Journal of Graph Algorithms and Applications
http://jgaa.info/
vol. 7, no. 3, pp. 287–303 (2003)

Finding Shortest Paths With Computational Geometry

Po-Shen Loh
California Institute of Technology
http://www.caltech.edu/
po@caltech.edu

Abstract

We present a heuristic search algorithm for the $\mathbb{R}^d$ Manhattan shortest path problem that achieves front-to-front bidirectionality in subquadratic time. In the study of bidirectional search algorithms, front-to-front heuristic computations were thought to be prohibitively expensive (at least quadratic time complexity); our algorithm runs in $O(n \log^d n)$ time and $O(n \log^{d-1} n)$ space, where n is the number of visited vertices. We achieve this result by embedding the problem in $\mathbb{R}^{d+1}$ and identifying heuristic calculations as instances of a dynamic closest-point problem, to which we then apply methods from computational geometry.

Communicated by Joseph S.B. Mitchell: submitted October 2002; revised June 2003.

Research supported by Axline and Larson fellowships from the California Institute of Technology.

1 Introduction

Let $\mathcal{E}$ be a finite set of axis-parallel line segments in $\mathbb{R}^d$ intersecting only at endpoints, let $\mathcal{V}$ be the set of all endpoints in $\mathcal{E}$, and let $\mathcal{G}$ be the graph defined by $\mathcal{V}$ and $\mathcal{E}$. The Manhattan shortest path problem is to find the shortest path in $\mathcal{G}$ between two distinct terminals $T_1, T_2 \in \mathcal{V}$. We will assume that the graph has already been processed and stored in memory; our algorithm is meant for applications in which multiple runs will be conducted on the same graph.

1.1 Notation

In this paper, we will use the following conventions.

- The symbol XY denotes the Manhattan distance between two points X and Y in space.
- The symbol $\overline{XY}$ denotes the edge connecting two adjacent vertices $X, Y \in \mathcal{G}$.
- The symbol $\mathbf{XY}$ denotes the length of a shortest path in $\mathcal{G}$ between two vertices $X, Y \in \mathcal{V}$.
- The symbol $\mathcal{S}(\mathbf{XY})$ denotes the set of all shortest paths in $\mathcal{G}$ between two vertices $X, Y \in \mathcal{V}$.
- The function $l(\mathcal{P})$ denotes the length of a path $\mathcal{P}$ in $\mathcal{G}$. A path is a set of vertices and edges that trace a route between two vertices in a graph.
- The direct sum $\mathcal{S}(\mathbf{XY}) \oplus \mathcal{S}(\mathbf{YZ})$ forms all pairwise concatenations between paths in $\mathcal{S}(\mathbf{XY})$ and $\mathcal{S}(\mathbf{YZ})$. That is,

$$\mathcal{S}(\mathbf{XY}) \oplus \mathcal{S}(\mathbf{YZ}) = \{\mathcal{P} \cup \mathcal{Q} : \mathcal{P} \in \mathcal{S}(\mathbf{XY}), \mathcal{Q} \in \mathcal{S}(\mathbf{YZ})\}.$$

1.2 Previous Work

Dijkstra's algorithm solves this problem in $O(n \log n + dn)$ time, where n is the number of vertices visited by an algorithm[1], which is a less expensive function of n than that of our algorithm. However, the purpose of our algorithm's search heuristic is to reduce n itself, thereby recapturing the additional log-power complexity factor.

Our algorithm is an extension of *A* heuristic search* [4, 5], which is a priority-first search in $\mathcal{G}$ from T_1 to T_2. A* maintains two dynamic point sets, a *wavefront* $\mathcal{W} \subseteq \mathcal{V}$ and a set $\Omega \subseteq \mathcal{V}$ of *visited* points, which begin as $\mathcal{W} = \{T_1\}$ and $\Omega = \varnothing$. The priority of a vertex V is a conservative estimate of the length of a shortest path connecting the terminals via V; specifically:

$$\text{priority}(V) = \text{“}\mathbf{T_1V}\text{”} + VT_2,$$

[1] Note: we do not define n to be the number of edges on the shortest path.

where "$\mathbf{T_1T_1}$" $= 0$,

$$\text{“}\mathbf{T_1V}\text{”} = \min_{U \in \mathcal{A}} \{\mathbf{T_1U} + UV\}, \tag{1}$$

and $\mathcal{A}$ is the set of all visited vertices that are adjacent to V. At each iteration, A* *visits* a point $P \in \mathcal{W}$ of minimal priority: it transfers P from $\mathcal{W}$ to Ω and inserts P's unvisited neighbors into $\mathcal{W}$. This technique is called *wave-propagation* because the wavefront $\mathcal{W}$ grows like a shock wave emitted from T_1.

The proofs of our algorithm's validity in Section 3 will show that "$\mathbf{T_1V}$" approximates $\mathbf{T_1V}$ and is in fact equal to $\mathbf{T_1V}$ by the time V is visited, hence the quotation marks. Once T_2 is visited, the length of the shortest path is then priority(T_2). To recover the shortest path, *predecessor* information is stored: the predecessor of a point V is defined to be the point $V^* \in \mathcal{A}$ that yields the minimum in equation (1). If there are multiple minimal points, then one of them is arbitrarily designated as the predecessor. Since $\mathcal{A}$ is a dynamic set, the predecessor of a point may vary as A* progresses, but it turns out that once a point P has been visited, its predecessor is always adjacent to it along some path in $\mathcal{S}(\mathbf{T_1P})$. Thus, after T_2 is visited, a shortest path from T_1 to T_2 can be obtained by tracing back through the predecessors from T_2 to T_1.

The purpose of a search heuristic is to help the wavefront grow in the direction of T_2. When no shortest paths from T_1 to T_2 approach T_2 directly, however, unidirectional search heuristics become misleading. This is not a rare occurrence; in computer chip design, one may wish to route wires between terminals that are accessible only through certain indirect channels. In order to develop a more intelligent heuristic, one must therefore explore the search space from both terminals. This motivates bidirectional search.

We concentrate here on bidirectional searches that are based on wave propagation, which expand one wavefront from each terminal. Such searches come in two types: *front-to-end* and *front-to-front*. In front-to-end searches, points are assigned heuristics without regard to information about points on the other wavefront. In contrast, the wavefronts cooperate in front-to-front searches; heuristic calculations for points on a given wavefront incorporate information from points on the opposing wavefront. This is illustrated in Figure 1.

