Shortest path problem on a grid network with unordered intermediate points

Veekeong Saw, Amirah Rahman, and Wen Eng Ong
School of Mathematical Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
E-mail: kentvk_90@hotmail.com, amirahr@usm.my, weneng@usm.my

Abstract. We consider a shortest path problem with single cost factor on a grid network with unordered intermediate points. A two stage heuristic algorithm is proposed to find a feasible solution path within a reasonable amount of time. To evaluate the performance of the proposed algorithm, computational experiments are performed on grid maps of varying size and number of intermediate points. Preliminary results for the problem are reported. Numerical comparisons against brute forcing show that the proposed algorithm consistently yields solutions that are within 10% of the optimal solution and uses significantly less computation time.

1. Introduction

Shortest path problem (SPP) is a classical optimization problem in network theory. The objective of SPP is to find a path from a starting point to a destination point in a network, such that the resource consumption is minimized. There are a handful of well-known algorithms used to find optimal solution for SPP, such as Dijkstra algorithm, A* algorithm, Bellman-Ford algorithm and Floyd-Warshall algorithm, depends on the nature of SPP itself ([4, 7, 1]). Different variations of SPP has been applied in various fields such as polar image segmentation, texture analysis, vehicle routing and military operations ([2, 5, 8]). SPP can be represented in various metric representations such as grid network, visibility graphs and Voronoi diagram. The metric representation that is widely used in path planning is grid network. For example, in military operations, the path planning for unmanned combat vehicles was represented in grid map ([3, 6]).

In this paper, we consider the SPP with multiple unordered intermediate points (SPP-UIP) on a grid network, where the objective is to find the minimum cost Hamilton path with unique starting and ending points. Note that the order of the intermediate points are not given in advance and it is to be determined during the process of searching for optimal path. To date, there has been no apparent research on shortest path problem that involves grid maps and unordered intermediate points as in our research. A two phase heuristic is proposed as a solution method for the SPP-UIP.

2. Problem Description

In this work, the grid network which is represented in an $N \times N$ grid, was modeled as a graph $G = (V, E)$, where $V$ is the set of vertices, which represents all cells in the grid and $E$ is the set of edges which represents the link between two adjacent cells in the grid. Associated with each vertex is the cost value. We assume that the grid is 8-connected, meaning that any cell in the
grid can move to any one of the eight adjacent cells (Figure 1), except the cells on the border of the grid. A path between two vertices can be represented by a sequence of cells. Figure 2 shows an example of a shortest path between two cell, (1,1) and (14,8), on a 15 × 15 grid. The total cost of a path is calculated by summing up the cost of all the traversed vertices.

Figure 1. Accessible directions from a cell in different positions. ([6])

![Accessible directions from a cell in different positions.](image)

Figure 2. Shortest path from (1,1) to (14,8) with a total cost of 24.

![Shortest path from (1,1) to (14,8) with a total cost of 24.](image)

A path between any two vertices is known as a partial path. A path that links the starting point to the destination point, passing through all intermediate points once is known as a complete path. A complete path is given as a concatenation of partial paths. The total cost of a path is known as the cumulative cost of the path.

Before proceeding to the mathematical formulation of SPP-UIP, a list of notation used throughout this paper is shown as follows.
The integer linear programming formulation of SPP-UIP is as follows.

\[
\text{Minimize } \sum_{i \in V} x_i c(i) \tag{1}
\]

\[
\text{s.t.}
\]

\[
\sum_{j \in \text{Suc}(i)} x_j = 1, \quad \forall i \in \Lambda \setminus \{t\} \tag{2}
\]

\[
\sum_{j \in \text{Pre}(i)} x_j = 1, \quad \forall i \in \Lambda \setminus \{s\} \tag{3}
\]

\[
\sum_{j \in \text{Suc}(i)} x_j - \sum_{j \in \text{Pre}(i)} x_j = 0, \quad \forall i \in V \setminus \{s,t\} \tag{4}
\]

\[
\sum_{i \in S} \sum_{j \in \text{Suc}(i)} x_j \leq |U| - 1, \quad \forall U \subset V, U \neq \emptyset \tag{5}
\]

\[
x_i \in \{0,1\}, \quad \forall i \in V
\]

Equation (1) represents the objective function to minimize the total cost of the complete path. Equation (2) and (3) ensure that there can only be one arrival and one departure from any point except for the starting point (no arrival) and destination point (no departure) respectively. Equation (4) is the flow conservation constraint. Inequality (5) prevents the solution path from forming subtours.

