Computing Close to Optimal Weighted Shortest Paths in Practice

30th International Conference on Automated Planning and Scheduling (ICAPS 2020)

Nguyet Tran [1], **Michael J. Dinneen**, **Simone Linz**

School of Computer Science, University of Auckland, New Zealand

October 29-30, 2020

1. Email: ntra770@aucklanduni.ac.nz

Contents

Introduction

Problem Definition

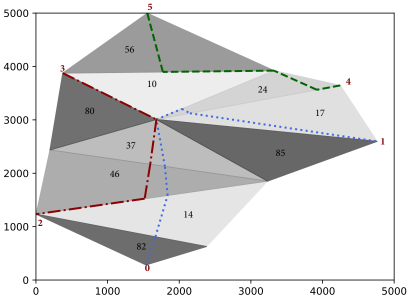

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of non-overlapping *regions*
- E: the set of *edges*
- V: the set of *vertices*

- $WS = (T, E, V)$: a continuous two-dimensional workspace
- Each region $t_i \in T$ is a triangle, and assigned a unit *weight* (or cost) $w_i > 0$.
- Let p and q be two points on a region $t_i \in T$,
  - $d(p, q)$: the Euclidean distance between p and q
  - $D(p, q) = w \cdot d(p, q)$: the *weighted length* (or cost) between p and q, where w is the unit weight of t_i or the edge that the segment (p, q) is on.

Introduction

Problem Definition

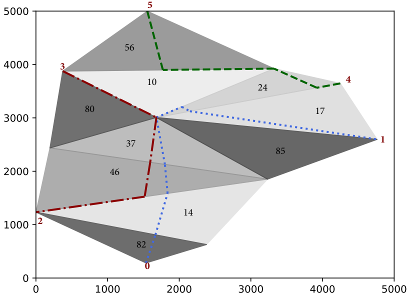

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of non-overlapping *regions*
- E: the set of *edges*
- V: the set of *vertices*

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 length

$$D(P^*(u, v)) = \sum_{i=0}^{k} D(o_i, o_{i+1}) \text{ is minimum,}$$

where every o_i, $i \in \{1, \ldots, k\}$, called a *crossing point*, can be a point on an edge in E or a vertex in V.

Introduction

Difficulties

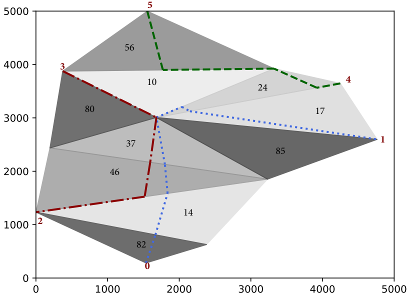

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of *regions*
- E: the set of *edges*
- V: the set of *vertices*

- The problem is NP-hard or not: Unknown
- Currently, there is no known polynomial or exponential time algorithm for finding the exact weighted shortest path.
- The exiting algorithms to solve WRP are all approximations.

Introduction

Existing approaches

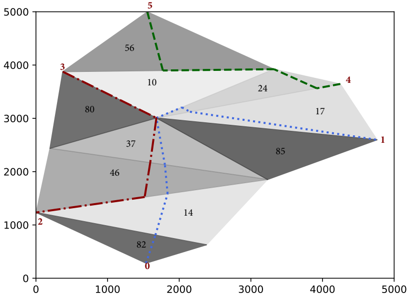

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of *regions*
- E: the set of *edges*
- V: the set of *vertices*

An overview of the existing approaches:

- Exploiting Snell's law (impractical solutions)
- Using heuristic methods (unpredictable results)
- Applying decomposition ideas, with a grid of cells or a graph of discrete points, called Steiner-points (time-consuming for a close optimal result).

Introduction

Our approach

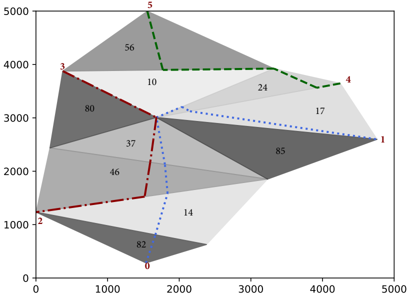

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of *regions*
- E: the set of *edges*
- V: the set of *vertices*

An overview of the existing approaches:

- Exploiting Snell's law
  (impractical solutions →practical solution)
- Using heuristic methods
  (unpredictable results)
- Applying decomposition ideas, with a grid of cells or a graph of discrete points, called Steiner-points
  (time-consuming for a close optimal result).

