Round-Trip Voronoi Diagrams and Doubling Density in Geographic Networks

Matthew T. Dickerson[1], Michael T. Goodrich[2], Thomas D. Dickerson[3], and Ying Daisy Zhuo[1]

[1] Dept. of Computer Science, Middlebury College, Middlebury, VT, USA
{dickerso,yzhuo}@middlebury.edu
[2] Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA, USA
goodrich@ics.uci.edu
[3] St. Michael's College, Colchester, VT, USA
tdickerson@smcvt.edu

Abstract. Given a geographic network G (e.g. road network, utility distribution grid) and a set of sites (e.g. post offices, fire stations), a *two-site Voronoi diagram* labels each vertex $v \in G$ with the pair of sites that minimizes some distance function. The *sum* function defines the "distance" from v to a pair of sites s, t as the sum of the distances from v to each site. The *round-trip* function defines the "distance" as the minimum length tour starting and ending at v and visiting both s and t. A *two-color* variant begins with two different types of sites and labels each vertex with the minimum pair of sites of different types. In this paper, we provide new properties and algorithms for two-site and two-color Voronoi diagrams for these distance functions in a geographic network, including experimental results on the *doubling distance* of various point-of-interest sites. We extend some of these results to multi-color variants.

Keywords: Voronoi diagrams, road networks, round-trip, point-of-interest, two-color.

1 Introduction

Given a set S of d-dimensional points, called *sites*, the *Voronoi diagram* of S is defined to be the subdivision of $\mathbf{R}^d$ into cells, one for each site in S, where the Voronoi cell for site $p \in S$ is the loci of all points closer to p than to any other point in S. The Voronoi diagram is an important and well-studied geometric structure, now more than 100 years old [3,4,14,16,24,25]. For example, when coupled with point location data structures and algorithms, the Voronoi diagram provides a solution to Knuth's well-known *post office problem* [18]: given a set of n post offices, create a structure that allows us to identify, for each house, its nearest post office.

As useful as is the geometric version of the Voronoi diagram defined above for many real world applications, for other applications it is inappropriate because it assumes that distance can be measured via a straight line in Euclidean space—or, in the case of post offices, on a Euclidean plane. That is, it applies a *geometric* structure to a *geographic* problem. In the real world, by contrast, postal delivery cars travel along roads, and not on straight lines (across fields, mountains, rivers, and through forests.) Post offices as well as mailboxes or addresses are points in a geographic network—the graph defined by the set of roads in a given geographic region—not points in $\mathbf{R}^2$. Thus, a Voronoi diagram defined geometrically is, in practice, a poor solution to Knuth's post office problem, especially in geographic regions with natural obstacles, like lakes, rivers, bridges, and freeways, that make the Euclidean metric a inappropriate distance function for determining a nearest post office. A more practical and realistic solution to Knuth's post office problem should use a version of the Voronoi diagram that is defined for geographic networks, or non-negatively weighted graphs.

Furthermore, traditional Voronoi diagrams define the distance only from a single vertex to a single site. However the "cost" related to a particular vertex in a geographic network may be dependent on its relationship to several sites. For example, if one likes to visit three different grocery stores each week for maximum savings, then one would want to associate each vertex on a graph with the *three* grocery store sites that together minimize the sum distance, and so we want to compute the 3-site Voronoi diagram using the sum function. If one visits two grocery stores on each trip, then the goal is to find the sites that minimize the round-trip distance function—which may be different sites than those that minimize the sum function—and we want a round-trip distance function Voronoi diagram to evaluate each vertex. If one visits a post office, grocery store, and department store each week, then optimal sites can be determined by a multi-color Voronoi diagram.

1.1 Graph-Theoretic Voronoi Diagrams

A *geographic network* is a graph $G = (V, E)$ that represents a transportation or flow network, where commodities or people are constrained to travel along the edges of that graph. Examples include road, flight, and railroad networks, utility distribution grids, and sewer lines. We assume that the edges of a geographic network are assigned weights, which represent the cost, distance, or penalty of

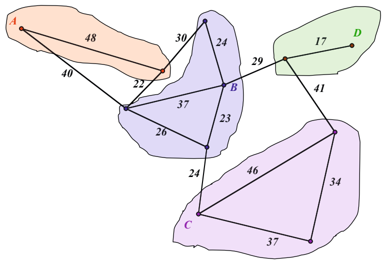

Fig. 1. An example graph-theoretic Voronoi diagram with sites A,B,C,D

moving along that edge, or some combination of these and other factors, such as scenic or ecological value. The only requirement we make with respect to these weights is that they be non-negative. In this paper, we also restrict our attention to undirected geographic networks.