### 3. Two Phase Heuristic

We present a two phase heuristic to solve the SPP-UIP. In Phase 1, we use the Dijkstra algorithm to generate the shortest partial path between every pair of points in \( \Lambda \), and represent it in incidence matrix form. Then, in Phase 2, we choose the best combination of partial paths that form the complete path with minimum cost. There are many ways to search for such
combinations. In this study, we compare the repetitive nearest neighbor algorithm (RNNA) with the brute force method (BF). A detailed description for each phase is presented as follows.

3.1. Phase 1: Generate the complete graph

In Phase 1, we compute the shortest partial path for all pairs points in \( \Lambda \) by applying Dijkstra algorithm \((n+2)(n+1)/2\) times. Note that in typical SPP, the cost value is associated with edges of the graph and the cumulated path cost is calculated by adding all the cost of path edge associated to the complete path. However, in SPP-UIP the cost value is associated with vertices of the graph. Hence, while applying the Dijkstra algorithm, the path cost is calculated by adding the cost associated to the vertices instead of edges. The details of algorithm is given in Algorithm 1.

**Algorithm 1** Dijkstra algorithm to find shortest distance for all pairs of input points

**Require:** \( \Lambda = \{s,v_1,v_2,\ldots,v_n,t\} \) and re-indexing it as \( \{P_1,P_2,\ldots,P_{n+2}\} \)

**for all** two vertices \( P_i, P_j \in \Lambda, (i \neq j, i > j) \) **do**

\[ S \leftarrow V, S^* \leftarrow \emptyset, C(P_i,P_j) \leftarrow c(P_i), C(P_i,k) \leftarrow \infty \text{ for } k \in V \setminus \{P_i\} \]

**while** \( P_j \notin S^* \) **do**

- Select vertex with minimum \( C(P_i,k) \) among \( k \in S \) and name it as \( i^* \)
- \( S \leftarrow S - \{i^*\}, S^* \leftarrow S^* \cup \{i^*\} \)
  **for all** \( w \in Suc(i^*) \) **do**
  - **if** \( C(P_i,w) > C(P_i,i^*) + c(w) \) **then**
    - \( C(P_i,w) \leftarrow C(P_i,i^*) + c(w) \)
    - \( P_{\text{Prev}(w)} \leftarrow i^* \)
  **end if**
  **end for**
**end while**
**end for**

The shortest path between \( P_i \) and \( P_j \) can be traced by tracking from \( P_{\text{Prev}(P_j)} \) until reaching \( P_{\text{Prev}(w)} = P_i \) and rewriting the whole sequence in reverse order. After executing Algorithm 1, we have a subgraph \( G' = (V',E') \), which is a complete graph made up of the paths linking all pairs of given points in \( \Lambda \). Note that \( G' \) can be represented as an \((n+2) \times (n+2)\) incidence matrix \( I_{G'} = [a_{i,j}]_{(n+2) \times (n+2)} \).

3.2. Phase 2: Search for the best path

In Phase 1, we obtain a complete graph \( G' \) which contains the cost of all pairs of points in \( \Lambda \). Our next task is to search for the best Hamilton path from \( s \) to \( t \). Note that this is similar to the well known travelling salesman problem (TSP), except that in TSP, the path starts and ends at the same point while SPP-UIP have unique starting and ending points. The nearest neighbor algorithm (NNA) is a greedy but simple algorithm that solves the TSP. It selects the immediate best (nearest) point to visit next, until all have been visited. Hence NNA executes quickly but does not guarantee the optimality of solution all the time. To increase the likelihood of getting good solution, we apply the repetitive nearest neighbor algorithm (RNNA), where the NNA is repeated \( n + 2 \) times by starting at each point in \( \Lambda \) once. Finally, the shortest path among the \( n + 2 \) solution paths is selected. The procedure is described in Algorithm 2.

In the beginning of Algorithm 2, we model the problem into traveling salesman problem by changing \( C(s,t) \) in \( G' \) into \(-M\), where \( M \) is a very big integer that exceed all the entries in \( I_{G'} \). Using this new problem set, we select one point in \( \Lambda \) as a beginning point. Then we repeatedly select the next point in \( \Lambda \) with the least cumulative cost until all the points in \( \Lambda \) are visited, and
Algorithm 2 RNNA to find best shortest complete path

