A Comparative Study on Informed and Uninformed Search for Intelligent Travel Planning in Borneo Island

Raymond Chiong, Jofry Hadi Sutanto and Wendy Japutra Jap
School of Computing & Design
Swinburne University of Technology (Sarawak Campus)
State Complex, 93576 Kuching, Sarawak, Malaysia
rchiong@swinburne.edu.my, jofrisutanto@gmail.com, my.keyblade@gmail.com

Abstract

In this paper, we examine different search algorithms from artificial intelligence that can be used for solving the shortest path problem. We investigate the use of three informed search algorithms and three uninformed search algorithms for intelligent travel planning based on some major cities and towns in Borneo Island. We also present an improved dijkstra's algorithm for this task. We show that while best-first and A are effective at finding short useful paths, hill-climbing and most of the uninformed search algorithms are much less useful. The improved dijkstra's algorithm is the best in terms of accuracy and shortness of path found.*

1. Introduction

The reliance on road transports nowadays has made travel planning by land a vital issue. Due to increased traffic and complicated modern road networks, finding a good route to travel from one place to another is no longer trivial. While many artificial intelligence methods have been used effectively for finding the shortest path between two locations, limited research was done in directly adapting these algorithms for practical route finding [1]. As such, algorithms that may be effective from a theoretical point of view might not work well in a practical context.

In this paper, we present a study to examine six informed and uninformed search algorithms from the field of artificial intelligence. Three informed search algorithms that have been selected are hill-climbing, best-first search and A*, whereas the three uninformed search are depth-first search, breadth-first search and random-walk. We draw comparison on the performance of these six best known algorithms by applying them to the Borneo road network. In addition to the six algorithms, a custom search algorithm based on dijkstra's will also be introduced.

The rest of this paper is organised as follows: the next section briefly introduces the background of the problem. Following which, we present the methodology we use for this study by describing our implementation on the various kinds of algorithms we are dealing with. Finally, we discuss our experimental results and draw conclusion based on the findings.

2. Background

Shortest path problem has been well-studied for decades. It concerns finding a path between two nodes or vertices in a weighted network such that the sum of the weights (i.e. distances) of its constituent edges is minimised. Over the years, many algorithms [2, 3, 4, 5, 6, 7, 8] were proposed to solve the problem. The most direct solution would be to test out every possible path and select the shortest one. However, this works well only for a low number of cities, and it becomes impractical when the number of cities is in the range of hundreds or even thousands.

Generally, algorithms for finding shortest path perform searches by traversing a search tree from the root node to the destination node. Dijkstra's algorithm, invented by Dutch computer scientist Edsger Dijkstra [9], has been one of the most effective algorithms for the problem as many other good solutions are indeed a variation of it. Besides dijkstra's, another good algorithm can be found in A* [5]. A* is a heuristic search based on the use of a cost estimating function. Most of the time it produces not only an optimal solution, but also keeps a lower computational time and complexity as compared to dijkstra's.

978-1-4244-2328-6/08/$25.00

The past results in applying these algorithms to path finding problem were promising. In a practical context, though, they have severe drawbacks. The time taken for these algorithms to work on huge networks is far too lengthy, and yet not necessarily obtaining the best solution. Therefore, there is a need for improvement.

With this in mind, we use the road network of Borneo Island to examine the practicality of these algorithms. Borneo, the third largest island in the world, is divided between Indonesia, Malaysia and Brunei administratively. As the island is large, most of the cities and towns are connected through some main roads along the coastal. This means that the connectivity between two smaller towns is minimal. Because of this, we are expecting some algorithms that work well theoretically to perform badly in our study.

3. Methodology

In this section, we will present our methodology in implementing all the search algorithms we use for this comparative study. We will discuss how each of the algorithms works fundamentally, and the way they are being applied to our problem.

3.1. Depth-First Search

Depth-first search is an uninformed search technique for graph traversal. It traverses a tree graph by starting at the root node, and always chooses to go deeper into the graph. When it comes to the bottom of a finite graph, backtracking is performed to allow it to return to the most recent node it has not finished exploring. Figure 1 depicts the way depth-first search looks for a solution.

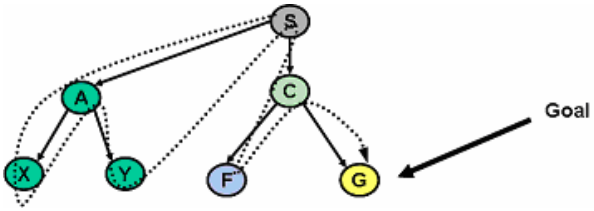

Figure 1. The way depth-first search looks for solution.