The paper [6] by Kaindl and Kainz surveys many approaches to bidirectional search, and advocates front-to-end heuristics because of the apparently prohibitive computational complexity required for front-to-front calculations. In particular, the fastest such heuristic ran in time proportional to wavefront size, which would yield a worst-case running time that was at least quadratic. In this paper, we present a Manhattan shortest path algorithm that attains front-to-front bidirectionality for only a nominal log-power complexity cost over front-to-end search.

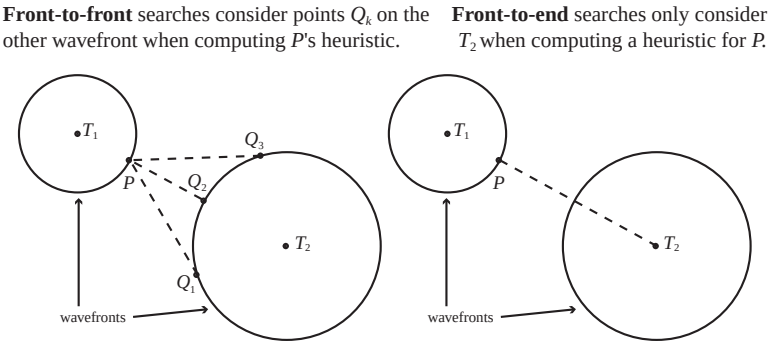

Figure 1: Two varieties of bidirectional heuristics.

1.3 Front-to-Front Bidirectionality

The natural front-to-front generalization of A* uses a two-parameter search heuristic that we shall call the *estimated total length*, or *ETL* for short. For any two points $P, Q \in \mathcal{G}$, the ETL function λ is defined as follows:

$$\lambda(P, Q) = \text{“}\mathbf{T_1P}\text{”} + PQ + \text{“}\mathbf{QT_2}\text{”},$$

where “$\mathbf{T_1P}$” and “$\mathbf{QT_2}$” are defined along the lines of equation (1). This search algorithm, which we shall call *FF* (Front-to-Front), maintains two wavefronts, $\mathcal{W}_1$ and $\mathcal{W}_2$, and two visited sets, Ω_1 and Ω_2. Initially, $\mathcal{W}_1 = \{T_1\}$, $\mathcal{W}_2 = \{T_2\}$, and $\Omega_1 = \Omega_2 = \varnothing$, and at each iteration, a pair of vertices $P_1 \in \mathcal{W}_1$ and $P_2 \in \mathcal{W}_2$ with minimal $\lambda(P_1, P_2)$ is visited just as in A*. If several pairs tie with minimal ETL, then FF visits a pair with minimal P_1P_2 among those tied. This helps to steer the wavefronts toward each other.

FF's termination condition is quite delicate, as it is possible for the wavefronts to intersect before a shortest path is found. Since our algorithm has a similar termination condition, we postpone the details until we present our algorithm in Figure 2.

2 Proposed Algorithm

Since FF finds the minimal $\lambda(P_1, P_2)$ at each iteration, it computes an accurate but costly heuristic. Our algorithm, which we name *FFF* (Fast Front-to-Front), accelerates FF by relaxing the minimality constraint on $\lambda(P_1, P_2)$. To motivate our approximation, we embed the wavefronts into $\mathbb{R}^{d+1}$ and apply computational geometry.

2.1 Dimensional Augmentation

The first step is to identify our ETL minimization as a *dynamic bichromatic closest-pair* problem. To accomplish this, we enter $\mathbb{R}^{d+1}$ space, embedding our points $P \in \mathcal{W}_1 \cup \mathcal{W}_2$ as follows:

$$P(x_1, x_2, \ldots, x_d) \mapsto \begin{cases} P'(x_1, x_2, \ldots, x_d, +\sigma_1), & \text{if } P \in \mathcal{W}_1, \\ P'(x_1, x_2, \ldots, x_d, 0), & \text{if } P \text{ is } T_1 \text{ or } T_2, \\ P'(x_1, x_2, \ldots, x_d, -\sigma_2), & \text{if } P \in \mathcal{W}_2, \end{cases}$$

where

$$\sigma_k = \min_{V \in \mathcal{A}} \{\mathbf{T_k V} + VP\},$$

and $\mathcal{A}$ is the set of vertices in Ω_k that are adjacent to P.

In this paper, we will use the convention that all primed points are in $\mathbb{R}^{d+1}$, all unprimed points are in $\mathbb{R}^d$, and if two points or sets in the same context have the same letter, a prime indicates passage between $\mathbb{R}^d$ and $\mathbb{R}^{d+1}$. If some point $P' \in \mathbb{R}^{d+1}$ is not in either embedded wavefront, then its unprimed form P will still represent the projection of P' onto its first d coordinates. Finally, we will refer to the last coordinate of a point $P' \in \mathbb{R}^{d+1}$ by the shorthand $\sigma(P')$.

2.2 Identification and Approximation

Since we are working in a Manhattan metric, $\lambda(P_1, P_2)$ is equal to the distance between P_1' and P_2' in $\mathbb{R}^{d+1}$. Therefore, finding a minimal pair is equivalent to finding a closest pair between $\mathcal{W}_1'$ and $\mathcal{W}_2'$ in $\mathbb{R}^{d+1}$. At each iteration of FF, the point sets only change slightly, so each minimization is an instance of a dynamic bichromatic closest-pair problem.

Using the method of Eppstein in [3], one can solve the $\mathbb{R}^{d+1}$ Manhattan dynamic bichromatic closest-pair problem with $O(\log^{d+2} n)$ amortized time per insertion and $O(\log^{d+3} n)$ amortized time per deletion. In each of our iterations, we will need to perform at least one insertion and one deletion, so that method would give us an algorithm that ran in $O(n \log^{d+3} n)$ amortized time.

We are only looking for a search heuristic, though, so we can content ourselves with *almost-closest pairs* without compromising the validity of our algorithm. This in fact speeds up our algorithm by a factor of $\log^3 n$ and changes the complexity bounds from amortized to worst-case; in low dimensions ($d \leq 3$), it provides a significant performance boost. To accomplish this, we take a cue from the method of successive approximations: starting from a well-chosen seed point $P_0' \in \mathbb{R}^{d+1}$, not necessarily in $\mathcal{W}_1'$ or $\mathcal{W}_2'$, we find a closest point $X_1' \in \mathcal{W}_1'$, and then find an $X_2' \in \mathcal{W}_2'$ that is closest to X_1'. We then use X_1 and X_2 instead of the P_1 and P_2 of FF. This pair may not have minimal λ, but it turns out to be close enough (cf. Theorems 4 and 5). Therefore, we only need *dynamic closest-point search*.