Require: Re-indexed $\Lambda$ and $I_{G'} = [a_{i,j}]_{(n+2) \times (n+2)}$  
\[ a_{1,n+2} = a_{n+2,1} = -M, \text{where } M > \max_{a_{i,j} \in I_{G'}} a_{i,j} \]

for all point $P_i \in \Lambda$ do
  $\Lambda \leftarrow \Lambda - \{P_i\}$, append $P_i$ in $\pi(P_i, P_i)$
  while $\Lambda \neq \emptyset$ do
    Let $i^*$ be last node in $\pi(P_i, P_i)$
    Select $k = \min_{j \in \Lambda} a_{i^*,j}$
    Append $j$ in $\pi(P_i, P_i)$, $\Lambda \leftarrow \Lambda - \{j\}$
  end while
  Append $P_i$ in $\pi(P_i, P_i)$ to join last node in $\pi(P_i, P_i)$ to $P_i$
  $C(P_i, P_i) = \sum_{i \in \Lambda} c(P_i) + \sum_{\text{selected } k} + M$

end for
Select path with minimum $C(P_i, P_i), P_i \in \Lambda$.
Detach path $s,t$ from the selected path and rewrite the path with $s$ as starting point and $t$ as ending point.

finally return to the first point. This will form a loop. The entire process is repeated for $n+2$ times, and finally the trials which yields the least cumulative cost was selected. The selected path was rewritten and we will get a sequence of points which yields the least cost complete path.

4. Computational Experiments

For evaluation purposes, computational experiments were conducted on grid networks of different sizes ($N = 30, 50, 75, 100$) with various number of intermediate points. The cost of each cell in the grid was randomly assigned with numbers ranging from 0 to 6. The position of the starting point, destination point and the intermediate points are provided by the user. To ensure that all points are far enough apart, we set that no point is within one cell radius of another.

In all cases, Phase 1 is performed to obtain $G'$ and $I_{G'}$, then followed by Phase 2 where the best solution path is searched by RNNA. As a benchmark for solution quality measurement, Brute Force (BF) search is also performed by making a list of all possible Hamilton paths, then calculate the cost of each Hamilton path by adding up the cost of its edges. Finally, the Hamilton path with the least total cost is the optimal path. All algorithms were coded in C++ and tests were done on a PC with a 2.5GHz processor and 4GB of RAM. Computation time recorded are average time from 30 trials for each instance and the maximum computation time allowed for each instance is 300 seconds. The performance of RNNA and BF are tabulated in Table 1 and Table 2.

Table 1 shows the performance of RNNA and BF in searching for a solution path. In general, the computation time for both algorithms increase as the map size and number of intermediate points increase. Note that by using BF, the computation time starts to exceeded the time limit when $n = 11$. This is because as the number of intermediate point increases from $n$ to $n+1$, BF uses $n+1$ times as many computation time as previous one. On the other hand, RNNA always obtains a solution in under a second. We also observe that BF works well on problems with fewer than 11 intermediate points.

Table 2 shows the quality of the solution found by using BF and RNNA. From the table we can see that the solution found by both BF and RNNA are almost the same when the number of intermediate points is small. But as the number of intermediate point increases,
Table 1. Average computation time (s) of BF and RNNA in path searching

| Int pts | 30×30 | 50×50 | 75×75 | 100×100 |
|---------|-------|-------|-------|---------|
| BF      | RNNA  | BF    | RNNA  | BF      | RNNA    |
| 5       | 0.0082| 0.1401| 0.0088| 0.1441  | 0.0141  |
| 7       | 0.0551| 0.1034| 0.1248| 0.2444  | 0.0494  |
| 9       | 4.7841| 0.1821| 5.4248| 0.1976  | 3.4975  |
| 11      | 403.8979| 0.1424| 400.1723| 0.1366  | 402.1891|

Table 2. The solution quality of RNNA compared to BF

| Int pts | 30×30 | 50×50 | 75×75 |
|---------|-------|-------|-------|
| Diff count*a | % Diff Avg*b | Diff count*a | % Diff Avg*b | Diff count*a | % Diff Avg*b |
| 5       | 9     | 2.2618| 9     | 1.6250| 7     | 0.7375| 10     | 2.0422|
| 7       | 14    | 1.5994| 12    | 1.6971| 19    | 2.8215| 19     | 2.6532|
| 9       | 19    | 2.9797| 21    | 2.8959| 19    | 2.4916| 16     | 2.2945|
| 11      | 19    | 3.4323| 21    | 3.0108| 25    | 5.3175| 23     | 4.2385|

a Number of trials (out of 30 for each problem) where both BF and RNNA obtain different C(s, t).
b Average percentage difference of solution cost obtained by RNNA compared to BF.

the cost of solution path found by RNNA starts to deviate from solution found by BF. This is because RNNA is a greedy algorithm. It always chooses the nearest point at the moment without considering the effect afterwards, hence resulting differences with the optimal solution path found by BF.

To observe how well can RNNA perform with increasing problem size, we conducted another experiment by increasing the number of intermediate points gradually. The computation time in Phase 1 and Phase 2 are recorded. The result are shown in Table 3.