Since all our edge weights are non-negative, and the edges are undirected, a shortest path exists between each pair of vertices in G. The distance, $d(v, w)$ for $v, w \in G$, is defined as the *length* of a *shortest* (i.e., minimum weight) path between v and w. This distance function, d, is well-defined, and $d(v, w) = d(w, v)$. Moreover, since by definition of "shortest", it follows immediately that for any vertices $v, w, x \in G$, $d(v, x) \leq d(v, w) + d(w, x)$; that is, the *triangle inequality* holds for this path distance d.

This observation allows us to define the Voronoi diagram of a geographic network. Formally, we define a geographic network, $G = (V, E)$, to be a set V of *vertices*, a set E of *edges* (which are unordered pairs of distinct vertices), and a *weight function* $w : E \rightarrow \mathbf{R}^+$ mapping edges E to non-negative real numbers. In a road network, this weight function could represent either distance along a road (that is, the Euclidean length of an edge) or the travel time. In the Voronoi diagram problem, we are also given a subset $K \subset V$ of special vertices called *sites*. These are the "post offices" in Knuth's post office problem, but of course they could also be any points of interest (or POIs) such as a schools, hospitals, fire stations, or grocery stores. Each site $v \in K$ is uniquely labeled with a natural number $n(v)$ from 0 to $|K| - 1$, so that we can refer to sites by number. The numbering is also used to resolve ties so that the ordering of sites by distance can be uniquely defined.

The standard first-order graph-theoretic *Voronoi diagram* [21] of G is a labeling of each vertex w in V with the number, $n(v)$, of the vertex v in K that is closest to w. All the vertices with the same label, $n(v)$, are said to be in the *Voronoi region* for v. Intuitively, if a site v in K is considered a post office, then the Voronoi region for v consists of all the homes that *ought* to be in v's zip code. (Note: if we want to consider each house on a block as separate entity with potentially a different closest post office, rather than model the entire block as a single vertex, we should use an individual vertex in V for each house, most of which would be degree-2 vertices in G.) (See Fig. 1.)

We also use these numbers, $n(v)$, to break ties in distances, which allows us to speak of unique closest sites in K for each vertex in V. That is, if we have two distinct sites $v, w \in S$ and a third vertex $x \in V$ such that $d(v, x) = d(w, x)$, then we say that x is closer to v if and only if $n(v) < n(w)$, and otherwise x is closer to y. For example, consider two distinct sites $v, w \in S$ with $n(v) = 0$ and $n(w) = 1$, and a third vertex $x \in V$, and with two edges (v, x) and (w, x) such that $d(v, x) = d(w, x)$. Then x is in the Voronoi region for site v. This becomes particularly important if there is a fourth vertex y and a third edge (x, y). Without the tie-breaking rule using $n(v)$, the entire edge (x, y) would be equidistant to v and w, and if there were no other alternate paths then any entire subgraph of G connected at x would be equidistant from v and w.

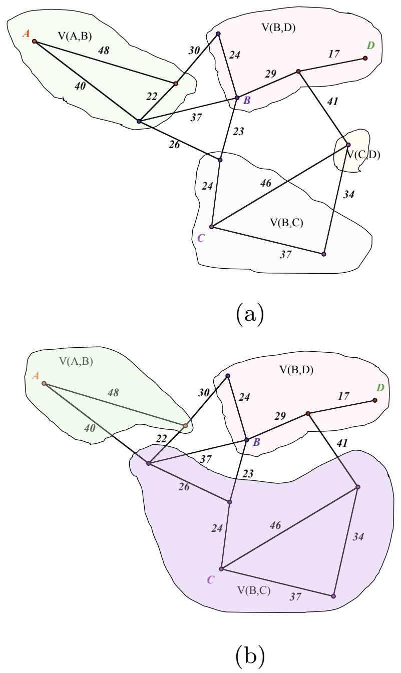

Fig. 2. (a) An example graph-theoretic two-site sum function Voronoi diagram of the same graph from Fig. 1. (b) An example graph-theoretic two-site round-trip function Voronoi diagram of the same graph as in Fig. 1.

Mehlhorn [21] shows that the graph-theoretic Voronoi diagram of a graph G, having n vertices and m edges, can be constructed in $O(n \log n + m)$ time. A similar algorithm is given by Erwig [15]. At a high level, these algorithms perform n simultaneous runs of Dijkstra's single-source shortest-path algorithm [13] (see also [10,17]).