2.3 Dynamic Closest-Point Search

This problem can be reduced to that of *dynamic range search for minimum* (discussed in [7, 8, 10]) by adapting the technique [1] of Gabow, Bentley, and Tarjan in [9]. Let $\mathcal{S}'$ be a set of points in $\mathbb{R}^{d+1}$, let m and M be the respective minimum and maximum $(d+1)$-st coordinates in $\mathcal{S}'$, and let $P'(p_1, p_2, \ldots, p_{d+1}) \in \mathbb{R}^{d+1}$ be a *query point* with $p_{d+1} \leq m$ or $p_{d+1} \geq M$. We wish to find the point in $\mathcal{S}'$ that is closest to P'. The constraint on the last coordinate of P' is introduced because FFF does not require the general case and this speeds up the search by a factor of $\log n$. In this section, we will only discuss the case when $p_{d+1} \leq m$; the other case can be treated similarly.

Let $\mathcal{I}$ be the set of all ordered $(d+1)$-tuples of the form $(\pm 1, \ldots, \pm 1)$, and for any $\epsilon(\epsilon_1, \ldots, \epsilon_{d+1}) \in \mathcal{I}$, let $\Omega'_\epsilon(P)$ refer to the following octant in $\mathbb{R}^{d+1}$:

$$X'(x_1, \ldots, x_{d+1}) \in \Omega'_\epsilon(P) \Longleftrightarrow \forall i \in \{1, 2, \ldots, d+1\}, \begin{cases} x_i < p_i, & \text{if } \epsilon_i = -1, \\ x_i \geq p_i, & \text{if } \epsilon_i = +1. \end{cases}$$

Also, define the family of functions $f_\epsilon : \mathbb{R}^{d+1} \to \mathbb{R}$ for all $\epsilon \in \mathcal{I}$:

$$f_\epsilon : X'(x_1, \ldots, x_{d+1}) \mapsto \sum_{i=1}^{d+1} \epsilon_i x_i,$$

where the parameter ϵ is defined in the same way as used in $\Omega'_\epsilon(P)$. Then when $X' \in \mathcal{S}' \cap \Omega'_\epsilon(P)$, its distance from P' is equal to

$$\sum_{i=1}^{d+1} |x_i - p_i| = \sum_{i=1}^{d+1} \epsilon_i (x_i - p_i) = f_\epsilon(X') - f_\epsilon(P'), \tag{2}$$

but since $p_{d+1} \leq m$, we must have $\mathcal{S}' \cap \Omega'_\epsilon(P) = \varnothing$ for all $\epsilon(\epsilon_1, \ldots, \epsilon_{d+1})$ that have $\epsilon_{d+1} = -1$. All of our computations on $X' \in \mathcal{S}' \cap \Omega'_\epsilon(P)$ in equation (2) must then turn out as

$$\sum_{i=1}^{d+1} |x_i - p_i| = \left(\sum_{i=1}^{d} \epsilon_i x_i \right) + x_{d+1} - \left(\sum_{i=1}^{d} \epsilon_i p_i \right) - p_{d+1},$$

so if we define the family of functions $g_\epsilon : \mathbb{R}^{d+1} \to \mathbb{R}$ for all $\epsilon \in \mathcal{I}$:

$$g_\epsilon : X'(x_1, \ldots, x_{d+1}) \mapsto \left(\sum_{i=1}^{d} \epsilon_i x_i \right) + x_{d+1},$$

then the distance from P' to any $X' \in \mathcal{S}' \cap \Omega'_\epsilon(P)$ is $g_\epsilon(X') - g_\epsilon(P')$.

Now since the only $\Omega'_\epsilon(P)$ that contain points in $\mathcal{S}'$ are those with $\epsilon_{d+1} = +1$, we can repartition $\mathbb{R}^{d+1}$ into the 2^d distinct $\Theta'_\epsilon(P)$, defined as follows:

$$X'(x_1, \ldots, x_{d+1}) \in \Theta'_\epsilon(P) \Longleftrightarrow \forall i \in \{1, 2, \ldots, d\}, \begin{cases} x_i < p_i, & \text{if } \epsilon_i = -1, \\ x_i \geq p_i, & \text{if } \epsilon_i = +1, \end{cases}$$

and the point in $\mathcal{S}'$ closest to P' can be found by looking for the minimum $g_\epsilon(X') - g_\epsilon(P')$ in each of the 2^d distinct $\Theta'_\epsilon(P)$. Within each region, $g_\epsilon(P')$ is constant, and we can compute and store all 2^d distinct $g_\epsilon(X')$ whenever we insert a point X' into $\mathcal{S}'$; thus, since the partition of $\mathbb{R}^{d+1}$ into $\bigcup \Theta'_\epsilon(P)$ ignores all $(d+1)$-st coordinates, we will have reduced our $\mathbb{R}^{d+1}$ dynamic closest-point problem to a $\mathbb{R}^d$ dynamic range search for minimum. According to [2, 3], range trees can solve this in $O(n \log^{d-1} n)$ space with $O(\log^d n)$ query/update time.

2.4 Implementation

FFF stores $\mathcal{W}'_1$ and $\mathcal{W}'_2$ in a pair of *Dynamic Closest-Point Structures*, or *DCPS's*, each of which holds a set of points in $\mathbb{R}^{d+1}$ and supports the following operations:

1. `insert(`P'`)`. Inserts P' into the DCPS if the structure does not yet contain any point Q' with $Q = P$. If it already has such a point Q', then Q' is replaced by P' if $|\sigma(P')| < |\sigma(Q')|$; otherwise, P' is ignored.
2. `delete(`P'`)`. Deletes P' from the DCPS.
3. `query(`P'`)`. Let m and M be the minimum and maximum $(d+1)$-st coordinates in the DCPS, respectively. Then, for a point $P' \in \mathbb{R}^{d+1}$ with $\sigma(P') \leq m$ or $\sigma(P') \geq M$, this function performs a closest-point query; it returns a point X' in the DCPS that is rectilinearly closest to P'. If several points $\{X'_i\}$ yield that minimal distance, then one with minimal PX_i is returned.