Table 3. Average computation time (s) of RNNA

| Int pts | 30×30 | 50×50 | 75×75 |
|---------|-------|-------|-------|
| BF      | RNNA  | BF    | RNNA  |
| 5       | 0.3126| 0.1401| 0.4527| 0.7700| 0.1441| 0.9141| 3.4081| 0.2324| 3.6405|
| 7       | 0.4525| 0.1359| 0.5884| 1.4486| 0.1626| 1.6112| 4.9650| 0.1600| 5.1250|
| 9       | 0.8427| 0.2206| 1.0633| 2.6621| 0.2435| 2.9056| 9.8358| 0.2476| 10.0834|
| 11      | 1.6473| 0.3767| 2.0240| 4.3760| 0.4180| 4.7940| 15.8896| 0.3949| 16.2845|
| 15      | 2.2728| 0.5722| 2.8450| 6.4552| 0.5538| 7.0900| 22.7885| 0.5369| 23.3254|
| 20      | 3.3194| 0.8923| 4.2117| 9.1396| 0.7403| 9.8799| 31.0910| 0.6597| 31.7507|
| 25      | 4.5330| 1.0010| 5.5340| 12.7168| 0.9275| 13.6443| 40.0883| 0.7880| 40.8763|
| 30      | 6.1171| 1.3000| 7.4171| 16.2596| 1.1103| 17.3699| 55.3650| 1.0130| 56.3780|
| 35      | 8.5896| 1.5920| 10.1817| 21.9625| 1.2740| 23.2365| 71.8285| 1.2470| 73.0755|
| 40      | 11.0228| 1.8841| 12.9049| 27.5250| 1.4390| 28.9640| 89.2945| 1.4740| 90.7685|

From Table 3, we observe that the time used in Phase 1 increases with map size and the number of intermediate points. However, the computation time for Phase 2 is within 1 second for all problem instances. This shows that RNNA has stable performance even when both map size and number of intermediate points are increased.
5. Conclusion
In this paper, we study the shortest path problem with multiple unordered intermediate points (SPP-UIP) on a grid network, and present a two phase heuristic algorithm to find the solution path in SPP-UIP. In Phase 1, we use the Dijkstra algorithm to generate the shortest paths between every pair of points that need to be visited. Then, in Phase 2, we choose the best combination of partial paths that form the best complete path using RNNA. We conduct computational experiments on randomly generated grid networks of different sizes with different number of intermediate points. Experimental results show that despite deviation in solution found by RNNA from optimal solution, RNNA uses much lesser time to solve SPP-UIP disregarding the map size and the number of intermediate points. Therefore, we conclude that RNNA can be applied to solve SPP-UIP with large map size and large number of intermediate points.

This research can be extended in several ways. For example, we can consider SPP-UIP with multiple cost factors, as many real life optimization problems are multi-objective based. Besides that, we can also consider cases where the cost values in the grid network changes over the time, which occurs in situations such as rescue path planning in disaster area, where the cost value function are rarely constant. We may also consider the cases where the intermediate points are in clustered form. Such cases can be applied in route planning where we need to visit multiple sites in a cluster region before departing to another region.

Acknowledgments
The authors would like to acknowledge the funding by Universiti Sains Malaysia Short Term Research Grant 304/PMATHS/6313066.

References
[1] Alexander S 2002 Combinatorial Optimization: polyhedra and efficiency *Springer Science & Business Media* 24
[2] Appleton B and Sun C 2003 Circular shortest paths by branch and bound *Pattern Recognition* 36 (11) pp 2513-20
[3] Bae K Y, Kim Y D and Han J H 2015 Finding a risk-constrained shortest path for an unmanned combat vehicle *Computers & Industrial Engineering* 80 pp 245-253
[4] Cormen T H 2009 *Introduction to Algorithms*, 3rd ed (USA: MIT Press)
[5] De Mesquita Sá Junior J J, Backes A R and Cortez P C 2013 Texture analysis and classification using shortest paths in graphs *Pattern Recognition Letters* 34 (11) pp 1314-19
[6] Han D H, Kim Y D and Lee J Y 2014 Multiple-criterion shortest path algorithms for global path planning of unmanned combat vehicles *Computers & Industrial Engineering* 71 pp 57-69
[7] Hart P E, Nilsson N J and Raphael B 1968 A formal basis for the heuristic determination of minimum cost paths *IEEE Transactions on System Science and Cybernetics* 4 (2) pp 100-107
[8] Kuby M, Araz O M, Palmer M and Capar I 2014 An efficient online mapping tool for finding the shortest feasible path for alternative-fuel vehicles *International Journal of Hydrogen Energy* 39 (32) pp 18433-39