In this paper, however, we are not interested in these types of *single-site* Voronoi diagrams.

1.2 Round-Trip Distance

In a number of applications, we may be interested in labeling the vertices of a geographic network, G, with more information than just their single nearest neighbor from the set of sites, K. We may wish, for instance, to label each vertex v in G with the names of the C closest sites in K, for some $C \leq |K|$. For example, the sites in K may be fire stations, and we may wish to know the three closest fire stations for each house in our network, just in case there is a three-alarm fire at that location.

For many applications, "closest" among a set of neighbors should instead be defined by the *round-trip* or *tour* distance. (For $C = 2$, and for point sites on the

Euclidean plane, this distance was referred to in [7] as the "perimeter" distance, in reference to the perimeter of a triangle, since shortest paths are straight edges. In this paper, although we also focus on the case $c = 2$, we will refer to this function as the "round-trip" distance in reference both to road networks and to the more general case of $C \geq 2$.) (See Fig. 2.) In this notion of distance, we want to take a single trip, starting and ending at our "home" location and visiting two (or more) distinguished sites. Such distances correspond to the work that would need to be done, for example, by someone who needs to leave their house, visit multiple sites to run a number of errands, and then return home. Some hypothetical examples include the following:

– Some legal documents require the signatures of multiple witnesses and/or notaries in order to be executed, so we may need to travel to multiple locations to get them all.
– Some grocery stores place a limit on the amount of special "loss leader" sale items one can purchase in a single visit, so we may need to visit multiple stores to get enough of such items needed for a big party.
– A celebrity just out of rehab may wish to get multiple community service credits in a single trip, for instance, by tutoring students at an educational institution and speaking at an alcoholics anonymous meeting at a religious institution, all on the same day.

In each case, we are likely to want to optimize our travel time to visit all the sites of interest as quickly as possible.

Alternatively, we may have a number of different kinds of sites, such as gas stations, grocery stores, and coffee houses, and we are interested in the three that are closest to each house, in terms of how one could visit all three types of sites in a single trip from home. Thus we are also interested in *multi-color* Voronoi diagrams, where each type of site (such as coffee houses and grocery stores) is represented with a different color. While an individual is not likely to use a computer implementation of a round-trip search algorithm to make such local decisions (as he or she might use Google Maps to find the fastest route to a distant city), retail and service corporations do make location decisions based on such information: maximizing the size of the population for which the location can be efficiently visited.

Figure 3 shows a gray-scale image of the 2-color round-trip Voronoi diagram of the road network in the state of Vermont featuring hospitals as one "color" site, and religious institutions as the other set of sites. For each pair of sites including one hospital and one religious institution, there is a region (in most cases empty) of the set of all vertices having that pair of sites closer than any other pair. Each non-empty region is shaded with a particular shade of grey. (This Voronoi diagram was generated using the algorithm developed in this paper.)

1.3 Related Prior Work

Unlike prior work on graph-theoretic Voronoi diagrams, there is a abundance of prior work for geometric Voronoi diagrams. It is beyond the scope of this paper

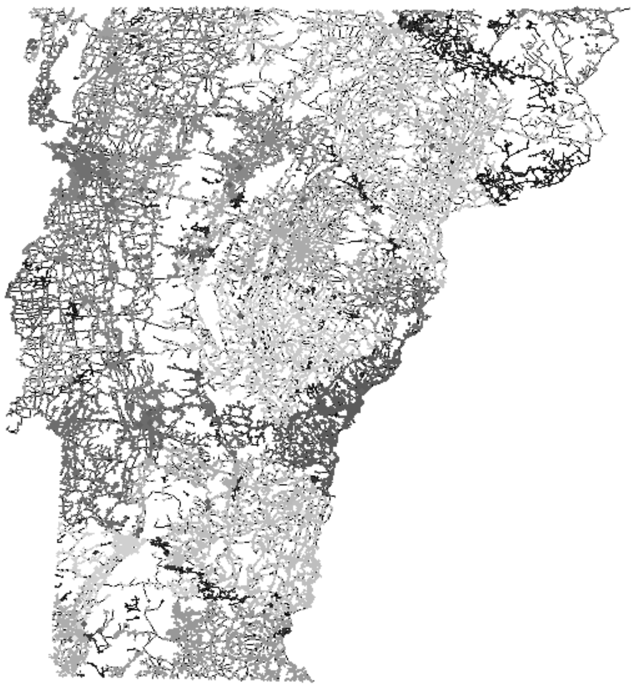

Fig. 3. A 2-Color Round-Trip Voronoi Diagram for the Road Network of the State of Vermont (Using Hospitals and Churches)

to review all this work and its applications. We refer the interested reader to any excellent survey on the subject (e.g., see [3,4,16,24]) and we focus here on previous work on multi-site geometric Voronoi diagrams and on graph-theoretic Voronoi diagrams.