The special features of our DCPS insertion and query can be added through simple modifications of implementation details, and do not affect its overall complexity.

We use a priority queue to select seed points for the successive approximations of Section 2.2. Our priority queue stores elements of the form $P'(\lambda, \delta)$, where $P' \in \mathbb{R}^{d+1}$ is the data and the ordered pair (λ, δ) is its associated priority. The λ estimates the length of a shortest path through P and the δ estimates how close P is to $\mathcal{W}_2$. We compare priorities by interpreting the (λ, δ) as sorting keys; that is, $(\lambda_1, \delta_1) > (\lambda_2, \delta_2)$ if and only if $\lambda_1 > \lambda_2$ or $\lambda_1 = \lambda_2$ and $\delta_1 > \delta_2$. When we pop an element off the queue, we retrieve the one with minimal priority.

2.5 Pseudocode

The algorithm is presented in Figure 2.

3 Justification

Throughout this section, k will be an index from the set $\{1, 2\}$. Also, the asterisk will be a shorthand for the predecessor of a point; for instance, P^* will denote the point in $\mathbb{R}^d$ that prompted the insertion (line 43) of P' into its $\mathcal{W}'_k$.

```
1:  algorithm FFF {
2:     insert T'_1 and T'_2 into W'_1 and W'_2, respectively;
3:     MIN := ∞;
4:     push T'_1(T'_1T'_2, T_1T_2) onto priority queue;
5:     while (priority queue, W'_1, and W'_2 are all nonempty) {
6:        pop P'(λ, δ) off priority queue;
7:        if (λ ≥ MIN) { end algorithm; }
8:        if (P ∉ W_1) { return to line 5; }
9:        P'_0 := P' with last coordinate set to zero;
10:       query:  X'_1 := point in W'_1 that is closest to P'_0;
11:       if (P ≠ X_1) { PriorityQueueInsert(P'); }
12:       if (WavefrontsCrossed(X_1, Ω_2)) { return to line 5; }
13:       Visit(X'_1, 1);
14:       query:  X'_2 := point in W'_2 that is closest to X'_1;
15:       if (WavefrontsCrossed(X_2, Ω_1)) { return to line 5; }
16:       Visit(X'_2, 2);
17:    }
18:    Assertion 1: MIN ≠ ∞ ⇔ S(T_1T_2) ≠ ∅ ⇔ S(T_1S) ⊕ S(ST_2) ⊆ S(T_1T_2);
19: }
20:
21: procedure PriorityQueueInsert(P') {
22:    if (P ∈ Ω_2) {
23:       Assertion 2: PT_2 is known;
24:       Y' := P' with last coordinate set to −PT_2;
25:    }
26:    else { query:  Y' := point in W'_2 that is closest to P'; }
27:    push P'(P'Y', PY) into priority queue;
28: }
29:
30: function WavefrontsCrossed(X, Ω) {
31:    if (X ∈ Ω) {
32:       Assertion 3: T_1X + XT_2 is known;
33:       if (T_1X + XT_2 < MIN) { S := X;  MIN := T_1X + XT_2; }
34:       return TRUE;
35:    }
36:    else { return FALSE; }
37: }
38:
39: procedure Visit(X', k) {
40:    Assertion 4: T_kX = |σ(X')|;
41:    move X from W_k to Ω_k;
42:    for (all {V ∉ Ω_k : XV ∈ E}) {
43:       insert V' into W'_k, where we use T_kX + XV for |σ(V')|;
44:       if (k = 1) { PriorityQueueInsert(V'); }
45:    }
46: }
```

Figure 2: FFF pseudocode

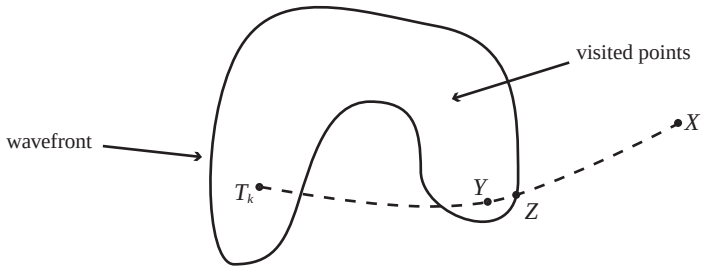

Figure 3: Lemma 1

Lemma 1 *At the beginning of some iteration after the first, let X be a point not in Ω_k and let $\mathcal{P}$ be a path in $\mathcal{G}$ connecting T_k and X. Then there exist two points $Y, Z \in \mathcal{P}$ such that $\overline{YZ} \in \mathcal{P}$, $Y \in \Omega_k$, and $Z \in \mathcal{W}_k$.*

Proof: The first iteration moves T_k into Ω_k, so $\mathcal{P} \cap \Omega_k \neq \varnothing$ in every iteration thereafter. Let Y be the point in that intersection whose distance from X along edges in $\mathcal{P}$ is minimal, and let $Z \in \mathcal{P}$ be the point adjacent to Y that is closer to X along $\mathcal{P}$ than Y was (fig. 3). From the minimality of Y, we know that $Z \notin \Omega_k$, so when Y was inserted into Ω_k (line 41), FFF must have inserted Z into $\mathcal{W}_k$ (lines 42–43). Again by the minimality of Y, Z cannot currently be in Ω_k, so this Y and this Z satisfy the statement of the lemma. □

Theorem 1 *In every iteration after the first, when FFF visits (lines 13, 16) a point $X \in \mathcal{W}_k$,*

$$|\sigma(X')| = \mathbf{T_k X}^* + X^* X = \min_{M \in \mathcal{W}_k} \{\mathbf{T_k M}^* + M^* M + MX\} = \mathbf{T_k X}, \qquad (3)$$

and a path in $\mathcal{S}(\mathbf{T_k X})$ can be obtained by tracing back through predecessors from X.

Proof: We proceed by induction.

Base Case. FFF begins with $\Omega_k = \varnothing$, $\mathcal{W}_k = \{T_k\}$, and $|\sigma(T_k')| = 0$, so in the second iteration, $X^* = T_k = M^*$, $\mathbf{T_k X} = T_k X$, and $\mathbf{T_k M}^* = 0$. Now $|\sigma(X')| = \mathbf{T_k T_k} + T_k X = \mathbf{T_k X}$ and by the triangle inequality $\mathbf{T_k M}^* + M^* M + MX \geq T_k X = \mathbf{T_k X}$; our base case is true.