The following is the pseudocode of depth-first search we have implemented.

```
function DFS()
    create a stack list: open
    create a list: track
    insert the origin town to the open list
    loop as long as the open list is not empty
        var current = the end of the open
        remove the town from the end of open
        insert current into track
        if current is equal to destination
                construct the path
                calculate the distance
                return
        else
                generate children of current town
                store children into the open
                if there is no children found
                                do backtracking
```

3.2. Breadth-First Search

Breadth-first search is another important type of uninformed search. In this search, a queue is normally used to enable level-by-level traversal. From the root node, breadth-first search moves horizontally. It explores all the neighbouring nodes until the goal node is found. Figure 2 shows how a breadth-first search works.

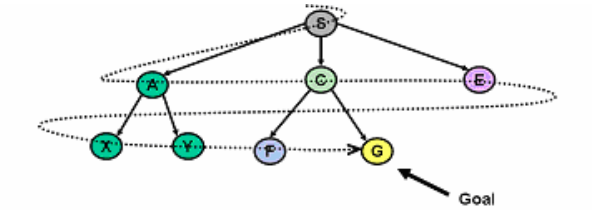

Figure 2. The way breadth-first search looks for solution.

Our implementation on breadth-first search is as follows:

```
function BFS()
    create a queue list: open
    create a list: close
    insert the origin town to the open list
    loop as long as the open list is not empty
        var current = beginning of the open
        remove the town from beginning of open
        if current is equal to destination
                construct the path
                display the result
                return
        else
                generate children of current town
                store children into the open
```

3.3. Random-Walk

Random-walk is a non-deterministic search representing a mixture of depth-first and breadth-first search. Starting from the root node, it selects one of the successor nodes randomly. Therefore, this search may traverse the tree in either a vertical manner or a horizontal manner. Facilitated by backtracking, it stops exploring when the goal node has been successfully located. The following is the pseudocode for random-walk.

```
function randomWalk()
    create list: open, track
    insert the origin to the open list
    loop as long as the open list is not empty
        var current = the end of the open
```

978-1-4244-2328-6/08/$25.00 © 2008 IEEE

```
        remove the town from the end of open
        insert current to the track
        if current is equal to destination
                construct the path
                display the result
                return
        else
                generate children of current town
                if there is no children generated
                        apply backtracking
                store the children randomly into
                the open
```

3.4. Hill-Climbing Search

Hill-climbing is a simple informed search that continuously traverses forward based on an evaluation function. The idea is to always head towards the best successor node with the least evaluation function cost, while the rest (unselected nodes) will be pruned. When it comes to a situation whereby no better nodes are available, it terminates.

The evaluation function we use for hill-climbing is *f(Node) = d(Node)*, where *d(Node)* is the straight line distance from the current node to the goal node. The pseudocode is as follows:

```
function hillClimbSearch()
    create a list: track
    initialise current as the origin
    loop while condition is true
        generate children of current
        loop through the children
                if one of the children is equal to
                destination
                        construct the path
                        display result
                        return
                compute evaluation function for the
                children
                take out a town with the least
                evaluation cost as current
```

3.5. Best-First Search

Best-first search is yet another type of informed search. It works in a similar way as hill-climbing, including the use of its evaluation function. Unlike hill-climbing, best-first search allows backtracking. It always maintains a list of nodes that have to be further explored, and expands on those nodes to find the best solution. The pseudocode for best-first search is as follows:

```
function bestFirstSearch()
    create list: open, close
    insert origin into the open
    loop as long as open is not empty
        initialise current = the beginning of open
        if current is equal to destination
                construct the path
                display result
                return
        generate children of current
        store the children to open
        compute evaluation function for the children
        store the evaluation cost of each child to
        an array
        insert the beginning of open to close
        erase the beginning of open
    sort the open list in ascending order based on
    the evaluation cost
```

3.6. A* Search

A* is indeed an extension of best-first search. It uses a more intensive evaluation function to choose the next node. The evaluation function consists of:

i. cost from the start node to the current node
ii. estimated cost from the current node to the goal node

With these criteria, A* will choose a node with the minimum total cost, which is likely to be optimal.