Lee [20] studies k-nearest neighbor Voronoi diagrams in the plane, which are also known as "order-C Voronoi diagrams." These structures define each region, for a site p, to be labeled with the C nearest sites to p. These structures can be constructed for a set of n points in the plane in $O(n^2 + C(n - C)\log^2 n)$ time [9]. Due to their computational complexity, however, order-C Voronoi diagrams have not been accepted as practical solutions to C-nearest neighbor queries. Patroumpas *et al.* [22] study methods for performing C-nearest neighbor queries using an approximate order-C network Voronoi diagram of points in the plane, which has better performance than its exact counterpart.

Two-site distance functions and their corresponding Voronoi diagrams were introduced by Barequet, Dickerson, and Drysdale [7]. (See also [8] for a visualization of this structure.) A two-site distance function is measured from a point to a pair of points. In Euclidean space, it is a function $D_f : \mathbf{R}^2 \times (\mathbf{R}^2 \times \mathbf{R}^2) \to \mathbf{R}$ mapping a point p and a pair of points (v, w) to a non-negative real number. In a graph, it is a function mapping a vertex p and a pair of vertices (v, w)—usually sites—to a $\mathbf{R}$. Two-site distance functions D_f are symmetric on the pair of points

(v, w), though not necessarily on p, thus $D_f(p,(v,w)) = D_f(p,(w,v))$. (For some two-site distance functions, it is also true that $D_f(p,(v,w)) = D_f(v,(p,w))$ but this is incidental.) The sum function, D_S, results in the same Voronoi diagram as the 2-nearest neighbor (order-2) Voronoi diagram, but the authors considered a number of other combination rules as well including *area* and *product*. The complexity of the *round-trip* two-site distance function Voronoi diagram was left by [7] as an open problem, and remains open.

As we mentioned above, single-site graph-theoretic Voronoi diagrams were considered by Mehlhorn [21], who presented an algorithm running in $O(n \log n + m)$ time. More recently, Aichholzer *et al.* [2] study a hybrid scheme that combines geometric distance with a rectilinear transportation network (like a subway), and Abellanas *et al.* [1] study a similar approach where the subway/highway is modeled as a straight line. Bae and Chwa [5,6] study hybrid schemes where distance is defined by a graph embedded in the plane and distance is defined by edge lengths.

As far as multi-site queries are concerned, Safar [23] studies k-nearest neighbor searching in road networks, but he does so using the first-order Voronoi diagram, rather than considering a multi-site Voronoi diagram for geographic networks. Likewise, Kolahdouzan and Shahabi [19] also take the approach of constructing a first-order Voronoi diagram and searching it to perform C-nearest neighbor queries. Instead, de Almeida and Güting [11] compute C-nearest neighbors on the fly using Dijkstra's algorithm. None of these methods actually construct a multi-site or multi-color graph-theoretic Voronoi diagram, however, and, to the the best of our knowledge, there is no previous paper that explicitly studies multi-site or multi-color Voronoi diagrams on graphs. In [12], Dickerson and Goodrich study two-site Voronoi diagrams in graphs, but without employing any techniques that could improve running times beyond repeated Dijkstra-like algorithms.

1.4 Our Results

In this paper, we focus on two-site and two-color Voronoi diagrams on graphs using the round-trip function D_P for defining these concepts, although we also discuss the sum distance function, D_S, as well. In particular, for a vertex p or a point p on an edge e and a pair of sites v, w, our two-site distance functions are defined as follows:

$$D_S(p,(v,w)) = d(p,v) + d(p,w)$$
$$D_P(p,(v,w)) = d(p,v) + d(p,w) + d(v,w)$$

The sum function can easily be extended from 2 to k sites: $D_S(p,(v_1,\ldots,v_k)) = \sum_{1\le i\le k} d(p,v_i)$. Note that with k-site distance functions, we also have a similar rule for breaking ties in distances. In the case that $D(p,(v_1,\ldots,v_k)) = D(p,(w_1,\ldots,w_k))$ for sites v_i, w_i and function D, as a means of breaking ties we consider p closer to whichever of $(v_1,\ldots,v_k)$ and $(w_1,\ldots,w_k)$ has a smaller lexicographical ordering of indices.