Induction Step. By line 43 and the induction hypothesis, $|\sigma(X')| = \mathbf{T_k X}^* + X^* X$, so the left equality is true. To prove the center equality, we use indirect proof; suppose, for the sake of contradiction, that

$$\mathbf{T_k X}^* + X^* X \neq \min_{M \in \mathcal{W}_k} \{\mathbf{T_k M}^* + M^* M + MX\}.$$

Then there must currently exist some point $M' \in \mathcal{W}_k'$ for which $\mathbf{T_k M}^* + M^* M + MX < |\sigma(X')|$, and $\mathbf{T_k M}^* + M^* M = |\sigma(M')|$ by line 43. Let Q' be the query

point (the P_0' or the X_1') that led to X' in line 10 or 14. Since $\sigma(Q')$ and $\sigma(M')$ are not of the same sign, $Q'M' = |\sigma(Q')| + QM + |\sigma(M')|$. Combining all of our information:

$$\begin{aligned} Q'M' &= |\sigma(Q')| + QM + |\sigma(M')| \\ &< |\sigma(Q')| + QM - MX + |\sigma(X')| \\ &\leq |\sigma(Q')| + QX + |\sigma(X')| \\ &= Q'X'. \end{aligned}$$

Thus X' is not the point in $\mathcal{W}_k'$ that is closest to Q'; contradiction.

Next, we prove the rightmost equality in equation (3). Let $\mathcal{P}$ be a path in $\mathcal{S}(\mathbf{T_k X})$. Since $X \in \mathcal{W}_k$ and $\mathcal{W}_k \cap \Omega_k = \varnothing$, Lemma 1 implies that there exists some point K in $\mathcal{P} \cap \mathcal{W}_k$ with adjacent point $K_0 \in \mathcal{P} \cap \Omega_k$. Since both K_0 and K^* are in Ω_k and are adjacent to K, they each must have attempted to insert an embedding of K into $\mathcal{W}_k'$ by lines 42–43. Our DCPS insertion operation has the special property that it retains only the best embedding, so $\mathbf{T_k K}^* + K^*K \leq \mathbf{T_k K_0} + K_0K$. Yet K_0 is adjacent to K along $\mathcal{P} \in \mathcal{S}(\mathbf{T_k K})$, so $\mathbf{T_k K_0} + K_0K = \mathbf{T_k K}$. Furthermore, $\overline{K^*K} \in \mathcal{E}$, so $K^*K = \mathbf{K}^*\mathbf{K}$ and thus $\mathbf{T_k K}^* + K^*K \geq \mathbf{T_k K}$. Hence $\mathbf{T_k K}^* + K^*K = \mathbf{T_k K}$ so

$$\begin{aligned} \min_{M \in \mathcal{W}_k} \{\mathbf{T_k M}^* + M^*M + MX\} &\leq \mathbf{T_k K}^* + K^*K + KX \\ &= \mathbf{T_k K} + KX \\ &\leq \mathbf{T_k K} + \mathbf{KX} \\ &= l(\mathcal{P}) \\ &= \mathbf{T_k X}, \end{aligned}$$

and

$$\mathbf{T_k X} \leq \mathbf{T_k X}^* + X^*X = \min_{M \in \mathcal{W}_k} \{\mathbf{T_k M}^* + M^*M + MX\}$$

by the center equality. Therefore, the rightmost equality is true.

Only the last claim of our theorem remains to be proven. By equation (3), $\mathbf{T_k X}^* + X^*X = \mathbf{T_k X}$, so since $\overline{X^*X} \in \mathcal{E}$, we know that $\mathcal{S}(\mathbf{T_k X}^*) \oplus \mathcal{S}(\mathbf{X}^*\mathbf{X}) \subseteq \mathcal{S}(\mathbf{T_k X})$. Now $X^* \in \Omega_k$, so it must have been visited at some earlier time. From the induction hypothesis, a path in $\mathcal{S}(\mathbf{T_k X}^*)$ can be obtained by following predecessors back from X^*; if we append the edge $\overline{X^*X}$ to this path, then we will have a path in $\mathcal{S}(\mathbf{T_k X})$, so we are done. □

Corollary 1 *At the start of every iteration, for every $O \in \Omega_k$, a path in $\mathcal{S}(\mathbf{T_k O})$ can be obtained by tracing back through predecessors from O.*

Proof: This follows immediately from Theorem 1. □

Corollary 2 *Assertions 2, 3, and 4 are true.*

Proof: These follow from Theorem 1 and Corollary 1. □

Corollary 3 *Let $\mathcal{W}'_k$ be one of the embedded wavefronts at the beginning of some iteration, and let P' be a point that is not on the same side of the $\sigma = 0$ hyperplane. That is, if $k = 1$, $\sigma(P')$ should be nonpositive, and if $k = 2$, $\sigma(P')$ should be nonnegative. Furthermore, suppose that $P \notin \Omega_k$ and let Q' be the point in $\mathcal{W}'_k$ that is closest to P'. Then $QQ^* + \mathbf{Q^*T_k} = |\sigma(Q')| = \mathbf{QT_k}$ and $\mathbf{PT_k} \geq PQ + \mathbf{QT_k}$.*

Proof: The first claim follows from the induction step of Theorem 1, since that proof only used the fact that X resulted from a query by a point that was not on the same side of the $\sigma = 0$ hyperplane. To prove the second claim, observe that if we replace X with P in the proof of the rightmost equality in Theorem 1, we find a point $K' \in \mathcal{W}'_k$ for which $\mathbf{T_kK^*} + K^*K + KP \leq \mathbf{T_kP}$. Yet $|\sigma(K')| = \mathbf{T_kK^*} + K^*K$ by line 43, and $|\sigma(Q')| = \mathbf{QT_k}$ by the first claim, so since $Q' \in \mathcal{W}'_k$ is closer to P' than $K' \in \mathcal{W}'_k$,

$$\begin{aligned} Q'P' &\leq K'P', \\ |\sigma(Q')| + QP + |\sigma(P')| &\leq |\sigma(K')| + KP + |\sigma(P')|, \\ |\sigma(Q')| + QP &\leq |\sigma(K')| + KP, \\ \mathbf{QT_k} + QP &\leq \mathbf{T_kK^*} + K^*K + KP \leq \mathbf{T_kP} \end{aligned}$$

as desired. □

Lemma 2 *Let X' be a point in $\mathcal{W}'_1$ at the start of some iteration after the first. If $\mathbf{T_1X^*} + X^*X + \mathbf{XT_2} <$ MIN, then the priority queue contains an element $X'(\lambda, \delta)$ with $\lambda \leq \mathbf{T_1X^*} + X^*X + \mathbf{XT_2}$.*

Proof: We first show that when a point X is visited and a neighbor V' is inserted into $\mathcal{W}'_1$, the priority queue element $V'(\lambda, \delta)$ that FFF inserts (line 44) has $\lambda \leq \mathbf{T_1X} + XV + \mathbf{VT_2}$.