Proposed method

1. Weighted shortest path crossing an edge sequence S

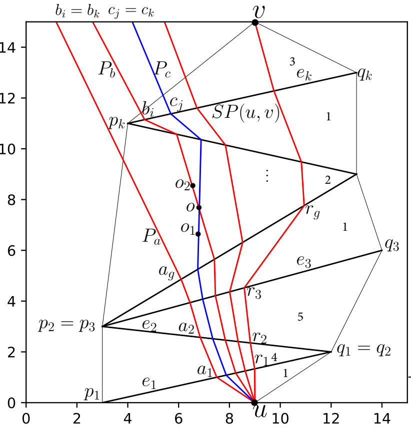

FIGURE – Illustration of Snell rays.

- Given an ordered sequence of k edges $S = (e_1, e_2, \ldots, e_k)$, where three consecutive edges in S cannot be in the same triangles [a].
- $W = (w_0, \ldots, w_k)$ is 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 $e_{k+1} = (v, v)$.
- $P(u, v) = (u = r_0, r_1, \ldots, r_k, r_{k+1} = v)$ is a path between two vertices $u, v \in V$, crossing an edge sequence, where r_i is on e_i with every $i \in \{1, \ldots, k\}$.

a. Otherwise, we present how to process it in the paper.

Proposed method

1. Weighted shortest path crossing an edge sequence S

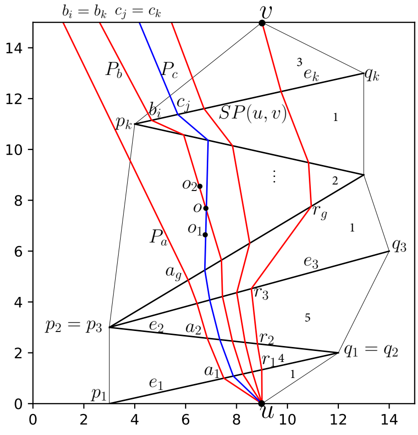

FIGURE – Illustration of Snell rays.

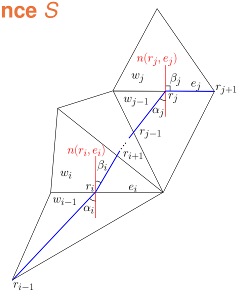

FIGURE – Illustration of Snell's law.

Snell's law:
$P(u, v)$ has the minimum weighted length crossing S if and only if at every crossing point r_i on e_i, for which r_i is not an endpoint of e_i, the following condition holds:

$$w_{i-1} \sin \alpha_i = w_i \sin \beta_i$$

Proposed method

1. Weighted shortest path crossing an edge sequence S

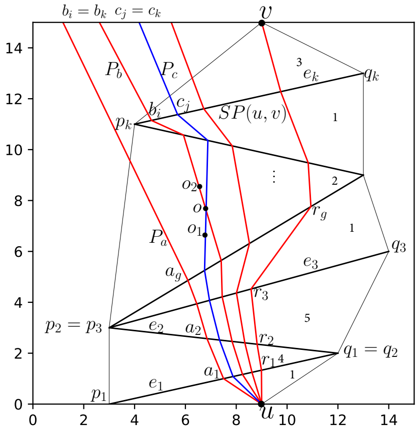

FIGURE – Illustration of Snell rays.

Snell ray:

- Let a_1 be a point on $e_1 \in S$.
- Apply 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 $1 \leq g \leq k$ 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, crossing S, from e_1 to e_g.

Snell path: $P(u, v)$ is a Snell path if

- Every point r_i, $i \in \{1, \ldots, k\}$, is on the interior of e_i, which cannot be one of the the two endpoints of e_i, and
- Snell's law is obeyed at every r_i.

Proposed method

1. Weighted shortest path crossing an edge sequence S

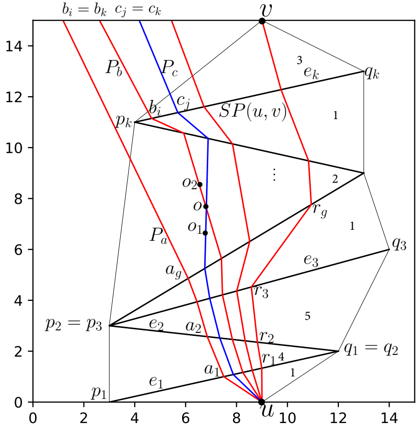

FIGURE – Illustration of Snell rays.

- Two Snell rays P_b and P_c:
  $P_b = (u, b_1, \ldots, b_i, \mathcal{R}_i^b)$
  $P_c = (u, c_1, \ldots, c_j, \mathcal{R}_j^c)$, where $b_1 \neq c_1$
- P_b and P_c cannot intersect each other.