We prove several new properties of two-site round-trip distance function Voronoi diagrams on geographic networks, and make use of these properties to provide a new family of algorithms for computing these diagrams. We extend our proofs for the two-color variant, which is arguably more applicable than the one-color variant. (Though as noted above, there are cases when one might wish to visit several grocery stores on one trip, it is easier to imagine a case where we want the shortest tour visiting both a grocery store and a post office.)

One property we explore relates to the *doubling densities* of various types of POI sites on a geographic networks. The doubling density of a class of sites from a vertex v is the number of sites of that type within twice the distance from v as the closest site to v of that type. The run-times of our algorithms depend in part on the average doubling density of various sites from other sites. They also depend on the related density of the total number of edges within twice the distance from one site to the nearest other site of that type. (This latter property could be thought of as a different kind of doubling density.) We will prove a property that allows us to prune our search based on doubling distances, and will also provide experimental results about the doubling densities of various POIs on a set of states.

The algorithms have run times whose expected case is asymptotically faster than the algorithm of [12] under realistic assumptions of how sites are distributed in the network.

Finally, we show how to extend two-site Voronoi diagrams to multi-site and multi-color diagrams, under the sum function, while only increasing the running time by a factor of C, where C is the multiplicity we are interested in.

2 Constructing Graph-Theoretic Voronoi Diagrams

In this section, we review the approach of Mehlhorn [21] and Erwig [15] for constructing a (single-site) graph-theoretic Voronoi diagram of a graph G, having n vertices and m edges, which runs in $O(n \log n + m)$ time, and, for completeness, we also review the two-site sum function algorithm of [12], but with one minor correction.

Given a geographic network, $G = (V, E)$, together with a set of sites, $K \subseteq V$, and a non-negative distance function on the edges in E, the main idea for constructing a graph-theoretic Voronoi diagram for G is to conceptually create a new vertex, a, called the *apex*, which was originally not in V, and connect a to every site in K by a zero-weight edge. We then perform a single-source, shortest-path (SSSP) algorithm from a to every vertex in G, using an efficient implementation of Dijkstra's algorithm. Intuitively, this algorithm grows the Voronoi region for each site out from its center, with the growth for all the sites occurring in parallel. Moreover, since all the Voronoi regions grow simultaneously and each region is contiguous and connected by a subgraph of the shortest-path tree from a, we can label vertices with the name of their Voronoi region as we go.

In more detail, the algorithm begins by labeling each vertex v in K with correct distance $D[v] = 0$ and every other vertex v in V with tentative distance

$D[v] = +\infty$, and we add all these vertices to a priority queue, Q, using their D labels as their keys. In addition, for each vertex v in K, we label v with the name of its Voronoi region, $R(v)$, which in each case is clearly $R(v) = n(v)$. In each iteration, the algorithm removes a vertex v from Q with minimum D value, confirming its D label and R label as being correct. It then performs a *relaxation* for each edge (v, u), incident to v, by testing if $D[v] + w(v, u) < D[u]$. If this condition is true, then we set $D[u] = D[v] + w(v, u)$, updating this key for u in Q, and we set $R(u) = R(v)$, to indicate (tentatively) that, based on what we know so far, u and v should belong to the same Voronoi region. When the algorithm completes, each vertex will have its Voronoi region name confirmed, as well as the distance to the site for this region. Since each vertex is removed exactly once from Q and each key is decreased at most $O(m)$ times, the running time of this algorithm is $O(n \log n + m)$ if Q is implemented as a Fibonacci heap. In addition, note that this algorithm "grows" out the Voronoi regions in increasing order by distance from the apex, a, and it automatically stops the growing of each Voronoi region as soon as it touches another region, since the vertices in an already completed region are (by induction) closer to the apex than the region we are growing.

2.1 Two-Site Distance Functions on Graphs

In this section, we discuss algorithms for two-site Voronoi diagrams, which we then generalize in a subsequent section to multi-site and multi-color Voronoi diagrams. The advantage of this approach is that it highlights the additional complications needed to go from single-site to two-site Voronoi diagrams, while also showing the perhaps surprising result that we can construct two-site Voronoi diagrams with only a small, constant factor blow up in the running time. As mentioned above, the two-site sum function Voronoi diagram is equivalent to the second order two-nearest neighbor Voronoi diagram, which identifies for each vertex v in our graph, G, the two nearest sites to v. We state and prove the equivalence of these two types of Voronoi diagrams in the following simple lemma, the proof of which holds for both Voronoi diagrams in the plane and on weighted undirected graphs.

