Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Computing Close to Optimal Weighted Shortest Paths in Practice

Nguyet Tran, Michael J. Dinneen, Simone Linz
School of Computer Science, University of Auckland, Auckland, New Zealand
ntra770@aucklanduni.ac.nz, mjd@cs.auckland.ac.nz, s.linz@auckland.ac.nz

Abstract

This paper proposes a new practical method for the weighted region problem (WRP). The objective of WRP is to find a minimum cost path between two vertices among different regions where each region incurs a traversal cost per unit distance. Currently, there is no practical algorithm that solves this problem exactly. Among the approximation methods that solve instances of WRP, there is a limited number of algorithms that compute paths whose lengths are close to optimal, which we call very-close optimum paths. However, they are considered as theoretical methods. On the other hand, algorithms for solving WRP that can be applied to practical data sets (using decomposition ideas or heuristics) are not guaranteed to find a very-close optimum path within an acceptable amount of time. In this paper, we consider an alternative method for solving WRP that exploits Snell's law of physical refraction. We compare the performance of our new algorithm with that of two existing algorithms, using at least 500 test cases for each such comparison. The experimental results show that our algorithm returns a very-close optimum weighted shortest path in reasonable time.

Introduction

Let $WS = (T, E, V)$ be a continuous two-dimensional workspace, where T, E and V are the set of *regions*, *region edges* and *vertices*, respectively. The regions are non-overlapping polygons, which are defined by vertices and edges (see Figure 3). We consider each region $t_i \in T$ as a triangle since any polygon that is not a triangle can be divided into triangles. Traversing each region $t_i \in T$ incurs a unit *cost* (or *weight*) $w_i > 0$. Let $w(e_i)$ be the unit weight of a region edge $e_i = (p_i, q_i) \in E$, where p_i and q_i are two vertices in V. If e_i has two adjacent regions t_a and t_b, $w(e_i) = \min(w_a, w_b)$, where w_a and w_b are the unit weights of t_a and t_b, respectively. Otherwise, if e_i only belongs to one region t_a, which is at the border of the workspace (or two regions, but t_b has $w_b = +\infty$), $w(e_i) = w_a$ and we call e_i as a *border edge*. Let $s = (p, q)$ be a segment between any two endpoints p and q. We define $d(p, q)$ as the Euclidean distance between p and q, and $D(p, q) = w \cdot d(p, q)$ as the *weighted length*, where w is the unit weight of the region or the edge that s is on. Then, for a pair of two vertices $u, v \in V$, the weighted region problem (WRP) asks for the minimum cost (or the weighted shortest) path $P^*(u, v) = (u = o_0, o_1, \ldots, o_k, o_{k+1} = v)$ such that the weighted distance $\sum_{i=0}^{k} D(o_i, o_{i+1})$ is minimum, where points o_i, $i \in \{1, \ldots, k\}$, are crossing points on edges in E or vertices in V of the workspace. For short, we call a path whose weighted length is "close to" the weighted length of $P^*(u, v)$ as a *very-close optimum path* (formally defined later).

WRP is a classical path planning problem, which has a large range of applications in robotics, geographical planning and manufacturing (Mitchell 2017). For example, the energy-consuming levels of a robot (or its moving speeds) can be different depending on the moving regions, which can be smooth flat, desert, rocks, water, forest, grassland, etc. Thus, if the corresponding energy consumption (or the speed) is modeled as its unit cost or weight, finding a weighted shortest path turns into finding an optimal energy path (or a minimum time path) for the robot.

Interestingly, the optimum path of WRP does not necessarily go straight when crossing a region edge, or it may only traverse along a portion of a region edge (see Figure 3). In solving an instance of WRP, the following question arises. Which sequence of region edges and which point on each region edge does the minimum cost path cross? It was established in (Mitchell and Papadimitriou 1991) that a path between two vertices is the minimum cost path if and only if it obeys Snell's law from physics each time it crosses a region edge. That is, $P^*(u, v)$ crosses a region edge as a ray of light crosses the boundary of two different isotropic media (see Known Fact 1).

However, even if a sequence of region edges that has to be crossed in order is given, finding the optimum path between two vertices such that the path respects the given sequence and each crossing boundary obeys Snell's law is still hard. The equation of Snell's law at each region edge is known and, so, a classical idea is to use an algebraic system of equations to find the crossing points of the shortest path. However, De Carufel et al. (2014) have shown that, even for a sequence of two region edges, the resulting system of equations of degree six cannot be solved in

Algebraic Computation Model over the Rational Numbers (ACM$\mathbb{Q}$). As a consequence, the exiting algorithms to solve WRP are mostly approximations. To give an overview of the existing approaches, we divide them into three groups: (1) exploiting Snell's law, (2) using heuristic methods, and (3) applying decomposition ideas. We next summarize some algorithms of each group. For a more detailed list of related work, we refer the interested reader to (Mitchell 2017; Bose et al. 2011).

One popular method of Group (1) is the work by (Mitchell and Papadimitriou 1991). Their approach uses ideas based on Snell's refraction law and continuous Dijkstra, and yields a $(1+\epsilon)$-approximation for finding a minimum weight shortest path in time $O(n^8 \log(nNW/w\epsilon))$, where n is the number of region edges, N is the maximum integer coordinate of any vertex, W and w are the maximum and minimum unit weights over all regions in the workspace, respectively. While the survey (Goerzen, Kong, and Mettler 2009) claims that the algorithm by Mitchell and Papadimitriou solves WRP exactly, we clarify here that their method computes an approximation solution. Indeed, their approach computes a path whose cost is at most $(1 + \epsilon)$ times the exact minimum cost. However, while $1/\epsilon$ of the other approximation solutions is polynomial, $1/\epsilon$ of the algorithm by Mitchell and Papadimitriou is the only known logarithmic solution. Hence, ignoring other factors, the this method can run, in reasonable time, with a very-close optimum result. However, when all factors involved in the time complexity, this method is considered as a theoretical method rather than a practical one (Szczerba, Chen, and John J. Uhran 1998; Jaklin, Tibboel, and Geraerts 2014). Furthermore, the algorithms by (Rowe and Alexander 2000; Rowe and Richbourg 1990) are similar in that they exploit Snell's law, but have been evaluated as unrealistic methods (Szczerba, Chen, and John J. Uhran 1998).

The heuristic approaches of Group (2), are more advantageous for quickly finding a solution than for finding very-close optimum path. Such methods can be found in (Kindl and Rowe 2012) that uses a heuristic search combined with simulated annealing, and in (Xidias 2019) that uses a genetic algorithm to solve WRP constrained by a list of points that a path needs to cross.