The following is the pseudocode of how A* is being implemented. It looks identical to best-first search. The only difference being the evaluation function. While best-first search considers only how close it has moved towards the goal node, A* also takes the cost of travelling from the start node to the current node into consideration.

```
function aStarSearch()
    create list: open, close
    insert origin into the open
    loop as long as open is not empty
        initialise current = the beginning of open
        if current is equal to destination
                construct the path
                display result
                return
        generate children of current
        store the children to open
        apply evaluation function to the children
        store the evaluation cost of each child to
        an array
        insert the beginning of open to close
        erase the beginning of open
    sort the open list in ascending order based on
    the evaluation cost
```

3.7. Custom Search

Our custom search is derived from dijkstra's algorithm [9]. This search will start traversing from the root of the tree. It loops through all the nodes in the graph tree by keeping the shortest distance so far from root to each node. It will eventually find the shortest distance from root to all nodes in the graph. In order to lower the computational cost taken by the original dijkstra's algorithm, our custom search does not look for the shortest distance to all nodes. Instead, termination criteria are engaged when the shortest distance to the goal node has been found. The pseudocode for our custom search is as follows:

```
function customSearch()
    var current = origin
    create array d to store the distance from origin
    to every other towns
    initialise array d to -1 (as undefined)
    create array ut to store the unmarked towns
    create array of lists: track to keep record of
    paths from origin to each of other towns
    loop as long as current is not the same as
    destination
        compute minimum distance from origin to the
        unmarked towns (which are directly connected
        with current) through current
```

978-1-4244-2328-6/08/$25.00 © 2008 IEEE

```
        put current as the town with minimum
        computed distance
    display the shortest path and distance from
    origin to the destination
```

4. Experimental Results

In this section, we report on the experimental results of our study. We have specifically selected 100 major cities and towns throughout Borneo Island to test the search algorithms we have implemented. The road connections between cities and towns are assumed to be symmetric, that is, for any 2 towns A and B, the distance from A to B is the same as that from B to A. In this case we will get exactly the same tour length if the order in which the two towns are visited is reserved. The locations of the cities and towns are recorded in longitudes *lon* and latitudes *lat*, and the distance is calculated with the "haversine" formula: *distance = r * acos[sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(lon2 - lon1)]*, where *lon* and *lat* are in radians, and the earth's radius *r* = 6378.7 km.

Table 1, 2, 3 and 4 below show the results gathered over 200 generations with 8 towns, 15 towns, 75 towns and 100 towns respectively.

Table 1. Experimental result with 8 cities/towns.

Algorithm	Average Accuracy (%)	Average Steps Taken
Depth-first	56.2354 %	4
Random walk	59.7912 %	5
Breadth-first	89.4324 %	5
Hill-climbing	99.1334 %	2
Best-first	99.8935 %	5
A*	100 %	3
Custom	100 %	4

Table 2. Experimental result with 15 cities/towns.

Algorithm	Average Accuracy (%)	Average Steps Taken
Depth-first	40.2077 %	9
Random walk	29.9601 %	9
Breadth-first	90.4794 %	11
Hill-climbing	92.3421 %	2
Best-first	98.8404 %	8
A*	99.7795 %	3
Custom	100 %	7

Table 3. Experimental result with 75 cities/towns.

Algorithm	Average Accuracy (%)	Average Steps Taken
Depth-first	76.8351 %	79
Random walk	79.8456 %	78
Breadth-first	98.1841 %	42
Hill-climbing	43.6451 %	7
Best-first	99.6606 %	37
A*	99.5693 %	25
Custom	100 %	36

Table 4. Experimental result with 100 cities/towns.

Algorithm	Average Accuracy (%)	Average Steps Taken
Depth-first	70.9024 %	1268
Random walk	68.6422 %	832
Breadth-first	95.9254 %	59
Hill-climbing	32.9008 %	7
Best-first	98.8035 %	48
A*	98.9812 %	33
Custom	100 %	47

From our experiments, we noticed that depth-first search, breadth-first search and random-walk were always able to find a solution, if there was one. However, depth-first search and random-walk did not perform well throughout the experiments as the solutions they produced were sub-optimal, i.e. not the shortest path. Among the uninformed search algorithms, only breadth-first search lived up to par with average accuracy of around 90% when 8-15 cities/towns were considered and close to 95-98% when the number of cities/towns were larger (e.g. 75-100). As breadth-first search applied exhaustive search, there was a slightly bigger overhead in searching for solution in terms of steps taken.

