Conical Routing For Lookahead

Gaurish Telang

gaurish108@gmail.com

March 7, 2022
12:45 Noon

Contents

1 Introduction

We report some results on schemes in computing paths with lookahead in the stalactite and stalagmite setting. In a path with lookahead any point on the path can see a large chunk of the path immediately ahead of it. [1] More precisely, given a parameter $l \in \mathbb{R}^+$ we want to compute a shortest path between s and t such that point on the computed path satisfies one of either of the two following properties

- The point can see at least a subpath of length $\geq l$ immediately ahead of it
- The part of the subpath that the point can see ahead of it contains t

We will give a set of schemes for doing this in the alternating *stalactites and stalagmites* — henceforth shortened to "*ites" — such that the shortest path between two given points s and t is taut against the tips of the "*ites".

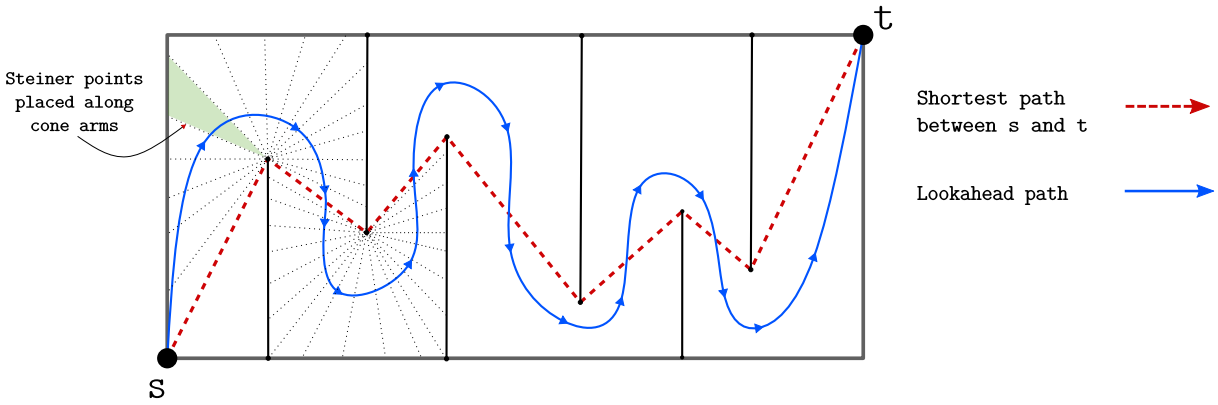

Figure 1: Conical Routing between start s and destination t

[1] we are using euclidean vision where two points A and B see one another iff the straight line segment AB is contained in the closure of the free space.

2 Basic scheme

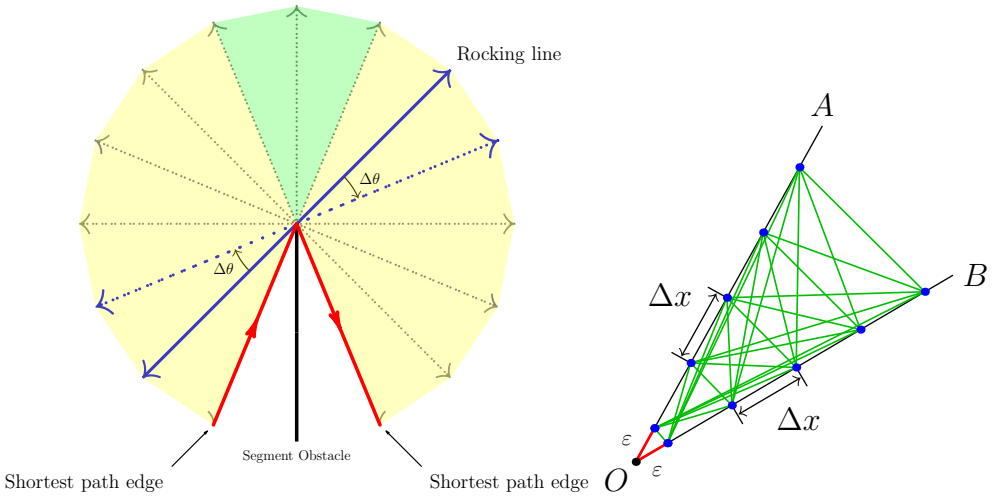

Figure 2: (Left) A rocking line (blue) creates a sequence of cones of angles $\Delta\theta$ between two successive shortest path edges. (Right) Complete bipartite graph (green) between points on two arms of a cone on the Steiner points. Distance between two consecutive Steiner points along each arm is Δx. O is the tip of the stalactite/stalagmite. See **??** for an instance of the discretization of a domain done using these rules.