For approaches of Group (3), the main idea is to partition the workspace into cells, or to discretize all region edges into points (named *Steiner points*). Then, based on all resulting cells (resp. points), a grid (resp. graph) is constructed in a second step before an approximated minimum cost path is computed. A list of algorithms that carefully place Steiner points and analyze asymptotic bounds can be found in the survey of Mitchell (2017). While the use of Steiner points is a common approach for solving WRP in geometry, grid-based methods are widely used in robotics. These methods are practical and some can be combined with other constraints, such as (Jaklin, Tibboel, and Geraerts 2014; Xidias 2019; Zheng et al. 2010; Szczerba, Chen, and John J. Uhran 1998). If, however, a particular application requires the computation of very-close optimum path, the resolution of the cell grids or the density of Steiner points on region edges must be large. Thus, these methods might not always have practical running times due to searching large grids or graphs.

An interesting point made by Rowe and Kindl (2012) is the following: *There is a misperception that the WRP has been "solved" since it has been discussed for a long time, but this is incorrect because current practice heavily uses approximation algorithms.* The works of (Gheibi et al. 2018; Mitchell 2017) also claim that there is no known exact solution for WRP. To our knowledge, we still have no practical solution that can compute a very-close optimum path in reasonable time. One might argue that many practical applications just need a fast approximate result. However, we believe, finding a very-close optimum path is still scientifically desirable and important for many real-world applications, such as GIS planning, laying of pipelines or electrical cables.

Our work summary

We now formally define a very-close optimum path. Let δ be an extremely small value. If the Euclidean distance between two points or a point to a line in a geometrical workspace is smaller than δ, two points are considered to be the same or the point is considered on the line, respectively. With $P^*(u, v) = (u, o_1, \ldots, o_k, v)$ being an exact optimal path between two vertices u and v in WRP, given δ, we define the path $P^*_c(u, v) = (u, r_1, \ldots, r_k, v)$ to be a very-close optimum path with respect to δ if for every $i \in \{1, \ldots, k\}$, $d(o_i, r_i) < \delta$, where $d(o_i, r_i)$ is the Euclidean distance between o_i and r_i.

1. We present in this paper a practical algorithm of finding a very-close optimum path $P^*_c(u, v)$ between any two vertices u and v in a workspace.
2. Since (Mitchell and Papadimitriou 1991) mentioned that WRP is sensitive in practice, in this paper, we choose to prove the feasibility and also benchmark the running time of our algorithm by experiments[1].
3. Because of no known exact optimal solution for WRP, and because other very-close optimum solutions are not practical, with $\delta = 10^{-5}$, we first show that our results are nearly equal to, but being remarkably faster than those of using *exact quadratic programming* with a sequence of five edges.
4. In a complete workspace, where exact quadratic programming cannot be applied, we show that our weighted paths are always shorter than those of one common solution that is based on decomposition ideas using Steiner points in which each edge in the workspace is divided into 6 to 400 Steiner points.
5. Furthermore, due to using the D-graph (defined later), our algorithm can rapidly find the weighted shortest path between *any two vertices* in the workspace, without recomputing over the whole workspace. This is preferable for applications that need to explore different paths in many times on the same map.

[1]Theoretical running time with asymptotic bounds will be presented in the extended version.

Proposed Algorithm

We first present how to find a very-close optimum cost path with respect to δ between two vertices crossing a given edge sequence. Then, in the main algorithm, we describe how to determine all adequate edge sequences in a full workspace of triangular regions to finally find a very-close optimum cost path between any pair of vertices. With S and S' being two sequences or arrays, to this end, we use the notation $S \circ S'$ with the meaning of appending S' to the end of S.

Weighted shortest path crossing an edge sequence

Let $P(u,v) = (u = r_0, r_1, \ldots, r_k, r_{k+1} = v)$ be a path between two vertices $u, v \in V$, crossing an edge sequence $S = (e_1, e_2, \ldots, e_k)$ in order, where r_i is on e_i with every $i \in \{1, \ldots, k\}$. Let $W = (w_0, \ldots, w_k)$ be the weight list of S, where every w_i, $i \in \{0, \ldots, k\}$, is the unit weight of the region between e_i and e_{i+1}, with $e_0 = (u,u)$ and $e_{k+1} = (v,v)$. For each $e_i = (p_i, q_i) \in S$, we name two endpoints p_i and q_i such that when moving from p_i to q_i on e_i, the edge e_{i-1} is on the right while the edge e_{i+1} is on the left of e_i (see Figure 1b). Two vertices u and v can also be on e_1 and e_k at the endpoints, respectively. When an edge sequence S satisfies this requirement, we say that S satisfies the *sequence arrangement* (the case that S cannot satisfy the sequence arrangement will be processed in the next section).

Let $n(r_i, e_i)$ be the line perpendicular to e_i at the point r_i, $i \in \{1, \ldots, k\}$. Let α_i and β_i be two acute angles at r_i created by $n(r_i, e_i)$ and the segments (r_{i-1}, r_i) and (r_i, r_{i+1}), respectively (see Figure 1a). We say that the path $P(u,v)$ comes in and out the edge e_i at r_i with the *in-angle* α_i and the *out-angle* β_i, respectively.

Known Fact 1: Snell's law

For every crossing point r_i in $P(u,v)$, $i \in \{1, \ldots, k\}$, if r_i is not an endpoint of e_i, the path $P(u,v)$ has the minimum weighted crossing S if and only if the *in-angle* α_i and the *out-angle* β_i of $P(u,v)$ at e_i satisfy: $w_{i-1} \sin \alpha_i = w_i \sin \beta_i$, where $w_i \in W$ is the unit weight of the region between e_i and e_{i+1}.

Known Fact 2: Critical point

For every edge $e_j \in S$, $j \in \{1, \ldots, k\}$, if $w_j < w_{j-1}$ (resp. $w_j > w_{j-1}$), the angle $\alpha_c = \sin^{-1}(w_j/w_{j-1})$ (resp. $\alpha_c = \sin^{-1}(w_{j-1}/w_j)$) is defined as the *critical angle* of e_j. If the path $P(u,v)$ comes in e_j with an *in-angle* $\alpha_j = \alpha_c$, obeying Snell's law, the *out-angle* will be $\beta_j = 90°$ (see Figure 1a). The point r_j is called a *critical point*.

The proofs of the Known Facts 1 and 2 can be found in (Mitchell and Papadimitriou 1991). We next present two new definitions as follows.

