Application of Greedy and BFS to Determine Tour Route by Plane

Saskia Imani
Teknik Informatika, Sekolah Teknik Elektro dan Informatika
STEI, Institut Teknologi Bandung
Kota Bandung, Indonesia
13517142@std.stei.itb.ac.id

***Abstract*—Tour agencies are a common service used to plan tour routes according to clients' wishes. Usually, there are various factors taken into account, such as the clients' desired destinations, the length of the tour, and the clients' adaptability to the resulting schedule which may or may not fulfill ever criteria given. It is common sense to assume that the shorter amount of time spent traveling will mean more days available for the actual tour experience. In this paper, we use breadth-first search algorithm to plot plane travel routes and then use greedy algorithm to determine the best route available.**

Keywords—route, greedy, breadth-first search (BFS), graph, weight, optimum

I. Introduction

Travel agencies are a commonplace method to plan a vacation or tour. Those using the service of a travel agency is called a client, and are assigned to travel agents, which will accommodate all the desires of the client regarding their vacation or tour. There are several reasons why travel agencies are frequently.

- It saves time. Travel agencies takes over all the planning and preparations necessary for the client's travel. The client needs only to review the suggested plan.
- It does not cost more because travel agencies obtain special prices from the services they include in planning. So the client pays the same amount of price whether they book themselves, or book through an agent.
- Travel agencies have exclusive rates for services, which clients will probably not find on their own. They also have partnerships with various services that they recommend on the client's travel plan, which will considerabl result in ease when making special requests or responding during the rise of an emergency situation.
- Most travel agents are also experienced travelers themselves, and they oftentimes understand the details of traveling and various aspects involved better than clients.
- Travel agencies also provide assurance and security, because travel agents are usually required to summarize the details of services they use in planning the client's travel, including important terms and conditions clients may miss if they plan on their own.
- Travel agencies also provide the proper insurance required in the travel plan.
- Travel agents makes bookings for the client and maintains reservations until the day of the trip, so the clients do not need to do anything by themselves before the day of the trip.

One aspect of travel planning that a travel agent may consider the most complicated to handle is the client's request to travel several destinations within a number of days. In places such as Europe, it is common for tourists to want to visit different countries, exploiting the relatively close distance between countries and cities. However, the more destinations the client wants to include, the more difficult for travel agents to plan the best route. Taking into account the amount of budget required and shifting flight schedules, it becomes nearly impossible for a travel agent to plan without the use of automated algorithms.

There has been several programs and applications which realizes this problem and offers a solution, where the client will use the application instead of a travel agent. The application will then keep track of the days spent traveling, the budget required, the number of destinations, and more. But this solution is not much different from self-planning.

In this paper, we will explore the use of two well-known algorithms for problem solving known as the greedy algorithm and the breadth-first search. We will determine if the use of these algorithms to plan a tour route by plane will be feasible to implement in a travel agency enterprise.

II. Graph and the Breadth-First Search algorithm

A. Graph

A graph consists of a set of vertices which are connected by several edges. A vertex represents an object, and an edge represents a relationship between two objects. Based on the graph complexity, there are two types of graph:

- Simple graph, which has no more than one edge for each two vertices, or an edge connecting a vertex to itself; and
- Complex graph, which contains double edges for one or more two vertices.

Based on the directional orientation of the edges, there are two types of graph:

- Bidirectional graph, which does not have directional orientation on its edges; and
- Unidirectional graph, which has directional orientation on its edges.

For the problem we are going to solve, each vertex represents each destination (city). We use a complex and unidirectional graph, with every edge defining a flight between two cities.

B. Breadth-First Search

Breadth-first search (BFS) is a graph traversal algorithm which explores each vertex systematically to produce the solution to a problem. There are two approaches to using BFS to solve a problem:

- Using a static graph, meaning the graph has been constructed commencing BFS; or
- Using dynamic graph, meaning the graph is constructed as the BFS process progresses.

We use the latter in searching the solution to our problem, since each flight are only available for certain dates.

