen in Figure 5 and Figure 6.

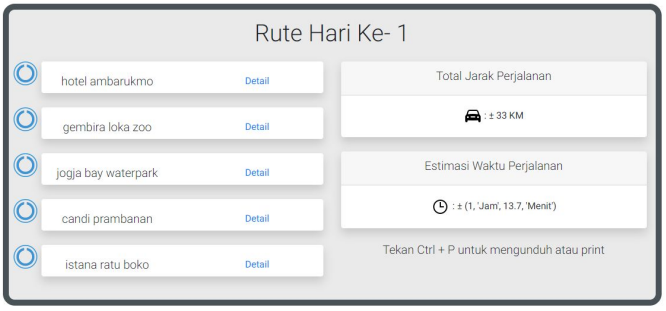

Figure 5. Route of day-1

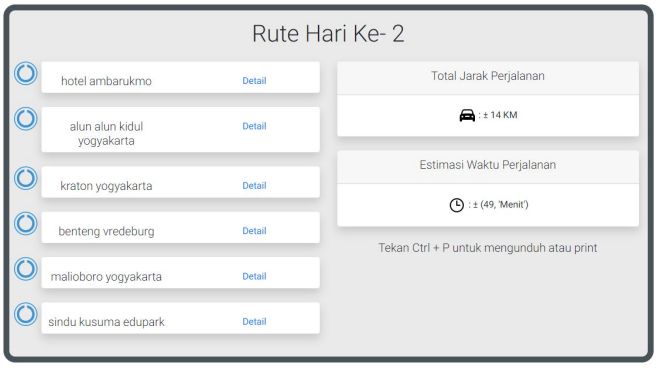

Figure 6. Route of day-2

On the first day user gets recommendation of 4 travel destinations to visit, from hotel ambarukmo we go to gembira loka zoo then to jogja bay waterpark, then we go to candi prambanan. while on the second day user gets 5 travel destinations, from hotel ambarukmo we go to alun alun kidul yogyakarta, then to kraton yogyakarta, then to benteng vredeburg, then to malioboro yogyakarta, last go to sindu kusuma edupark. By using constrained k-means with k number of cluster is 2 which it equal to the number of days, we get the silhouette score 0.49425. However, if we use regular k-means, the silhouette score is 0.63952. The total route distance in day-1 is 33 KM, and in day-2 is 14 KM. Since in this study, we use heuristic approach instead of exact calculation, the result may not be exactly the best route. However, it still provide the reasonable result.

Since we use genetic algorithm to solve TSP or in this case, finding the best route for itinerary recommendation, we run the algorithm with 50 population size and 50 generations. For small batch of TSP, using graph theory or brute force method may be more efficient. However, for large batch of TSP, this method is no longer considered as efficient, since the number of permutation will be very large. The implementation of genetic algorithm can be the reasonable method.

For experiment, we will analyze the application proccessing time. There are two type of experiments, first we run the program without multithreading and second we use multithreading. The multithreading is implemented by creating threads with the same number of cluster then run the specific function that solve TSP by finding the best route or in this case, the genetic algorithm function in each thread. For each scenario we run the program 5 times then calculate the mean value of the proccessing time. Even though the input data is the same, the proccessing time may vary due to the some factors such as, internet connection since we use Google Maps API, computer memory, and also proccessor speed. The result is presented in Table 1.

Table 1. Result without multithreading

No.	Days	Destinations	Mean of Proccessing Time (seconds)
1.	2	5	2.3736
2.	2	6	2.4424
3	2	7	2.6907
4.	2	8	2.7698
5.	2	10	3.1313
6.	2	12	3.3960
7.	2	15	3.6029
8.	2	20	4.0027
9.	3	6	2.4245
10.	3	7	2.5872
11.	3	8	2.8919
12.	3	10	3.3385
13.	3	12	3.5142
14.	3	15	3.6633
15	3	20	4.5537

Table 2. Result with multithreading

No.	Days	Destinations	Mean of Proccessing Time (seconds)
1.	2	5	2.0238
2.	2	6	2.1779
3	2	7	2.2835
4.	2	8	2.4687
5.	2	10	2.8557
6	2	12	2.8358
7.	2	15	3.1969
8.	2	20	3.9768
9.	3	6	2.5183
10.	3	7	2.5811
11.	3	8	2.6498
12.	3	10	3.0491
13.	3	12	3.3950
14.	3	15	3.6448
15.	3	20	4.2451

From this result we can see that with large number of destinations the program can run with reasonable amount of processing time. If we compare the result from table 1 to[1], our algorithm can be considered more efficient in terms of proccessing time. Although, for small number of destinations the proccessing time from[1] is slightly shorter, the diffrence is not very big. However, for the large number of destinations, at 12 destinations with 3 number of days for example, our algorithm is almost 53% faster. While in table 2, by implementing multithreading the processing time is sligthly increases. If we see from the result of table 2, multithreading implementation help minimize the program proccessing time. However, when the number of days and destinations increases, interestingly there are no much difference between table 1 and table 2. Even once it gets longer at 6 destinations with 3 number of days.

V. CONCLUSION AND FUTURE WORK

In this paper, we use k-means, traveling salesman problem, and multithreading approach to develop travel itinerary planning application. The application can help travellers to plan their vacation automatically. The use of genetic algorithm can singnificantly reduce the applications proccessing time. However, the algorithm can only be efficient with large batch of TSP, while in real world situation travellers usually don't visit many locations. On the other hand the implementation of multithreading slightly helps minimize the program processing time. As we can see from the result the difference is not very significant. For future work perhaps we should expand the multithreading approach to even paralel clustering.

VI. REFERENCES

[1] S. Rani, K. Kholidah and S. N. Huda, "A development of travel itinerary planning application using traveling salesman problem and k-means clustering approach," *Proceedings of the 2018 7th International Conference on Software and Computer Applications,* pp. 327-331, 2018.

[2] K. L. Hoffman, M. Padberg and G. Rinaldi, "Traveling Salesman Problem," *Encyclopedia of operations research and management science.*

[3] W. Cook, "History of the TSP," January 2007. [Online]. Available:

http://www.math.uwaterloo.ca/tsp/history/index.html. [Accessed 11 November 2019].

[4] C. Rego, D. Gamboa, F. Glover and C. Osterman, "Traveling salesman problem heuristics: Leading methods, implementations and latest advances," *European Journal of Operational Research, 211(3),* pp. 427-441, 2011.

[5] U. Nuriyev, O. Ugurlu and F. Nuriyeva, "Self-Organizing Iterative Algorithm for Travelling Salesman Problem," *IFAC-PapersOnLine, 51(30),* pp. 268-270, 2018.

[6] J.-S. Chen and F.-C. Hsu, "Interactive Genetic algorithms for a Travel Itenerary," *TSP 1,* p. 13, 2000.

[7] P. S. Bradley, K. P. Bennet and A. Demiriz, "Constrained K-means Clustering," *Microsoft Reasearch, Redmond,* vol. 20, 2000.

[8] A. A. da Silva, R. Morabito and V. Pureza, "Optimization approaches to support the planning and analysis of travel itineraries," *Expert Systems with Applications, 112,* pp. 321-320, 2018.