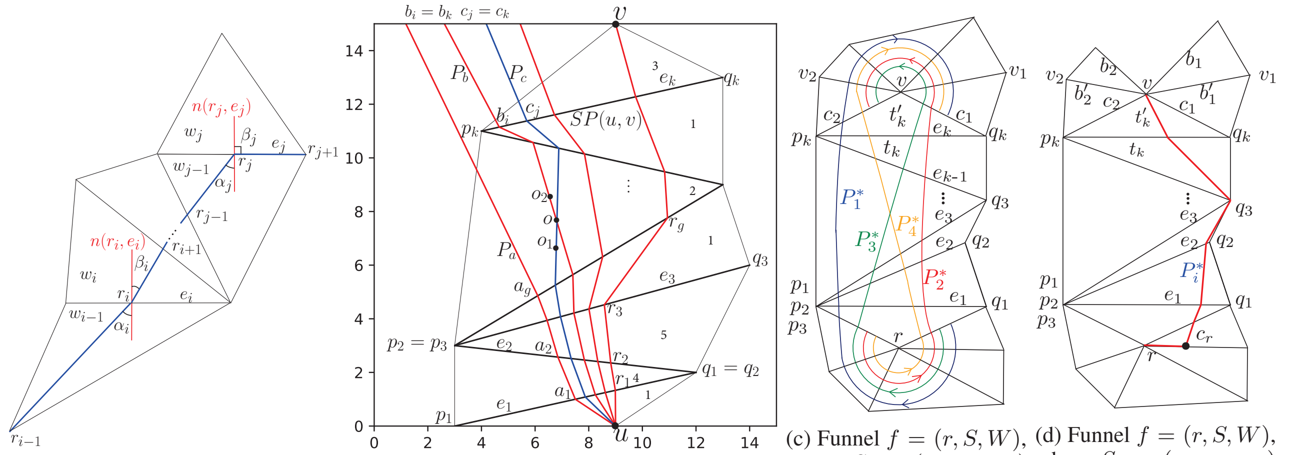
Snell path: If the path $P(u,v)$ satisfies: (1) every point r_i, $i \in \{1, \ldots, k\}$, is on the interior of e_i, which is not one of two endpoints of e_i, and (2) Snell's law is obeyed at each r_i, we call $P(u,v)$ as a *Snell path*, named $SP(u,v)$.

Snell ray: On the other hand, let a_1 be a point on $e_1 \in S$, applying Snell's law from u crossing e_1 at a_1, we can find the *out-ray* $\mathcal{R}_1^a$. Suppose that $\mathcal{R}_1^a$ intersects $e_2 \in S$ at a point a_2. Then, we can continue calculating the path $P_a = (u, a_1, a_2, \ldots, a_g, \mathcal{R}_g^a)$, where $1 \le g \le k$ and $\mathcal{R}_g^a$ is the *out-ray* of the path at $e_g \in S$. We define P_a to be a *Snell ray* of u, starting at the point a_1, through a sequence of g segments from e_1 to e_g of S (see Figure 1b).

Proposition 1. *Let $P_b = (u, b_1, \ldots, b_i, \mathcal{R}_i^b)$ and $P_c = (u, c_1, \ldots, c_j, \mathcal{R}_j^c)$ be two Snell rays from u to $e_i \in S$ and $e_j \in S$, respectively, crossing the same edge sequence $S = (e_1, \ldots, e_k)$, where $b_1 \neq c_1$, $i \le k$ and $j \le k$. Two Snell rays P_b and P_c cannot intersect each other.*

Proof. This is deduced from Lemma 4.2 of (Mitchell and Papadimitriou 1991). From Known Fact 1, two Snell paths $SP(u, b_i) = (u, b_1, \ldots, b_i)$ and $SP(u, c_j) = (u, c_1, \ldots, c_j)$ are two minimum weighted length paths from u to b_i and c_j crossing S, respectively. By contradiction, suppose that P_b and P_c intersect each other at a point o (see Figure 1b). Let $P_b^o = (u, b_1, \ldots, o)$ and $P_c^o = (u, c_1, \ldots, o)$ be two subpaths of P_b and P_c from u to o, respectively. We have $D(P_b^o) = D(P_c^o)$. This is because, if $D(P_b^o) > D(P_c^o)$, the path $SP(u, b_i)$ can follow P_c^o to have a less weighted length than its current one, which is contrary to the fact that $SP(u, b_i)$ has the minimum weighted length between u and b_i crossing S. This is also similar for the case $D(P_b^o) < D(P_c^o)$. Thus, $D(P_b^o) = D(P_c^o)$. Due to this, now, the Snell path $SP(u, b_i)$ can go to o following the path P_c^o instead of P_b^o with the weighted length not being changed. However, in this situation, if we get two points o_1 and o_2 that are very close to o, then the path $SP(u, b_i)$ can be locally improved by crossing from o_1 to o_2 ignoring o to get a shorter weighted length. This is again contrary to that $SP(u, b_i)$ has the minimum weighted length between u and b_i. $\square$

Find the Snell path $SP(u,v)$ crossing the k edge sequence S is presented in the function *Find-Snell-Path*. The main idea of the function is to use an iteration scheme, as follows. Let $P_m = (u, m_1, \ldots, R_g^m)$ be a Snell ray of u starting from the middle point m_1 of e_1 crossing S. Based on Proposition 1, if v is on the left (resp. right) of P_m, then the Snell ray $SP(u,v)$ must cross only the parts from p_i to m_i (resp. from m_i to q_i) of edges $e_i \in S$. Thus, we modify the original edges $e_i = (p_i, q_i) \in S$ to the new ones $e'_i = (p'_i, q'_i) = (p_i, m_i)$ (resp. $e'_i = (p'_i, q'_i) = (m_i, q_i)$). This process is repeated until P_m crosses v, or all of the new edges e'_i have $d(p'_i, q'_i) < \delta$. We employ this idea from the function *Find-Point* in the work of (Mitchell and Papadimitriou 1991). However, when there exists a new edge $e'_i = (p'_i, q'_i)$ of the original edge $e_i = (p_i, q_i)$ that has $d(p'_i, q'_i) < \delta$, but $d(m'_i, p_i) < \delta$ or $d(m'_i, q_i) < \delta$, where m'_i is the middle point of e'_i, we will stop the finding process. This is because, in this situation, the minimum weighted path from u to v crossing S is considered as crossing one of the endpoints of $e_i \in S$. Thus, the Snell path $SP(u,v)$ cannot exist. In this case, the function *Find-Snell-Path* will return the *crossing endpoint sides*. That is, if the crossing point is p_i or q_i of e_i, the crossing endpoint side is *left* (L) or *right* (R), respectively. The reason of this will be presented in the main algorithm section. In the function *Find-Snell-Path*, since the edges in S will be modified, but the original ones are still needed, we use one alternative edge sequence S', initially being S, as follows.



(a) Illustration of Snell's law at r_i and a critical point at r_j.

(b) $SP(u, v)$ and Snell rays from u crossing a k edge sequence S, where S satisfies the sequence arrangement and $W = (1, 4, 5, \ldots, 1, 3)$.

(c) Funnel $f = (r, S, W)$, where $S = (e_1, \ldots, e_k)$ with directions of four paths from P^*_1 to P^*_4.

(d) Funnel $f = (r, S, W)$, where $S = (e_1, \ldots, e_k)$ with Snell path P^*_i and its critical point c_r.