As its name implies, BFS determines which vertex to explore in breath rather than in depth. It is better explained in context to our problem using the algorithm below:

1. We start in vertex V, which is the client's current city.
2. We determine available flights from V to one of the client's desired destinations.
3. For each flight we apply a recursive search, meaning each flight destination becomes a new starting point.
4. For each recursive search, we note the cities that has already been traveled. Cities that has already been traveled cannot be traveled again in the sane vein of search.
5. If there are no more flights available from the current vertex (city), or there are no more flight destinations which are untraveled, we stop the search and calculate a score for the resulting route, which will be elaborated in the next part for greedy algorithm.

III. The Greedy Algorithm

The greedy algorithm is a popular method of solving problems which require an optimum solution, which may require a maximum or a minimum value. There are several elements to the greedy algorithm:

- Candidate set, C, which contains several candidates to be selected in a certain step of the algorithm;
- Solution set, S, which contains the result of choosing each candidate;
- Selection function, which determines the next candidate to select in the next step of the algorithm;
- Feasibility function, which determines whether the result of selecting a candidate is feasible as the solution to the problem; and
- Objective function, which determines the optimum solution

For our problem, our greedy consists of two factors, the route flexibility and the time flexibility. The route flexibility denotes a percentage in which the client is willing to adjust in terms of destination, if a resulting route of the BFS misses several destinations. The time flexibility route denotes a percentage in which the client is willing to adjust in terms of duration, if the resulting route of the BFS requires more days of travel than previously allocated. Greedy will automatically skip BFS-generated routes which does not qualify for both of the constraints above.

After determining feasible routes within constraints, greedy will calculate the final score, factoring the two accuracy percentages (respectively route- and time-based) of a BFS-generated route. Then, greedy will pick the optimum score as the solution to the problem.

IV. Implementation

A. Global Variables

In implementing BFS and greedy for our problem, there are several data components required.

- The amount of time allocated for travel,
- The client's current location / the starting point,
- A list of the client's desired destinations,
- The client's route and time flexibility, and
- A list of flights within the client's tour period involving cities from the list of client's desired destinations, including the duration of each flight.

These data components are stored each in a global variable, represented in the algorithmic notation below:

```
GLOBAL VARIABLES
 time: integer
 location: integer
```

```
destinations: array [1..20] of String
routeFlex: float
timeFlex: float
flight: type < from: integer
               to: integer
               length: integer >
flights: array [1..100] of flight
solution: type < route: array [1..20] of
                        integer
                 routeScore: float
                 timeScore: float >
solutions: array [1..1000] of solution
```

B. BFS Implementation

The implementation of the BFS algorithm elaborated a previous chapter is represented in the algorithmic notation below:

```
function FindIslands (location: integer,
traveled: array [1..20] of integer, flights:
array [1..100] of flight) → array [1..20] of
integer
// this function returns array of available
next destinations

procedure CalculateScore (input traveled:
array [1..20] of integer, input/output
routeScore, timeScore: float)
// this function calculates the score of route

procedure BFS (input traveled: array [1..20]
of integer, input location: integer,
input/output solutions: array [1..1000] of
solution, input flights: array [1..100] of
flight)

 LOCAL VARIABLES
  newTraveled: array [1..20] of integer
  tempArr: array [1..20] of integer
  newSol: solution
  routeScore: float
  timeScore: float

 ALGORITHM
  newTraveled ← traveled.copy()
  newTraveled.add(location)
  tempArr ← FindDestinations(location,
            traveled, flights)
  if (tempArr.length = 0)
   CalculateScore(traveled, routeScore,
                  timeScore)
   if ((1-routeScore) <= routeFlex) and ((1
      timeScore) <= timeFlex)
    newSol.route ← traveled
    newSol.routeScore ← routeScore
    newSol.timeScore ← timeScore
    solutions[solutions.length] ← newSol
   else
    i traversal [0..tempArr.length]
     BFS(newTraveled, tempArr[i], solutions,
     flights)
```