Algorithm

1. At each tip T of the *ites, consider the two shortest path edge between s and t incident at that point. Divide the angle between them into yellow and green wedges as shown in the left hand side Figure 2. Each yellow wedge is divided into cones of small angle $\Delta\theta$ whereas the green wedge is divided into small cones again of size $\Delta\theta$, allowing for one of the cones to have angle $\leq \Delta\theta$

2. Place Steiner points separated by a small distance Δx starting from the obstacle segment tip along each arm of every cone, for as long as the cone arm lies inside free space. Also add the endpoints of the segments as Steiner points of those respective segments. [2]

3. Draw the complete bipartite graph between the Steiner points of each cone as shown in Figure 2

4. Label the cones as $C_0, C_2, \ldots C_{n-1}$ in order from the cone containing s to the one containing t. Let the first arms of each cone (as encountered in going from s to t be named A_i).

5. Solve the following integer program.

$$\min \sum_{e \in E} c_e x_e \tag{1}$$

such that

[2] computing these is just a simple matter of ray-shooting.

$$x_e \in \{0,1\} \tag{2}$$

$$\sum_{e \in C_i} x_e = 1 \qquad 0 \le i \le n-1 \tag{3}$$

$$\sum_{u \in A_i} x_{u,v} = \sum_{w \in A_{i+1}} x_{v,w} \qquad \forall v \in A_i, \quad 1 \le i \le n-1 \tag{4}$$

$$\sum_{v \in A_0} x_{s,v} = 1 \tag{5}$$

$$\sum_{u \in A_{n-1}} x_{u,t} = 1 \tag{6}$$

$$\sum_{e \in H_{j,k}} x_e \ge \ell \qquad 1 \le j \le \lceil \frac{\Theta_k}{\Delta\theta} \rceil \tag{7}$$

Notes

- Equation 3 are constraints that force the choice of exactly one edge per cone.
- Equation 4 are flow constraints that make the selected set of edges form a connected path.
- Equation 5 and Equation 6 force s and t to be terminal nodes of the computed path.
- Equation 7 enforces the lookahead constraint around each *ite tip, Here, Θ_k is the angle between the edges of the shortest path incident at the k^{th} *ite tip such that the angle does not contain the *ite. Around each *ite tip there is a sequence of halfplanes. We globally label these halfplanes with a 2d index set $\{j,k\}$ where j refers to the rank of the half-plane within a sequence at the k^{th} *ite tip.

This scheme has been implemented in Python `2.7.12` using the Python bindings of the CGAL `4.3.0-4` kernel on a Thinkpad laptop running Linux Mint. It is available at `https://github.com/gtelang/conicalrouting`. For solving linear and integer programs, I have used the PuLP library available at `https://coin-or.github.io/pulp/`. For graph algorithms I use NetworkX.

Figure 3 and Figure 4 gives an example of the output of running the above program. The lookahead imposed on both curves is $\ell = 0.5$. The discretization of the domain with cones and the corresponding bipartite graphs is shown in light semi-transparent gray. The shortest path is colored red, while the lookahead path is colored green. Note that the shortest path has been computed using Dijkstra's algorithm in the graph. The exact shortest path between the two red nodes in the continuous domain in both cases is almost the same as the one computed here.