Figure 1: Snell path illustrations.

```
Find-Snell-Path:
Input: $u, v, S, W$
Output: If there exists the Snell path $SP(u, v)$[2], return $SP(u, v)$. Otherwise, if the path crosses one of the endpoints of edges in $S$, return the crossing endpoint side $L$ or $R$.
1. Initial: $SP(u, v) = (\,)$; $l = 1$; $root = u$; $S' = S$
2. Let $m_l$ be the middle point of $e'_l = (p'_l, q'_l) \in S'$. If $d(p'_l, q'_l) < \delta$, go to Step 4. Otherwise, go to Step 3.
3. Find the Snell ray $P_m = (root, m_l, m_{l+1}, \ldots, \mathcal{R}^m_g)$ from $root$, starting at $m_l$, crossing $S'$, where $g \leq k$, and the last ray $\mathcal{R}^m_g$ cannot intersect $e'_{g+1} \in S'$. Consider two cases:
  3.1 $P_m$ crosses the last segment $e'_k \in S'$ ($g = k$):
    – If $v$ is on the last ray $\mathcal{R}^m_g$ of $P_m$: $SP(u, v) = SP(u, v) \circ (root, m_l, m_{l+1}, \ldots, m_k, v)$. Go to Step 5.
    – Otherwise, if $v$ is on the left (resp. right) side of $\mathcal{R}^m_g$, then for every $e'_i = (p'_i, q'_i) \in S'$, $i \in \{l, \ldots, k\}$, set $(p'_i, q'_i) = (p'_i, m_i)$ (resp. $(p'_i, q'_i) = (m_i, q'_i)$). Go to Step 2.
  3.2 $P_m$ stops at $e'_g \in S'$ ($g < k$):
    – If $P_m$ comes in $e'_g = (p'_g, q'_g)$ at a critical angle: If $\mathcal{R}^m_g$ goes to $p'_g$ (resp. $q'_g$), then for every $e'_i = (p'_i, q'_i) \in S'$, $i \in \{l, \ldots, g\}$, set $(p'_i, q'_i) = (m_i, q'_i)$ (resp. $(p'_i, q'_i) = (p'_i, m_i)$).
    – Otherwise, $\mathcal{R}^m_g$ does not intersect $e'_{g+1}$. If $e'_{g+1}$ is on the left (resp. right) side of $\mathcal{R}^m_g$, then for every $e'_i = (p'_i, q'_i) \in S'$, $i \in \{l, \ldots, g\}$, set $(p'_i, q'_i) = (p'_i, m_i)$ (resp. $(p'_i, q'_i) = (m_i, q'_i)$).
    – Go to Step 2.
4. Get two endpoints $p_l$ and $q_l$ of $e_l \in S$.
 4.1. If $d(m_l, p_l) < \delta$, return $L$. If $d(m_l, q_l) < \delta$, return $R$.
 4.2. Otherwise, $SP(u, v) = SP(u, v) \circ (root)$. If $l = k$, $SP(u, v) = SP(u, v) \circ (m_l, v)$; Go to Step 5. If $l < k$, $root = m_l$; $l = l + 1$; Go to Step 2.
5. Return $SP(u, v)$.
```

Lemma 1. *In the worst case, the function Find-Snell-Path needs $O\left(k \log\left(L/\delta\right)\right)$ loops to find $SP(u, v)$ crossing S, where L is the largest Euclidean length of edges in S.*

Proof. Let L_l be the Euclidean length of e_l, $l \in \{1, \ldots, k\}$. In the function *Find-Snell-Path*, from $root$, we take the middle point of e_l, and finding the Snell ray. These steps are repeated until the length $d(p_l, q_l) < \delta$. Thus, we need $\log(L_l/\delta)$ loops. Since l is from 1 to k, the number of loops in the worst case is $O(k \log(L/\delta))$. $\square$

Main algorithm

To describe the main idea of our algorithm, we first observe an example as follows. Suppose that the final weighted shortest path $P^*(u, v)$ between two vertices u and v contains another vertex $g \in V$ such that $P^*(u, v) = P^*(u, g) \circ P^*(g, v)$. Let S_{ug} and S_{gv} be two edge sequences that $P^*(u, g)$ and $P^*(g, v)$ cross, respectively. The edge sequence that $P^*(u, v)$ crosses is $S = S_{ug} \circ S_{gv}$. Suppose that there exists another edge sequence $S' = S'_{ug} \circ S_{gv}$, where the last edge sequences S_{gv} of S and S' are the same, and the minimum weighted path between u and v crossing S' also crosses the vertex g when crossing S'_{ug}. In this situation, the method of (Mitchell and Papadimitriou 1991) finds the minimum weighted length paths between u and v with both S and S' separately, then choosing the one that has smaller weighted length. However, with two such edge sequences S and S', our solution can avoid unnecessarily repeated calculations on the sequence S_{gv}. Our main idea is as follows.

[2]We note that, $SP(u, v) = (u, r_1, \ldots, r_k, v)$, which is returned by the function *Find-Snell-Path*, is a very-close optimum path with respect to δ, but not necessarily an exact Snell path. Let $SP^*(u, v) = (u, o_1, \ldots, o_k, v)$ be the exact Snell path, the function *Find-Snell-Path* only guarantees that $d(o_i, r_i) < \delta$ for every $i \in \{1, \ldots, k\}$. However, to avoid creating new definitions, to this end, we generally say that the function *Find-Snell-Path* returns the Snell path $SP(u, v)$ between the u and v crossing S.

We use an undirected graph named D-graph = (V_D, E_D), where $V_D = V \cup V_c$ with V_c being the set of critical points (explained later). In the workspace, for each pair of two vertices $u, v \in V$, if there exist Snell paths between u and v, which only cross the interiors of edges, an edge between u and v in E_D is created. The weight of this edge in E_D is the minimum weighted length among Snell paths crossing all different possible edge sequences between u and v. With such the D-graph, we can finally find the last weighted shortest path between any pair of vertices in V by using any shortest path graph algorithm, such as Dijkstra. Thus, our main algorithm has two steps: (1) build the D-graph and (2) apply a shortest path graph algorithm on the D-graph to find the weighted shortest path between any pair of vertices. Since step (2) is easy, we only present how to find the D-graph in the function *Build-D-graph*. To support the function, we use a data structure called *funnel* and a function named *Find-Path*, as follows.

We denote $f = (r, S, W)$, where $S = (e_1, \ldots, e_k)$ and $W = (w_0, \ldots, w_{k-1})$ as a funnel with $r \in V$ being the root and the last edge $e_k \in S$ being the bottom of the funnel. The edge sequence S is defined as in the function *Find-Snell-Path* such that S must satisfy the sequence arrangement. Let t_k be the triangle that contains e_k and e_{k-1} in S of f (see Figure 1c). In case e_k is not a border, let t'_k be the adjacent triangle of t_k at e_k. Let v be a vertex of t'_k that is not on e_k, and c_1 and c_2 be two adjacent edges at v of t'_k. We need to find the Snell path from r to v crossing S. However, the Snell path from r to v crossing S might go around adjacent edges at r and adjacent edges at v with critical points. Thus, the function *Find-Path* helps connect the adjacent edges at r and v into S before finding the Snell paths by the function *Find-Snell-Path*.