C. Greedy Implementation

The implementation of the greedy algorithm elaborated in a previous chapter is presented by the algorithmic notation below:

```
function FindMaxScore(solutions: array
[1..1000] of solution) → array [1..20] of
integer
 LOCAL VARIABLES
  totalScore: float
  maxScore: float
  maxIdx: integer

 ALGORITHM
  totalScore ← (solution[0].routeScore +
               solution[0].timeScore) / 2
  maxScore ← totalScore
  maxIdx ← 0
  i traversal [1..solutions.length]
   totalScore ← (solution[i].routeScore +
                 solution[i].timeScore) / 2
   if (totalScore > maxScore)
```

```
    maxScore ← totalScore
    maxIdx ← i
  → solutions[maxIdx]
```

D. Main Program

The algorithmic notation below serves a description of how the program as a whole will look like:

GLOBAL VARIABLES

```
// global variables required as defined in part A of this chapter, plus the status variable below
traveled: array [1..20] of integer
ultSolution: solution
```

FUNCTIONS

```
function FindDestinations (location: integer, traveled: array [1..20] of integer, flights: array [1..100] of flight) → array [1..20] of integer
// this function returns array of available next destinations

function FindMaxScore(solutions: array [1..1000] of solution) → array [1..20] of integer
// this function calculates the scores of all solutions and returns the optimum route as defined in part C of this chapter
```

PROCEDURES

```
procedure CalculateScore (input traveled: array [1..20] of integer, input/output routeScore, timeScore: float)
// this function calculates the score of route

procedure BFS (input traveled: array [1..20] of integer, input location: integer, input/output solutions: array [1..1000] of solution, input flights: array [1..100] of flight)
// this function recursively searches for a solution using BFS and adds feasible solutions to the solution array as defined in part B of this chapter
```

ALGORITHM

```
// by any preferred method, initialize the following variables
time ← time
location ← location
destinations ← destinations
routeFlex ← routeFlex
timeFlex ← timeFlex
flights ← flights

// initialize status variable with null values
traveled ← []
ultSolution ← ultSolution

// commence BFS
BFS(traveled, location, solutions, flights)
// commence Greedy
ultSolution ← FindMaxScore(solutions)
```

V. Algorithm Complexity

The criteria of a good algorithm can be viewed from the following aspects:

- Completeness, which is whether a solution is guaranteed to be reached if it exists;
- Optimality, which is whether the method ensures an optimal solutions;
- Time complexity, which is the time required to reach the solution; and
- Space complexity, which is the memory required to implement the algorithm.

Whereas the time and space complexity can be measured using the following terms:

- *b*: (branching factor) maximum branching from a certain vertex
- *d*: (depth) depth of the optimum solution
- *m*: maximum depth of the status space (may be indefinite)

For required values to be determined in each case, we use these variables:

- *n*: number of cities

- *f*: number of flights

A. Completeness

The completeness of the algorithm above depends on the amount of time allocated by the client for travel. In most cases, a solution is likely to be reached. However, due to the use of the greedy algorithm, we have eliminated the possibility of visiting the same destination twice, assuming that it will only waste more time. In reality, visiting the same destination twice may serve as the shortest amount of time from an untraveled destination to another in a certain case. On the contrary, allowing the visitation of the same destination twice will defeat the purpose of using BFS, since the resulting route will be infinite.

B. Optimality

Similar to the argument made in the previous section A for the completeness of the algorithm, the optimality of the algorithm also cannot be ensured. In most cases, an optimum solution is likely to be reached. However, due to eliminating the possibility of visiting the same destination twice may very well result in eliminating the optimum solution for a certain case.

C. Time Complexity

The time complexity of the algorithm can be observed from the functions and procedures used. We ignore the complexity of initializing the global and status variables.