In the call to `priorityQueueInsert`, if line 24 is used, then by Corollary 2, $\lambda = |\sigma(V')| + \mathbf{VT_2} = \mathbf{T_1X} + XV + \mathbf{VT_2}$. If line 26 is used, we can apply Corollary 3 to see that $\mathbf{VT_2} \geq VY + \mathbf{YT_2} = VY + |\sigma(Y')|$, so

$$\begin{aligned} \mathbf{T_1X} + XV + \mathbf{VT_2} &\geq \mathbf{T_1X} + XV + VY + |\sigma(Y')| \\ &= |\sigma(V')| + VY + |\sigma(Y')| \\ &= V'Y' = \lambda. \end{aligned}$$

Thus the lemma's condition holds at the moment of insertion, and we are left to deal with the case where an element $P'(\lambda, \delta)$ is popped off the priority queue but P' is not removed from $\mathcal{W}'_1$.

Note that we may pop off an element that is not in $\mathcal{W}'_1$. This happens when a point V is inserted into $\mathcal{W}_1$ more than once, with different V' embeddings. In

such cases, our DCPS only retains the one with lesser $|\sigma(V')|$, although both remain in the priority queue. We wish to ignore all other V' since they represent longer paths from T_k to V; such is the purpose of line 8.

If our iteration passes line 8 but P is not visited, then we may need to reinsert P' into the priority queue to satisfy the lemma. If $P \neq X_1$, this job is successfully accomplished by line 11, since the above analysis of `priorityQueueInsert` applies here. The only case left to consider is when the iteration terminates via line 12 with $P = X_1 \in \Omega_2$. Now, $\mathbf{T_1X_1^*} + X_1^*X_1 + \mathbf{X_1T_2} = \mathbf{T_1X_1} + \mathbf{X_1T_2}$ by Corollary 3 since X_1' resulted from a DCPS query by P_0'. Yet line 33 makes this at least MIN and the lemma makes no claim when $\mathbf{T_1X_1^*} + X_1^*X_1 + \mathbf{X_1T_2} \geq$ MIN, so we are done. □

Lemma 3 *After each iteration except the last, Ω_1 has grown or the number of elements in the priority queue with $\lambda <$ MIN has decreased by at least one.*

Proof: If an iteration makes it to line 13, it inserts a point into Ω_1, and if the iteration does not satisfy line 11, then the size of the priority queue decreases by one. Therefore, the only nontrivial case is when the iteration satisfies line 11 but fails to reach line 13; this is when $P = X_1 \in \Omega_2$. The iteration will pass lines 22 and 33, setting $\lambda = \mathbf{T_1X_1} + \mathbf{X_1T_2} \geq$ MIN, but by line 7, λ was originally less than MIN, so the lemma is true. □

Lemma 4 *FFF terminates; it eventually reaches assertion 1.*

Proof: We have a finite graph, so Lemma 3 implies that each iteration that does not decrease the number of elements with $\lambda <$ MIN must grow Ω_1 and insert finitely many elements into the priority queue. The set Ω_1 cannot grow indefinitely, however, because $|\mathcal{V}| < \infty$; hence only a finite number of insertions occur. Once the priority queue runs out of elements with $\lambda <$ MIN, FFF will terminate via line 7. Therefore, we cannot loop forever. □

Theorem 2 *FFF finds a shortest path when one exists; assertion 1 is true.*

Proof: By Lemma 4, FFF terminates, so it suffices to show that assertion 1 is true. We begin with the first equivalence. The reverse implication is trivial, so we move on to prove the forward direction. Suppose for the sake of contradiction that $\mathcal{S}(\mathbf{T_1T_2}) \neq \varnothing$ but FFF terminates with MIN $= \infty$. Let $\mathcal{P}$ be a path in $\mathcal{S}(\mathbf{T_1T_2})$; lines 12 and 15 ensure that $\Omega_1 \cap \Omega_2 = \varnothing$, so since $T_1 \in \Omega_1$ and $T_2 \in \Omega_2$, Lemma 1 applies to $\mathcal{P}$. Thus both wavefronts must be nonempty. By Lemma 2, the priority queue contains an element with finite λ, so FFF could not have terminated via line 5. Yet MIN $= \infty$, so FFF could not have terminated via line 7 and we have a contradiction.

Next we prove the second equivalence. Note that S is undefined until a path is found, so if $\mathcal{S}(\mathbf{T_1S}) \oplus \mathcal{S}(\mathbf{ST_2}) \subseteq \mathcal{S}(\mathbf{T_1T_2})$, then a shortest path exists. It remains to prove the forward implication. If $\mathcal{S}(\mathbf{T_1T_2}) \neq \varnothing$, then from the first equivalence, MIN must be eventually be set to some finite value, at which point S is defined. We must show that $\mathcal{S}(\mathbf{T_1S}) \oplus \mathcal{S}(\mathbf{ST_2}) \subseteq \mathcal{S}(\mathbf{T_1T_2})$. Once we reach

this stage, we will indeed know a shortest path in $\mathcal{S}(\mathbf{T_1T_2})$ because Corollaries 1 and 3 apply to S.

We proceed by contradiction; suppose that FFF does not find a minimal path, terminating with MIN $> \mathbf{T_1T_2}$. Consider the situation at the beginning of the last iteration. Let $\mathcal{P}$ be a path in $\mathcal{S}(\mathbf{T_1T_2})$; by Lemma 1, we can find $X_1 \in \mathcal{W}_1 \cap \mathcal{P}$ and $Y_1 \in \Omega_1 \cap \mathcal{P}$ with $X_1Y_1 \in \mathcal{P}$. This is illustrated in Figure 4.