For adjacent edges at r and v, there are two considering cases. (1) If there is no border edge among adjacent edges at v (see Figure 1c), we call the edge sequences of adjacent edges at v from c_2 to c_1 clockwise and from c_1 to c_2 counterclockwise, including both c_1 and c_2, as the left and the right adjacent sequences of v, respectively. (2) Otherwise, if there are two border edges b_1 and b_2 among adjacent edges at v (see Figure 1d), let b'_1 and b'_2 be two adjacent edges at v that are in the same triangles with b_1 and b_2, respectively. We call the edge sequence from c_2 to b'_2, including both c_2 and b'_2, as the left adjacent sequence of v, and the edge sequence from c_1 to b'_1, including both c_1 and b'_1, as the right adjacent sequence of v. The left and right adjacent sequences of r are determined similarly.

Let L_r, R_r, L_v and R_v be the left and right adjacent sequences of r and v, respectively. There are at most four edge sequences $S_1 = L_r \circ S \circ L_v$, $S_2 = R_r \circ S \circ R_v$, $S_3 = L_r \circ S \circ R_v$ and $S_4 = R_r \circ S \circ L_v$ that the path from r to v can cross. We note that, to this end, we only consider the general case that these four edge sequences are different. We can similarly prove the special cases where a part of these edge sequences are equal to each other. Thus, correspondingly, there are four minimum weighted shortest paths $P_i^*(r, v)$ (P_i^* for short) from r to v crossing S_i, where $i \in \{1, \ldots, 4\}$ (see Figure 1c). For every $i \in \{1, \ldots, 4\}$, if the path P_i^* does not cross any endpoint of edges in S_i, P_i^* becomes the Snell path crossing S_i.

Proposition 2. *For the four paths from P_1^* to P_4^*, we have:*
1. *If P_1^* (resp. P_2^*) crosses a q (resp. p) endpoint of an edge in S_1, then P_2^* (resp. P_1^*), P_3^* and P_4^* must also cross q (resp. p) endpoints of edges in S_2 (resp. S_1), S_3 and S_4, respectively.*
2. *If P_3^* (resp. P_4^*) crosses a q (resp. p) endpoint of an edge in S_3 (resp. S_4), then P_2^* and P_4^* (resp. P_1^* and P_3^*) must also cross q (resp. p) endpoints of edges in S_2 and S_4 (resp. S_1 and S_3), respectively.*

Proof. Suppose that P_1^* crosses an edge $e_i = (p_i, q_i)$ at q_i (the edge e_i must be in S). Let P_1 be the Snell ray that contains the Snell path from r to q_i crossing S_1 in P_1^*. By contradiction, suppose that P_3^* does not cross any q endpoint of edges in S_3. In this case, P_1 and P_3^* must intersect each other in the regions containing S. However, P_1^* (containing P_1) and P_3^* cross the edge sequences $S_1 = L_r \circ S \circ L_v$ and $S_3 = L_r \circ S \circ R_v$, respectively. Therefore, S_1 and S_3 have the same $L_r \circ S$, and from Proposition 1, P_1 and P_3 cannot intersect each other in $L_r \circ S$. Thus, P_3^* must cross a q endpoint of an edge in S_3. We can similarly prove that P_4^* must cross a q endpoint of an edge in S_4 because S_1 and S_4 also have the same $S \circ L_v$. The path P_4^* crossing a q endpoint of an edge in S_4 entails that P_2^* must also cross a q endpoint of an edge in S_2 because S_2 and S_4 have the same $R_r \circ S$. The proof of the second claim is analogous to the proof of the first claim. □

Based on Proposition 2, the function *Find-Path* below can avoid finding all four P_1^* to P_4^* to return the one that does not cross any endpoint and has the minimum weighted length. Additionally, for each edge in $e_i \in E$, we use an array A_c to store critical points that appear on e_i after finding the Snell path (see Step 8 of the function *Find-Path*). We note that, there can be at most two critical points that only appear on adjacent edges at r and v (see an example of a critical point c_r in Figure 1d). The reason why we process critical points as in Step 8 of the function *Find-Path*, along with why we need to return the crossing endpoint sides if no Snell path exists, will be explained after the function *Build-D-graph*.

***Find-Path*:**
Input: (r, v, S, W, V_D, E_D)
Output: If there exist Snell paths from r to v crossing S and adjacent edges at r and v, where they do not cross any endpoint, the output is the one with minimum weighted length. Otherwise, if all four P_1^* to P_4^* cross endpoints, the output is the crossing endpoint sides, L or R or both (L and R).

```
1. Initial: $SP_{min} = (\,)$
2. Determine $L_r, R_r, L_v, R_v$. Set $S_1 = L_r \circ S \circ L_v$, $S_2 =$
   $R_r \circ S \circ R_v$, $S_3 = L_r \circ S \circ R_v$ and $S_4 = R_r \circ S \circ L_v$. Let
   $W_i$ be the weight sequence corresponding to $S_i$, where
   $i \in \{1, \ldots, 4\}$.
3. Run the function Find-Snell-Path$(r, v, S_1, W_1)$ to find
   $P_1^*$. If $P_1^*$ does not cross any endpoint, which is the Snell
   path returned by Find-Snell-Path, calculate $D(P_1^*)$. Otherwise,
   if $P_1^*$ crosses a $q$ endpoint (because Find-Snell-Path
   returns $R$), return $R$. If $P_1^*$ crosses a $p$ endpoint (because
   Find-Snell-Path returns $L$), $D(P_1^*) = +\infty$.
```

4. Run the function *Find-Snell-Path*(r, v, S_2, W_2) to find P_2^*. If P_2^* does not cross any endpoint, calculate $D(P_2^*)$. Otherwise, if P_2^* crosses a p endpoint, return L. If P_2^* crosses a q endpoint, $D(P_2^*) = +\infty$.
5. Run the function *Find-Snell-Path*(r, v, S_3, W_3) to find P_3^*. If P_3^* does not cross any endpoint, calculate $D(P_3^*)$. Otherwise, $D(P_3^*) = +\infty$. Then, if P_3^* crosses a q endpoint, go to Step 7. Otherwise, go to Step 6.
6. Run the function *Find-Snell-Path*(r, v, S_4, W_4) to find P_4^*.

6.1. If P_4^* does not cross any endpoint, calculate $D(P_4^*)$. The result SP_{min} is determined by $D(SP_{min}) = \min(D(P_1^*), D(P_2^*), D(P_3^*), D(P_4^*))$. Go to Step 8.

6.2. If P_4^* crosses a p endpoint (in this case, P_1^* and P_3^* must also cross p endpoints): If $D(P_2^*) \neq +\infty$, $SP_{min} = P_2^*$. Go to Step 8. Otherwise, return (L and R) (because P_2^* crosses a q endpoint while P_1^*, P_3^* and P_4^* cross p endpoints).