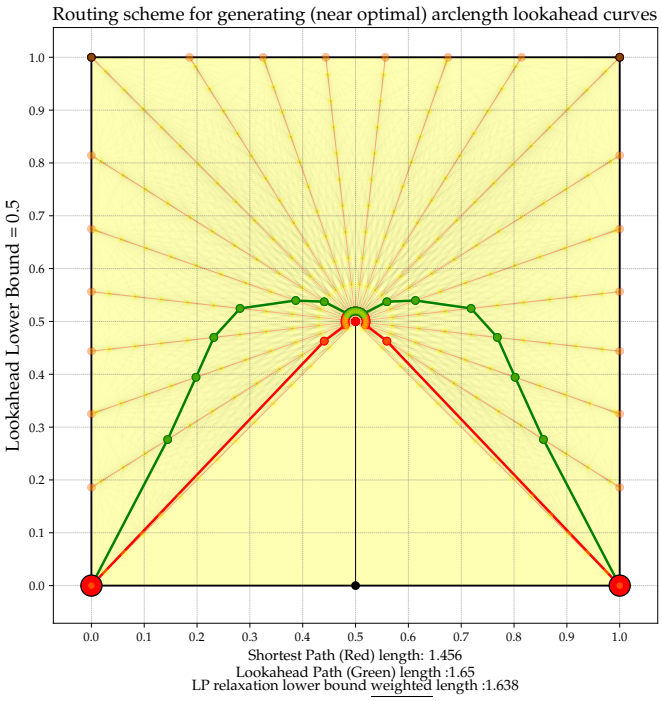

Figure 3: An example of a path generated by solving the above program

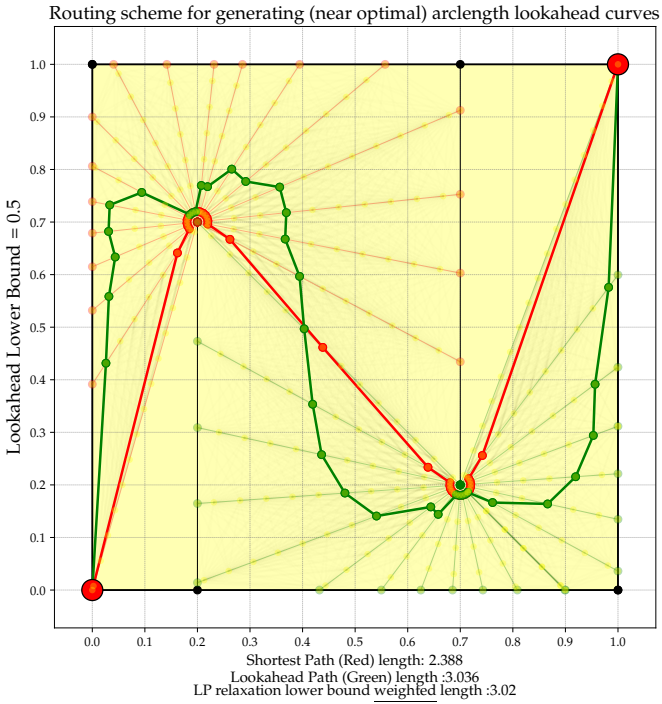

Figure 4: An example of a path generated by solving the above program

A simple graph theoretic problem can be abstracted from this. See Figure 5

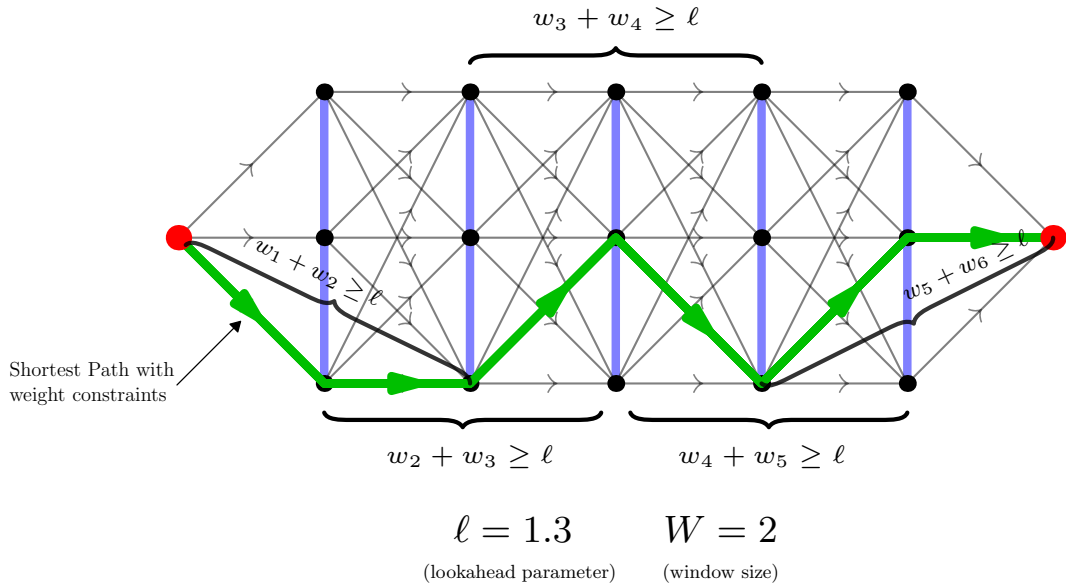

Figure 5: Instance of the (l, W)-multipartite lookahead problem. $W \in \mathbb{N}$ is the window size and $l \in \mathbb{R}^+ \cup \{0\}$ is the lookahead parameter. In this problem we compute need to compute the shortest path in a weighted multipartite graph with positive edge weights such that the sum of consecutive W edges have weight $\geq l$. The node groups of the multipartition come sequentially one after the other. Notice that $\ell = 0$ for any $W \in \mathbb{N}$ is the usual shortest path problem in such a graph.