- The BFS procedure produces a complexity equal to the permutation of destination cities, so

$$\boldsymbol{T(n) = O(n!)}$$

- The function FindDestinations traverses through the list of flights and compares the flight's destinations to the array of traveled cities. Assuming the best case scenario, no flights are available from the current destination, which results in

$$T(f) = O(f)$$

Otherwise, in the worst case scenario, all flights' destination is the last element of the traveled cities, which (assuming the traveled cities number *t*) results in

$$T(f) = f + fn = O(2f)$$

Thus, the average time complexity for this procedure is

$$\boldsymbol{T(f) = f + f + fn = O(1.5f)}$$

- The CalculateScore procedure detects the number of cities traveled and the amount of time taken to travel. Then it produces the percentage of similarity with the time allocation given by the client and the maximum number of cities. In the worst case scenario, the resulting route ends without a single flight. So

$$T(n) = 1$$

Whereas the worst case scenario is all cities have been traveled. The variable f' denotes the comparisons made to the list of flights to determine the length of travel between one city to another.

$$T(n) = (n-1) + (n-1)f' = O(2n)$$

Thus, the average time complexity of this procedure is

$$\boldsymbol{T(n) = (1 + (n-1) + (n-1)f') / 2 = O(n)}$$

- The greedy algorithm traverses the list of solutions once and determines the best solution. In the best case scenario, there is no solution, so

$$T(n) = 0$$

Whereas in the worst case scenario the solutions consist of combinations of destinations. This results in

$$T(n) = n^n = O(n^n)$$

Thus, the average time complexity of this algorithm is

$$\boldsymbol{T(n) = n^n / 2 = O(0.5n^n)}$$

By calculating these different components, we get the time complexity

$$\boldsymbol{T(n, f) = n! + 1.5f + n + 0.5n^n = O(n^n, f)}$$

D. Space Complexity

The space complexity of the algorithm can be observed from the variables used previously. We ignore the space used by variables outside of those required by the algorithms. The total space complexity is the total size allocated to implement the algorithm, which is

$$\boldsymbol{S(n, f) = n + f + n^n}$$

VI. Conclusion

The usage of greedy and the BFS algorithm to plan the perfect tour route is theoretically guaranteed to work, since the use of BFS ensures that a solution is reachable if it exists. The optimality of this algorithm, however, relies on the use case. In most cases, an optimum solution is likely acquired. In other cases, however, characteristics of the BFS may prove to prevent the algorithm from finding the optimum solution. In

addition, the amount of resource required to implement the algorithm poses another problem. The complexity is exponential, and as such, there needs to be a limitation of how many destinations the client may request. Otherwise, the resource required are likely to be unaffordable by many travel agency enterprises.

VII. Acknowledgements

The author would like to express her gratitude to God Almighty, for only because of His amazing grace the author is able to find the inspiration to begin this paper, and the ability to finish it. The author is also thankful to to Dr. Ir. Rinaldi Munir, M.T., as the beloved lecturer of IF2120 Discrete Mathematics of Class 01, for his dedication and enthusiasm, and unique little quirks in lecturing his students for this semester. The author also wishes to express gratitude to her parents, her sister and brother, and her friends for all their support and help during the writing process of this paper. The author also remembers the teamwork her class has shown in making sure that every student's paper will be unique and considerably different from each other's. The efforts of the people involved in making the database of paper titles is highly appreciated.

References

The following are references used in making the introduction and definition of terms in this paper.

[1] Black, Sally. 2017. How a Travel Agent Works. Accessed at *https://www.huffpost.com/entry/how-a-travel-agent-works_n_7903072* on Friday, April 26 2019

[2] Munir, Rinaldi. 2017. Diktat Strategi Algoritma. Bandung: Insitut Teknologi Bandung

[3] RoutePerfect: explore the world your way. *Tour Planner*. Accessed at *https://www.routeperfect.com/trip-planner* on Friday, April 26 2019

[4] MapQuest. Multi-Stop Route Planning and Optimization Tools. Accessed at *https://www.mapquest.com/routeplanneri* on Friday, April 26 2019

PERNYATAAN

I hereby state that the paper I have written is my own writing, not a copy, or a translation of another person's work, and is not a result of plagiarism.

Bandung, 29 April 2012

Saskia Imani, 13517142