6.3. If P_4^* crosses a q endpoint (in this case, P_2^* must also cross a q endpoint): If $D(P_1^*) = D(P_3^*) = +\infty$, return both ($L$ and R) (because P_1^* and P_3^* cross p endpoints while P_2^* and P_4^* cross q endpoints). Otherwise, the result SP_{min} is determined by $D(SP_{min}) = \min(D(P_1^*), D(P_3^*))$. Go to Step 8.

7. P_3^* crosses a q endpoint (in this case, P_2^* and P_4^* must also cross q endpoints): If $D(P_1^*) \neq +\infty$, $SP_{min} = P_1^*$. Go to Step 8. Otherwise, return both (L and R) (because P_1^* crosses a p endpoint while P_2^*, P_3^* and P_4^* cross q endpoints).
8. Consider (at most) two critical points in SP_{min}: If there exists a critical point c_r (resp. c_v) on an edge e_r (resp. e_v) among edges that are adjacent at r (resp. v), insert c_r (resp. c_v) into V_D and into the array A_c of e_r (resp. e_v). Then, create one edge between v and c_r (resp. between r and c_v) in E_D with the weight being $D(v, c_r)$ (resp. $D(r, c_v)$) according to SP_{min}.
9. Return SP_{min}.

Next, with a complete workspace, how to build the whole D-graph with the help of funnels and the function *Find-Path* is presented in the function *Build-D-graph*, as follows.

***Build-D-graph*:**
Input: $WS = (T, E, V)$
Output: D-graph (V_D, E_D)

1. Initial: $V_D = V$. Assign a matrix $|V| \times |V|$ to E_D. For every $1 \leq i, j \leq |V|$, $E_D[i][j] = 0$ if $i = j$ and $E_D[i][j] = +\infty$ if $i \neq j$. For every $e_i \in E$, let A_c be an array that saves all critical points appearing on e_i, $A_c = [\,]$. Let Q be a queue that contains funnels, $Q = \{\,\}$.
2. For each vertex $u \in V$:

2.1. Let T_i be the set of triangles that are adjacent at u. For each triangle $t_i \in T_i$, let e_i be the edge of t_i that does not contain u, create a funnel $f = (u, S = (e_i), W = (w_i))$, where w_i is the unit weight of t_i. Push f into Q.

2.2. Loop until Q is empty

2.2.1. Pop one funnel $f = (r, S = (e_1, \ldots, e_k), W = (w_0, \ldots, w_{k-1}))$ out Q. Let t_k be the triangle that contains e_k and e_{k-1}. Consider two cases. If e_k is a border, go to Step 2.2. Otherwise, let t'_k be the adjacent triangle of t_k at e_k with the unit weight w'_k. Let v be a vertex of t'_k that is not on e_k. If $E_D[v][r] \neq +\infty$, go to Step 2.2.3. Otherwise, go to Step 2.2.2.

2.2.2. Find the Snell path $SP(r, v)$ (if exists) by the function *Find-Path*$(r, v, S, W \circ (w'_k), V_D, E_D)$.
– If *Find-Path* returns $SP(r, v)$ and if $D(SP(r, v)) < E_D[r][v]$, then $E_D[r][v] = D(SP(r, v))$. Go to Step 2.2.3.
– If *Find-Path* returns L or R, then go to Step 2.2.4. If *Find-Path* returns both (L and R), go to Step 2.2.

2.2.3. Let c_1 and c_2 be two adjacent edges at v in the triangle t'_k. Create two funnels $f_1 = (r, S_1, W_1)$ and $f_2 = (r, S_2, W_2)$, where $S_1 = S \circ (c_1)$, $S_2 = S \circ (c_2)$, $W_1 = W \circ (w'_k)$ and $W_2 = W \circ (w'_k)$. Push two funnels f_1 and f_2 into Q. Go to Step 2.2.

2.2.4. Consider two crossing endpoint sides. If the crossing endpoint side is L (resp. R), create only one funnel: $f_1 = (r, S_1, W_1)$ (resp. $f_2 = (r, S_2, W_2)$), where $S_1 = S \circ (c_1)$ (resp. $S_2 = S \circ (c_2)$) with c_1 (resp. c_2) being the edge in t'_k that is adjacent to $e_k = (p_k, q_k)$ at q_k (resp. p_k). Push f_1 (resp. f_2) into Q. Go to Step 2.2.

3. For each $e_i = (p_i, q_i) \in E$:

3.1. Find the Snell path $SP(p_i, q_i)$ between p_i and q_i that crosses only adjacent edges at p_i and q_i by the function *Find-Path*$(p_i, q_i, S, W, V_D, E_D)$, where S and W are empty. Then, if $D(SP(p_i, q_i)) < E_D[p_i][q_i]$, $E_D[p_i][q_i] = D(SP(p_i, q_i))$.

3.2. If the critical array A_c of e_i is not empty, sort critical points in A_c such that $D(p_i, c_j) < D(p_i, c_{j+1})$, $j \in \{1, \ldots, |Ac| - 1\}$. Then, in E_D, create edges between c_j and c_{j+1} with the edge weights being $D(c_j, c_{j+1})$.

In *Build-D-graph*, at Step 2.1., each vertex $u \in V$ is initiated as roots of funnels that contain only one edge in S, which is the edge opposite to u in triangles adjacent at u. For each funnel $f = (r, S, W)$, we then find the Snell path from the root r to the vertex v, which is opposite to the last edge e_k in S, crossing the edge sequence S and adjacent edges at r and v. The funnels in Q, after that, are spread as follows. Let v_1 and v_2 be two vertices opposite to c_1 and c_2 in two triangles that are adjacent to t'_k at c_1 and c_2, respectively (see Figure 1c). For short, let $S_r^v = \{S_i \mid u \in \{1, \ldots, 4\}\}$ be the set of four edge sequences created by S and adjacent edges at r and v, as presented previously. Similarly, let S_r^1 and S_r^2 be the set of four edge sequences created by ($S \circ (c_1)$ and adjacent edges at r and v_1) and by ($S \circ (c_2)$ and adjacent edges at r and v_2), respectively. It is easy to see that, if there exists the Snell path $SP(r, v)$ crossing the edge sequences in S_r^v, there may exist two Snell paths from r to v_1 and v_2 crossing the edge sequences in S_r^1 and S_r^2, respectively. Thus, at Step 2.2.3., we create two funnels f_1 and f_2 with the same root r and the edge sequences being $S \circ (c_1)$ and $S \circ (c_2)$, respectively. However, if there is no Snell path from r to v crossing the edge sequences in S_r^v, the way we process at Step 2.2.4. is proven to be correct in Proposition 3, which is easily to show.