Hill-climbing tried to overcome the exhaustive search of depth-first search by using heuristics. Based on an evaluation function, hill-climbing managed to positively cut off the search overhead, which was reflected by fewer steps taken in completing a search in our experiments. As we can see from the results shown in the tables, it worked well with small number of cities and towns where high connections are available. When the number of cities and towns being considered was large, hill-climbing often got stuck at local optima. This was evidenced by the low average accuracy it achieved with data size of 75 to 100 cities and towns.

The reason hill-climbing did not work well when the number of cities and towns became larger was very much due to the road connections. As aforementioned, Borneo Island is large, and most of the cities and towns are connected only through some main roads along the coastal, resulting in low connectivity between different locations. In such case, hill-climbing tends to fail frequently because it terminates when there is no better way to move forward. When the connections are low, the possibility of getting stuck increases.

978-1-4244-2328-6/08/$25.00 © 2008 IEEE

The performance of best-first search and A* were satisfying. By extending the performance of breadth-first search, best-first search maintained a good average accuracy between 98-99%. A* further improved the performance of best-first search by enhancing the evaluation function, thus was always able to keep the average accuracy up to 99%.

Based on the experimental results, we see that informed search algorithms are generally doing very well apart from hill-climbing. However, we observe that the performance of both best-first search and A* degraded slightly as the number of cities and towns grew bigger. Inspired by dijkstra's algorithm, the custom search we presented worked pretty well by maintaining 100% accuracy throughout the experiments. Even though the search overhead was slightly worse than A*, it has been much better as compared to the typical dijkstra's algorithm. This is because in our custom search, we stop the search as soon as the algorithm has found the minimum distance to the destination, unlike the original dijkstra's where the distances from origin to all locations are calculated. As a result, our improved dijkstra's algorithm is able to achieve high accuracy towards optimum solution, and at the same time maintain an acceptable search overhead.

5. Conclusion

In this paper, we have examined the performance of three informed search and three uninformed search algorithms for intelligent travel planning in Borneo Island. We have also presented a custom search in an improved dijkstra's algorithm.

From our studies, we showed that depth-first search and random-walk are obviously not suitable for this problem. Breadth-first search produces near-optimal solution but the search overhead is slightly higher.

Hill-climbing has been quite effective for small data size but when the number of cities and towns grows large, its accuracy decreases drastically. Meanwhile, the other two heuristic algorithms, best-first search and A*, have been working very well in solving the problem.

The improved dijkstra's algorithm also works very well for this task. Indeed, it is the only algorithm that is able to maintain an optimal solution in every attempt, even though the steps required for it to find the solution are slightly more than A*.

This study demonstrated that simple heuristic algorithms such as best-first search, A* and our improved dijkstra's algorithm are feasible solutions to travel planning. While some fancy meta-heuristic approaches like Genetic Algorithms, Ant Colony Optimisation, Tabu Search and Simulated Annealing have been used extensively for this kind of problems in recent years, simple heuristic searches can be far simpler and involve less computation.

6. References

[1] B. Liu, "Intelligent route finding: Combining knowledge, cases and an efficient search algorithm," in *Proc. 12th European Conference on Artificial Intelligence (ECAI 1996)*, Budapest, Hungary, 1996, pp. 380-384.

[2] G. Gallo, and S. Pallottino, "Shortest path methods: a unified approach," *Mathematical Programming Study*, vol. 26, 1986, pp. 38-64.

[3] A. K. Goel, K. S. Ali, M. W. Donnellan, A. G. de Silva Garza, and T. J. Callantine, "Multistrategy adaptive path planning," *IEEE Expert*, vol. 9, no. 6, 1994, pp. 57-64.

[4] R. V. Helgason, J. L. Kennington, and B. D. Stewart, "The one-to-one shortest-path problems: an empirical analysis with the two-tree Dijkstra algorithm," *Computational Optimization and Applications*, vol. 1, 1993, pp. 45-75.

[5] J. Pearl, *Heuristics*, Addison-Wesley, Reading, MA, 1988.

[6] H. K. Chen, and G. Feng, "Heuristics for the stochastic/ dynamic user-optimal route choice problem," *European Journal of Operational Research*, vol. 126, 2000, pp. 13-30.

[7] F. Guerriero, and R. Musmanno, "Label correcting methods to solve multicriteria shortest path problems," *Journal of Optimization Theory and Applications*, vol. 111, no. 3, 2001, pp. 589-613.

[8] G. Khachaturov, "An approach to trip- and route-planning problems," *Cybernetics and Systems: An International Journal*, vol. 33, 2002, pp. 43-67.

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

978-1-4244-2328-6/08/$25.00 © 2008 IEEE