Proposed method

1. Weighted shortest path crossing an edge sequence S

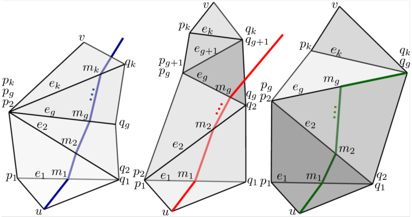

FIGURE – Illustration of P_m.

Finding the Snell path $P(u, v)$ (approximately):

- From the middle point m_1 of e_1, create a Snell ray $P_m = (u, m_1, \ldots, R_g^m)$ crossing S.
- If e_{g+1}, where $e_{k+1} = (v, v)$, is on the left (resp. right) of R_g^m, then the Snell ray that hits v must cross only the parts from p_i to m_i (resp. from m_i to q_i). Thus, we trim $e_i = (p_i, q_i)$ to (p_i, m_i) (resp. (m_i, q_i)).
- This process is iterated until P_m hits v, or all edges in S are trimmed such that $d(p_i, q_i) < \delta$, where δ be an extremely small value.
- Here, we employ the idea from the function *Find-Point* in the work of (Mitchell and Papadimitriou 1991).
  However, *Find-Snell-Path* is different from *Find-Point* by that, if the Snell path crosses an endpoint of any original e_i in S, *Find-Snell-Path* will be stopped while *Find-Point* will still continue the process.

Proposed method

2. Main algorithm

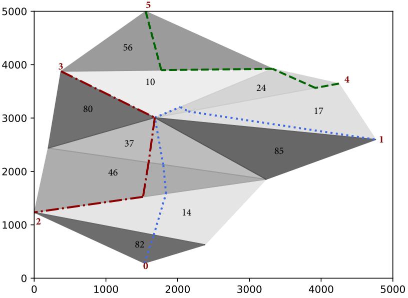

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of *regions*
- E: the set of *edges*
- V: the set of *vertices*

D-graph: an undirected graph (V_D, E_D), where

- $V_D = V \cup V_c$ with V_c being the set of critical points [a].
- An edge in E_D between two points u and v in V_D is created if there exists a Snell path between u and v, **which only cross the interiors of the edges in E.**
- The weight of every edge between u and v in E_D is the minimum weighted length among all possible Snell paths between u and v.

a. For V_c, please see in the paper.

Proposed method

2. Main algorithm

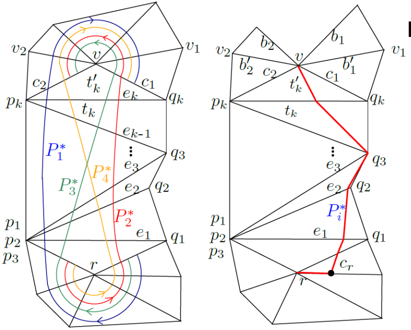

FIGURE – Illustration of a funnel.

Funnel: $f = (r, S, W)$, where

- $S = (e_1, \ldots, e_k)$, $W = (w_0, \ldots, w_{k-1})$
- $r \in V$ is the root of f
- The last edge $e_k \in S$ is the bottom of f.

Notes:

- The Snell path from r to v crossing S can go around the adjacent edges at r and v with critical points (at most four possible paths, P_1^* to P_4^*).
- We present in the paper how to avoid finding all of these four Snell paths.
- After finding the Snell path from r to v, let $S_1 = S \circ (c_1)$, $S_2 = S \circ (c_2)$ [a], and W_1 and W_2 be two weight lists with respect to S_1 and S_2, respectively. One of the following three conditions holds:
  - Two new funnels $f_1 = (r, S_1, W_1)$ and $f_2 = (r, S_2, W_2)$ are created.
  - Only one new funnel $f_1 = (r, S_1, W_1)$ or $f_2 = (r, S_2, W_2)$ is created.
  - No new funnel is created.

a. Appending c_1 or c_2 to the end of S.

Proposed method

2. Main algorithm

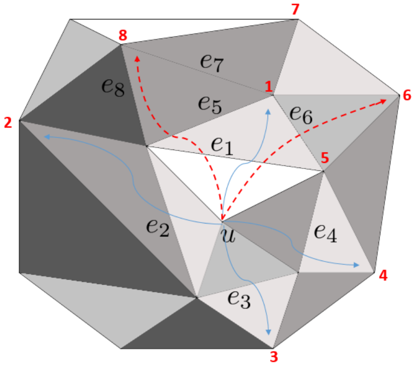

FIGURE – Illustration of a funnel.

Main idea:

- Building a D-graph for $WS = (T, E, V)$.
- Applying a shortest path graph algorithm on the D-graph to find the weighted shortest path between any pair of vertices.