Proposition 3. *With the funnel $f = (r, S, W)$, if there is no Snell path from r to v crossing the edge sequences in S_r^v:*
1. Among four paths from P_1^* to P_4^*, if a part of these paths have the same crossing endpoint side at L and the re-

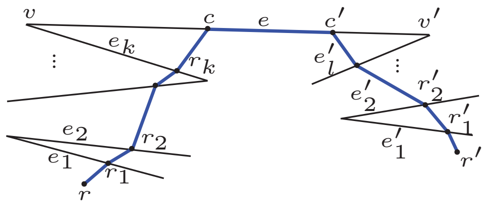


Figure 2: An edge sequence that the sequence arrangement is fail.

maining paths have the same crossing endpoint side at R, there is no Snell path from r to v_1 or from r to v_2 crossing the edge sequences in S^1_r or S^2_r, respectively.

2. *If all four paths from P^*_1 to P^*_4 from r to v have the same crossing endpoint side at L (resp. R), there is no Snell path from r to v_2 (resp. v_1) crossing the edge sequences in S^2_r (resp. S^1_r).*

Finally, we consider the case that an edge sequence cannot satisfy the sequence arrangement. Let $SP(r, r') = (r, r_1, \ldots, r_k, c, c', r'_l, \ldots, r'_1, r')$ be the Snell path between two vertices r and r' crossing the edge sequence $S_{rr'} = (e_1, \ldots, e_k, e, e'_l, \ldots, e'_1)$, where $S_{rr'}$ cannot satisfy the sequence arrangement (see Figure 2). The Snell path $SP(r, r')$ has two critical points c and c' on the edge $e = (v, v')$. We prove that this case will not be missed in our algorithm based on Proposition 4.

Proposition 4. *The Snell path $SP(r, r')$ crossing $S_{rr'}$ exists if and only if $SP(r, v') = (r_1, \ldots, r_k, c, v')$ is the Snell path from r to v' crossing $S = (e_1, \ldots, e_k, e)$, and $SP(r', v) = (r'_1, \ldots, r'_l, c', v)$ is the Snell path from r' to v, crossing $S' = (e'_1, \ldots, e'_l, e)$.*

Proof. This is correct because if each crossing point of $SP(r, v')$ and $SP(r', v)$ satisfies Snell's law, then each crossing point of $SP(r, r')$ also satisfies Snell's law, and vice versa. □

Since S and S' are two edge sequences that can satisfy the sequence arrangement, two paths $SP(r, v')$ and $SP(r', v)$ crossing S and S' with two critical point c and c', respectively, will be found by the function *Find-Path* in our algorithm. At Step 8 of the function *Find-Path*, we store c and c' in the critical array A_c of e, and create two new vertices corresponding to c and c' in the D-graph. Then, we connect (r and c) and (r' and c') together in the D-graph with the connection weights being $D(r, c)$ and $D(r', c')$ according to $SP(r, v')$ and $SP(r', v)$, respectively. After that, in the function *Build-D-graph*, at Step 3.2., we connect c and c' together. By this way, we can assure that, when a shortest path graph algorithm is applied on the D-graph, the path $SP(r, r')$ will be considered.

Experimental results

We have two experimental scenarios, as follows. First, with a given sequence of edges, we would like to ensure that the function *Find-Snell-Path* can return a very-close optimum path. As presented in the introduction, the work of (Carufel et al. 2014) claims that an exact weighted shortest path between two given points crossing an edge sequence cannot be found by using algebraic systems of equations. Hence, we use *quadratic programming* to find the minimum cost paths to compare against our results. Since the exact quadratic programming is NP-hard, we can only run with a small number of edges. However, this result can help us show in practice that the function *Find-Snell-Path* really returns very-close optimum paths. For our second scenario, in the workspace of triangles, we compare our method with one popular decomposition method of (Lanthier, Maheshwari, and Sack 2001), named the *Steiner-Point method*. Although this work is rather old, it seems to be a standard one of the decomposition group. We notice that many newer methods of this group mostly focus on reducing the running time at the expense of the accuracy with respect to the optimum length. However, our target here is the comparison of the path accuracy, and this standard method can assure to supply a high accuracy path when the number of Steiner points on each edge of the workspace is large enough.

We randomly generate all test cases within a plane of 5000×5000 units and the unit weights are in the range $[1, 100]$. All the running times presented in this section are obtained using a system of CPU Intel Core i5, 8G RAM and using Microsoft Visual Studio C++.

Scenario 1: Compare against Quadratic Programming
With a sequence of k edges $S = (e_1, \ldots, e_k)$, where S satisfies the sequence arrangement, the quadratic programming model of finding the minimum weighted path $P(r, v) = (r = r_0, r_1, \ldots, r_k, r_{k+1} = v)$ between r and v crossing S, named the *QP-model*, is as follows:
Variables: A point r_i is created on every edge $e_i \in S$, $i \in \{1, \ldots, k\}$, thus two real variables x_i, y_i, which are the x and y coordinates of r_i, are created, respectively. For each pair of two points r_i and r_{i+1}, $i \in \{1, \ldots, k\}$, a real variable d_i, which corresponds to the Euclidean distance between r_i and r_{i+1}, is created.
Objective function: Minimize $\sum_{i=0}^{k} w_i d_i$
Subject to: For every d_i, $i \in \{0, \ldots, k\}$, $d_i^2 = (x_i - x_{i+1})^2 + (y_i - y_{i+1})^2$. For every $e_i = (p_i, q_i) \in S$, $i \in \{1, \ldots, k\}$, set constraints that r_i must be on the segment (p_i, q_i).

Since the QP-model is non-convex quadratic, which cannot be solved by the basic quadratic programming solvers, we use Couenne[3], an exact solver for the non-convex mixed integer non-linear programming. As we tested, with $k \geq 6$, running the QP-model is extremely time-consuming. Thus, in this scenario, we choose $k = 5$ to show that the results of the function *Find-Snell-Path* and the QP-model are very close together when the Snell path exists between r and v.

We run our method with $\delta = 10^{-5}$. The experiment shows that the average difference between our results and the QP-model results over 500 test cases is **0.01**, where the longest Euclidean distance of edges in S is $\sqrt{2} \cdot 5000$, with the weights being in $[1, 100]$. This result confirms our claim. Furthermore, the average running time of our method is **0.04** seconds while that of the QP-model is **28.44** seconds.

[3]https://projects.coin-or.org/Couenne/

Table 1: Summary of our method and the Steiner-Point method over 600 test cases, within six groups of the number of regions from 5 to 30. The Steiner-Point method is run with m from 6 to 400. The running times are measured in seconds.