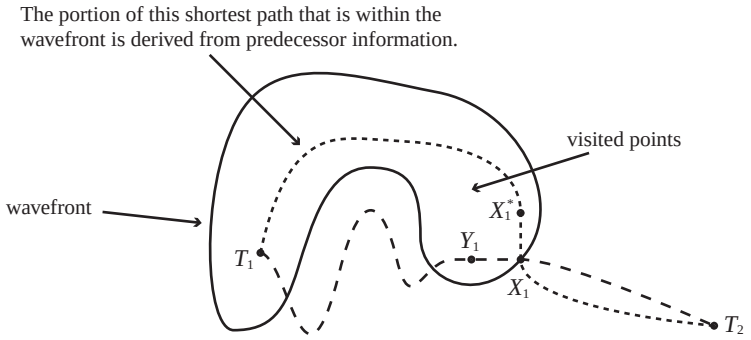

Figure 4: Theorem 2: the dashed curves represent two different shortest paths between T_1 and T_2, and $\mathcal{P}$ is the curve with longer dashes. Wavefront $\mathcal{W}_2$ is not shown.

Since $Y_1 \in \Omega_1$ and $Y_1X_1 \in \mathcal{P} \in \mathcal{S}(\mathbf{T_1T_2})$, the portion of $\mathcal{P}$ between T_1 and X_1 is a shortest path, and hence the embedding X_1' with $|\sigma(X_1')| = \mathbf{T_1X_1}$ is in $\mathcal{W}_1'$. Then $\mathbf{T_1X_1^*} + X_1^*X_1 + \mathbf{X_1T_2} = \mathbf{T_1X_1} + \mathbf{X_1T_2} = \mathbf{T_1T_2}$ and by Lemma 2, there must be an element in the priority queue with $\lambda \leq \mathbf{T_1T_2} <$ MIN. We have a priority queue that looks for minimal λ, so the loop could not have terminated via line 7. Furthermore, the wavefronts are nonempty by the reasoning in the proof of the first equivalence; thus the loop could not have terminated via line 5. Hence we have a contradiction, and FFF is indeed valid. □

4 Performance

Lemma 5 *The main loop runs at most* $(2d + 1)n + 1$ *times, where* d *is the dimension of the search space and* n *is the number of visited vertices.*

Proof: By definition of $\mathcal{G}$, the degree of each vertex is at most $2d$, because each vertex only has $2d$ axis-parallel directions of approach and no two edges overlap in more than one point. We can apply this fact to the argument used in the proof of Lemma 4 to classify iterations into three types: either an iteration adds a point to Ω_1 and inserts up to $2d$ priority queue elements with $\lambda <$ MIN, or it reduces the number of such priority queue elements by at least one, or it is the last iteration.

Since $n = |\Omega_1| + |\Omega_2|$, Ω_1 can grow no more than n times, so we can have no more than n iterations of the first type. The number of priority queue insertions with $\lambda <$ MIN must then be bounded by $2dn$, so we can have no more than $2dn$ iterations of the second type. We can only have one iteration of the third type, and thus the number of iterations is bounded by $(2d+1)n+1$. □

Theorem 3 *FFF's worst-case space and time complexities are $O(n \log^{d-1} n)$ and $O(n \log^d n)$, respectively, where n is the number of vertices visited.*

Proof: Using the method in Section 2.3, our DCPS can achieve $O(n \log^{d-1} n)$ space and $O(\log^d n)$ time for updates and queries, where n is the number of points in the structure. In our algorithm, the DCPS's determine the space complexity, so FFF takes $O(n \log^{d-1} n)$ storage. As for speed, the dominant operations in the main loop are the DCPS update/query, both of which take $O(\log^d n)$. Lemma 5 bounds the number of iterations by $(2d+1)n+1$, so the overall time complexity is $O(n \log^d n)$. □

Theorem 4 *Suppose that in some graph, $\mathcal{S}(\mathbf{T_1T_2}) = \varnothing$ and the A* search described in Section 1.2 visits N vertices. Then FFF will visit at most $2N$ vertices.*

Proof: A* terminates after it visits all N vertices that are connected to T_1 through paths in $\mathcal{G}$. Similarly, FFF will have exhausted $\mathcal{W}_1$ by the time it has visited all of these vertices, so it will have terminated via line 5. In each iteration, X_1 is visited before X_2, so the the number of vertices visited by FFF must be no more than $2N$. □

Theorem 5 *Suppose that in some graph, $\mathcal{S}(\mathbf{T_1T_2}) \neq \varnothing$. Let $\mathcal{N}_0$ be the set of all vertices $X \in \mathcal{G}$ for which $\mathbf{T_1X} + XT_2 < \mathbf{T_1T_2}$, and let $\mathcal{N}_1$ be the set of all vertices $X \in \mathcal{G}$ for which $\mathbf{T_1X} + XT_2 \leq \mathbf{T_1T_2}$. Then A* will visit between $|\mathcal{N}_0|$ and $|\mathcal{N}_1|$ vertices, while FFF will visit no more than $2|\mathcal{N}_1|$ vertices.*

Proof: The set $\mathcal{N}_0$ consists of vertices with A* priority less than $\mathbf{T_1T_2}$, so since the priority of T_2 is $\mathbf{T_1T_2}$, the priority-first search must visit all vertices in $\mathcal{N}_0$. There is no estimate, however, of how many vertices with priority $\mathbf{T_1T_2}$ are visited by A*; the method could visit as few as one or as many as all, depending on the search space.

We now prove the second claim. By the reasoning in the proof of the previous theorem, it suffices to show that $\Omega_1 \subseteq \mathcal{N}_1$. Suppose that $X_1 \in \Omega_1$; by lines 10 and 9, X_1 resulted from a query by some point P_0', which was derived from some P'. When P' was inserted into $\mathcal{W}_1'$, it had a priority $\lambda = P'Y' = |\sigma(P')| + PY + |\sigma(Y')| \geq \mathbf{T_1P} + PY + |\sigma(Y')|$ for some Y. If Y came from line 24, then $\mathbf{T_1P} + PY + |\sigma(Y')| = \mathbf{T_1P} + 0 + \mathbf{PT_2} \geq \mathbf{T_1P} + PT_2$. If Y came from line 26, Corollary 3 implies:

$$\begin{aligned} \mathbf{T_1P} + PY + |\sigma(Y')| &= \mathbf{T_1P} + PY + \mathbf{YT_2} \\ &\geq \mathbf{T_1P} + PY + YT_2 \\ &\geq \mathbf{T_1P} + PT_2. \end{aligned}$$

Therefore, in both cases we get $\lambda \geq \mathbf{T_1P} + PT_2$.