Building D-graph:

- Using a queue Q.
- For each vertex $u \in V$, initializing funnels $f = (r, S, W)$, where $r = u$ and S contains only one edge opposite to u.
- Pushing the funnels into Q.
- Popping one funnel $f = (r, S, W)$ out Q, 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 S.
- If the Snell path between r and v crossing S exists, updating the D-graph.
- Then, the new funnels (at most two) corresponding to f are created and pushed into Q.
- The process will be stopped when Q is empty.

Note: In practice, finding a Snell path will easily cross a vertex in V and stop. Thus, not too many funnels will be created.

Experimental results

- **Scenario 1: Compare against Quadratic Programming**
- **Scenario 2: Compare against the Steiner-Point method** (Lanthier, Maheshwari, and Sack 2001)

Experimental results

Scenario 2: Compare against the Steiner-Point method

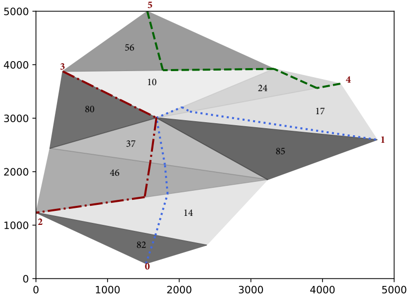

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- T: the set of *regions*
- E: the set of *edges*
- V: the set of *vertices*

The Steiner-Point method:

- For each $e_i \in E$, m discrete points, called Steiner points, are placed evenly along the length of e_i.
- Let $G_m = (V_m, E_m)$ be a graph, where V_m contains all the Steiner points and the vertices in V, and E_m be the set of connections.
- For each region $t_i \in F$, the three vertices and the Steiner points on the edges of t_i are connected mutually, creating connections in E_m.
- Let $v, u \in V_m$, the weight of the connection between v and u in E_m 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_m, we apply Dijkstra algorithm to find the weighted shortest path between any two vertices in V.

Experimental results

Scenario 2: Compare against the Steiner-Point method

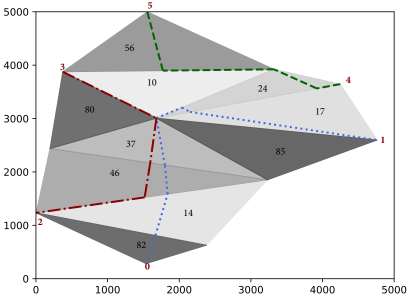

FIGURE – An example of WRP problem with 10 regions and three very-close optimum paths between vertices (0 and 1), (2 and 3) and (4 and 5).

- *T*: the set of *regions*
- *E*: the set of *edges*
- *V*: the set of *vertices*

Our method:

- $\delta = 10^{-5}$
- After creating the D-graph, we also use Dijkstra algorithm to find the weighted shortest path between two vertices.

Experimental results

Scenario 2: Compare against the Steiner-Point method

Table 1. With the Release mode of Visual Studio C++[a].

Number of regions		5	10	15	20	25	30
Our method's average times		**0.02**	**0.11**	**0.37**	**0.75**	**1.82**	**3.03**
$m = 6$	average times	0.00036	0.00083	0.0023	0.0029	0.0036	0.0044
	$\%D$	0.29%	0.69%	1.07%	2.02%	2.45%	3.10%
$m = 150$	average times	0.17	0.42	0.78	1.32	1.76	2.52
	$\%D$	0.00098%	0.0021%	0.0035%	0.0069%	0.0086%	0.013%
$m = 250$	average times	0.44	1.33	2.29	3.63	4.87	7.07
	$\%D$	0.00039%	0.00074%	0.0013%	0.0025%	0.0033%	0.0046%
$m = 300$	average times	0.63	1.73	3.60	6.21	7.04	9.79
	$\%D$	0.00028%	0.00050%	0.00095%	0.0019%	0.0024%	0.0031%
$m = 350$	average times	0.86	2.40	4.43	6.82	9.57	13.05
	ΔD	0.00021%	0.00038%	0.00069%	0.0014%	0.0017%	0.0024%
$m = 400$	average times	1.13	3.33	5.80	8.94	12.64	17.32
	$\%D$	0.00015%	0.00029%	0.00056%	0.0010%	0.0013%	0.0018%

Let D and D' be the weighted length sums of all the result paths of our method and the Steiner-Point method per test case, respectively.

- $\Delta D = D' - D$
- $\%D = \dfrac{\Delta D}{(D + D')/2}$

Results:

- Our results are always shorter in weighted length.
- Our running times are faster in case a close to an optimal path is needed.

a. We note that, Table 1 in the paper is with the Debug mode of Visual Studio C++, where we can limit and stop the case that occupies larger than 1.7 GB of memory. We have already updated the experimental results to that the test cases are run in the Release mode of Visual Studio C++ (see Table 1 above), and the memory is not constrained.

THANK YOU