Number of regions		5	10	15	20	25	30
Our method's average times		**1.37**	**8.58**	**29.66**	**59.69**	**145.40**	**246.37**
$m = 6$	average times	0.03	0.09	0.18	0.31	0.43	0.58
	ΔD	6152.53	50574.69	160133.55	510206.22	931260.48	1763458.92
$m = 150$	average times	12.63	36.92	72.15	118.27	167.36	233.93
	ΔD	23.93	154.45	522.21	1713.76	3126.83	6768.65
$m = 250$	average times	34.65	101.19	196.84	324.14	463.59	640.30
	ΔD	10.04	53.32	197.66	620.69	1198.45	2478.39
$m = 300$	average times	49.63	146.20	278.48	474.27	678.74	out of memory
	ΔD	7.28	36.09	142.55	468.88	841.60	
$m = 350$	average times	68.61	198.89	376.43	out of memory	out of memory	out of memory
	ΔD	5.17	27.71	103.40			
$m = 400$	average times	89.36	258.16	out of memory	out of memory	out of memory	out of memory
	ΔD	3.62	20.82				

Scenario 2: Compare against the Steiner-Point method With the Steiner-Point method, for each $e_i \in E$, we place m Steiner points evenly along the length of e_i, where m is a positive integer. Let $G_\epsilon = (V_\epsilon, E_\epsilon)$ be a graph, where V_ϵ is the set of vertices and E_ϵ is the set of edges. The set V_ϵ contains all vertices in V and the Steiner points on all edges in E. For each region $t_i \in T$, all vertices and Steiner points on edges of t_i are connected mutually, creating edges in E_ϵ. Let $v, u \in V_\epsilon$, the weight of the connection between v and u in E_ϵ is $w_{uv} \cdot d(v, u)$, where w_{uv} is the unit weight of the region or the edge that both v and u are on. After building G_ϵ, we apply Dijkstra algorithm to find the weighted shortest path between any two vertices in V. Our method is with $\delta = 10^{-5}$. After creating the D-graph, we also use Dijkstra algorithm to find the weighted shortest path between any two vertices in V. We use the standard Dijkstra algorithm in the Boost library for both the Steiner-Point method and our method in this experiment. We randomly create 600 test cases of six groups of the workspaces having 5, 10, 15, 20, 25 and 30 triangles (100 test cases for each group). For each test case, we find the weighted shortest paths of all pairs of vertices, by our method and the Steiner-Point method with $6 \leq m \leq 400$. Let D and D' be the weighted length sums of all the result paths from our method and the Steiner-Point method per test case, respectively. We denote the average weighted length difference per test case between our method and the Steiner-point method as $\Delta D = D' - D$. Table 1 summarizes our comparison.

Among all 600 test cases, there is no test case that the weighted length of our method is greater than that of the Steiner-Point method, even when the number of Steiner points per each edge of the Steiner-Point method grows up to 400. Additionally, the running times are significantly shorter with our method. First, we test with $m = 6$ since the work of (Lanthier, Maheshwari, and Sack 2001) recommends this value. However, as in Table 1, the average weighted length of the Steiner-Point method with $m = 6$ in each group is too far away from our result. When m is increased, up to 400, with the cases of five and ten regions, obviously, the average weighted lengths of the Steiner-Point method come close to our results. However, with five regions, our running time is only 1.37 seconds while the Steiner-Point method running time is 89.36 seconds, a speed up of around 65 times. Similarly, with ten regions, we are around 30 times faster. For the remaining cases, when the number of regions and m are large, the Steiner-Point method runs out of memory (larger than 1.7 GB). We present one of our test cases with ten regions in Figure 3.

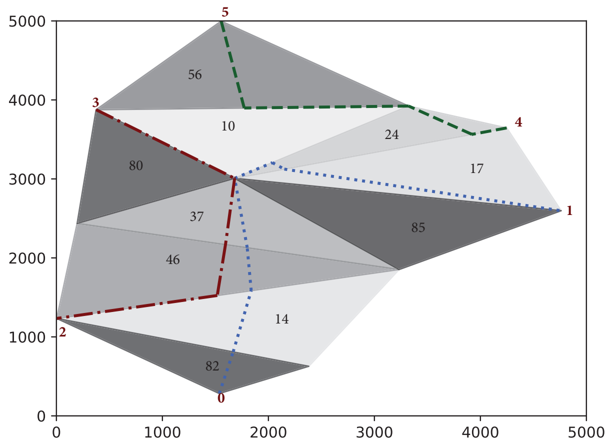


Figure 3: One test case with 10 regions, 20 region edges and 11 vertices, showing three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

Conclusion

We have presented experiments for WRP with $\delta = 10^{-5}$. If we change δ, the closeness of the optimal path lengths in our method to the exact solution depends on δ. As δ decreases, the difference between the lengths of these two paths becomes smaller. The value of δ, accordingly, affects the running time of our method only in a logarithmic function (see Lemma 1). We also note that, to find the weighted shortest path between two points that are not vertices of the workspace, one can easily make these two points vertices by adding new triangles. For the future work, we will improve the running time of building the D-graph by using heuristics and extend it to the three-dimensional workspaces.

References

Bose, P.; Maheshwari, A.; Shu, C.; and Wuhrer, S. 2011. A survey of geodesic paths on 3d surfaces. *Computational Geometry* 44(9):486–498.

Carufel, J.-L. D.; Grimm, C.; Maheshwari, A.; Owen, M.; and Smid, M. 2014. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry* 47(7):724–727.

Gheibi, A.; Maheshwari, A.; Sack, J.-R.; and Scheffer, C. 2018. Path refinement in weighted regions. *Algorithmica* 80(12):3766–3802.

Goerzen, C.; Kong, Z.; and Mettler, B. 2009. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems* 57(1):65.

Jaklin, N.; Tibboel, M.; and Geraerts, R. 2014. Computing high-quality paths in weighted regions. In *Proceedings of the Seventh International Conference on Motion in Games*, MIG '14, 77–86. New York, NY, USA: ACM.

Kindl, M. R., and Rowe, N. C. 2012. Evaluating simulated annealing for the weighted-region path-planning problem. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, 926–931.

Lanthier, M.; Maheshwari, A.; and Sack, J.-R. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* 30(4):527–562.

Mitchell, J. S. B., and Papadimitriou, C. H. 1991. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM* 38(1):18–73.

Mitchell, J. S. B. 2017. Chapter 31: Shortest paths and network. In Toth, C. D.; O'Rourke, J.; and Goodman, J. E., eds., *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC.

Rowe, N. C., and Alexander, R. S. 2000. Finding optimal-path maps for path planning across weighted regions. *The International Journal of Robotics Research* 19(2):83–95.

Rowe, N. C., and Richbourg, R. 1990. An efficient snell's law method for optimal-path planning across multiple two-dimensional, irregular, homogeneous-cost regions. *The International Journal of Robotics Research* 9(6):48–66.

Szczerba, R. J.; Chen, D. Z.; and John J. Uhran, J. 1998. Planning shortest paths among 2d and 3d weighted regions using framed-subspaces. *The International Journal of Robotics Research* 17(5):531–546.

Xidias, E. K. 2019. On designing near-optimum paths on weighted regions for an intelligent vehicle. *International Journal of Intelligent Transportation Systems Research* 17(2):89–101.

Zheng, X.; Koenig, S.; Kempe, D.; and Jain, S. 2010. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics* 26(6):1018–1031.