Corollary 3 applied to line 10 tells us that $\mathbf{T_1X_1} + X_1P \leq \mathbf{T_1P}$, so

$$\begin{aligned} \mathbf{T_1X_1} + X_1T_2 &\leq \mathbf{T_1X_1} + X_1P + PT_2 \\ &\leq \mathbf{T_1P} + PT_2 \leq \lambda. \end{aligned}$$

By the argument in the last paragraph of the proof of Theorem 2, until MIN is set to $\mathbf{T_1T_2}$, there always exists a priority queue element with $\lambda \leq \mathbf{T_1T_2}$. At the beginning of every iteration thereafter (except for the last), line 7 ensures that such a priority queue element exists. Since we popped off P' and will visit X_1, $\lambda \leq \mathbf{T_1T_2} \Rightarrow \mathbf{T_1X_1} + X_1T_2 \leq \mathbf{T_1T_2}$. Hence $X_1 \in \mathcal{N}_1$, as desired, and we are done. □

5 Future Work

The theoretical bounds of the previous section are all upper limits; however, the purpose of front-to-front bidirectionality was to provide a more accurate heuristic that would reduce n, the number of visited nodes. Since the complexity is $O(n \log^d n)$, a significant reduction in n would justify the additional log-power complexity factor.

This amounts to classifying the spaces on which front-to-front searches outperform other search algorithms. Unfortunately, that is beyond the scope of this paper; instead, we just provide a simple thought-experiment that illustrates the existence of such spaces.

Figures 5 and 6 are ideal spaces for front-to-front algorithms because the terminals are only accessible through indirect channels that would confuse other heuristics. In fact, they yield relative reductions in n that follow $O(\sqrt{n})$. In light of these simple examples, we can identify one class of spaces for which our front-to-front search is favorable: if source and destination terminals are located in intricate, separated networks that are linked by a large, simple network, then the d-th root reduction in n can be attained.

In this paper, we have established the feasibility of subquadratic front-to-front bidirectional heuristic search. We close by posing a few questions that are opened by this result. Our algorithm is specific to lattice graphs; do there exist analogous tricks that produce subquadratic front-to-front searches in other situations? What types of graphs favor front-to-front heuristics? Do these types of graphs arise in "real-world" situations?

Acknowledgements

Special thanks to Alain Martin and Mika Nyström for introducing this problem to the author, and to Charles Leiserson for providing pointers toward related literature. Thanks also to Po-Ru Loh for providing many valuable suggestions that significantly improved the clarity of this paper.

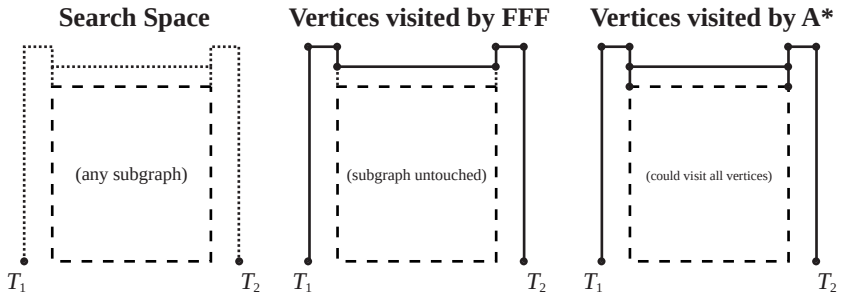

Figure 5: The subgraph can be arbitrarily complex, but FFF will always visit 8 vertices while A* must visit all subgraph vertices with priority less than $\mathbf{T_1T_2}$. Those vertices include all points that can be reached from the top-left vertex by moving down and to the right along edges in the graph.

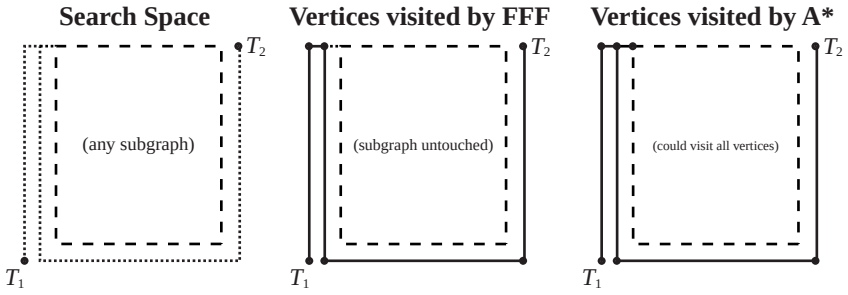

Figure 6: This is another family of search spaces in which FFF always visits a fixed number of vertices while A* can visit arbitrarily many.

References

[1] H. Gabow, J. Bentley, and R. Tarjan, *Scaling and Related Techniques for Geometry Problems*, Proc. 16th Annual ACM Sympos. Theory of Computing (1984), pp. 135–143.

[2] Y.-J. Chiang and R. Tamassia, *Dynamic Algorithms in Computational Geometry*, Proc. IEEE, 80 (1992), pp. 1412–1434.

[3] D. Eppstein, *Dynamic Euclidean Minimum Spanning Trees and Extrema of Binary Functions*, Discrete Comput. Geom., 13 (1995), pp. 111–122.

[4] F. O. Hadlock, *A Shortest Path Algorithm for Grid Graphs*, Networks, 7 (1977), pp. 323–334.

[5] P. Hart, N. Nilsson, and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Trans. Systems Science and Cybernetics, 4 (1968), pp. 100–107.

[6] H. Kaindl and G. Kainz, *Bidirectional Heuristic Search Reconsidered*, J. Artificial Intelligence Res., 7 (1997), pp. 283–317.

[7] G. Lueker, *A Data Structure for Orthogonal Range Queries*, Proc. 19th IEEE Sympos. Foundations of Computer Science (1978), pp. 28–34.

[8] G. Lueker and D. Willard, *A Data Structure for Dynamic Range Queries*, Inform. Process. Lett., 15 (1982), pp. 209–213.

[9] M. Smid, *Closest-Point Problems in Computational Geometry*, in Handbook of Computational Geometry, J. Sack and J. Urrutia, eds., Elsevier, 2000, pp. 877–935.

[10] D. Willard and G. Lueker, *Adding Range Restriction Capability to Dynamic Data Structures*, J. ACM, 32 (1985), pp. 597–617.