Lemma 1. *If v and w are the two closest sites to a vertex p in G, then the pair (v, w) minimizes $D_S(p, (v, w))$.*

Proof. Suppose that v and w are the two closest sites to a vertex p in G, but that the pair (v, w) did not minimize $D_S(p, (v, w))$. Without loss of generality, let $d(p, v) \leq d(p, w)$. By our assumption, there exists a vertex x such that $d(p, x) < d(p, w)$. It follows immediately that $D_S(p, (v, x)) < D_S(p, (v, w))$ which is a contradiction.

It follows that the two-site Voronoi diagram is equivalent to the two-nearest neighbor Voronoi diagram for a set of points in the plane or a graph.

So we are ready to formally define our construction problem.

Problem 1. Given a graph $G = (V, K, E)$ of n vertices V, m edges E, and a subset $K \subset V$ of s special vertices called "sites", compute the two-site Sum function Voronoi diagram of G; that is, label each vertex $v \in V$ with the closest pair of sites in K according to the two-site Sum distance function.

Intuitively, the algorithm of [12] for constructing a two-site Voronoi diagram under the sum function is to perform a Dijkstra single-source shortest-path (SSSP) algorithm from each site, in parallel, but visit each vertex twice—once for each of the two closest sites to that vertex.

More specifically, we begin by labeling each vertex v in K with correct first-neighbor distance $D_1[v] = 0$ and every other vertex v in V with tentative first-neighbor distance $D_1[v] = +\infty$, and we add all these vertices to a priority queue, Q, using their D_1 labels as their keys. We also assign each vertex $v \in V$ (including each site in K) its tentative second-neighbor distance, $D_2[v] = +\infty$, but we don't yet use these values as keys for vertices in Q. In addition, for each vertex v in K, we label v with the name of its first-order Voronoi region, $R_1(v)$, which in each case is clearly $R_1(v) = n(v)$. In each iteration, the algorithm removes a vertex v from Q with minimum key. How we then do relaxations depends on whether this key is a D_1 or D_2 value.

– Case 1: The key for v is a D_1 value. In this case we confirm the D_1 and R_1 values for v, and we add v back into Q, but this time we use $D_2[v]$ as v's key. We then perform a *relaxation* for each edge (v, u), incident to v, according to the following test:

```
**Relaxation**$(v, u)$:
    **if** $u$ has had its $R_2$ label confirmed **then**
        Return (for we are done with $u$).
    **else if** $u$ has had its $R_1$ label confirmed **then**
        **if** $R_1(v) \neq R_1(u)$ and $D_1[v] + w(v, u) < D_2[u]$ **then**
            Set $D_2[u] = D_1[v] + w(v, u)$
            Set $R_2(u) = R_1(v)$
        **if** $D_2[v] + w(v, u) < D_2[u]$ **then**
            Set $D_2[u] = D_2[v] + w(v, u)$
            Set $R_2(u) = R_2(v)$.
    **else**
        **if** $D_1[v] + w(v, u) < D_1[u]$ **then**
            Set $D_1[u] = D_1[v] + w(v, u)$
            Set $R_1(u) = R_1(v)$.
```

  In addition, if the D_1 or D_2 label for u changes, then we update this key for u in Q. Moreover, since we are confirming the D_1 and R_1 labels for v, in this case, we also do a reverse relaxation for each edge incident to v by calling **Relaxation**(u, v) on each one.

– Case 2: The key for v is a D_2 value. In this case we confirm the D_2 and R_2 values for v, and we do a relaxation for each edge (v, u), incident to v, as above (but with no reverse relaxations).

When the algorithm completes, each vertex will have its two-site Voronoi region names confirmed, as well as the distance to each of its two-nearest sites for this region.

The correctness of this algorithm follows from the correctness of the SSSP algorithm and from Lemma 1. The SSSP algorithm guarantees that vertices will be visited in increasing order of distance from the origin(s) of the search. Lemma 1 states that the closest two sites to v are also the closest pair according to the sum two-site distance function.

For the analysis of this algorithm, first note that no vertex will be visited more than twice, since each vertex is added to the queue, Q, twice—once for its first-order nearest neighbor and once for its second-order nearest neighbor. Moreover, once a vertex is added to Q, its key value is only decreased until it is removed from Q. Thus, this algorithm requires $O(n \log n + m)$ time in the worst cast when Q is implemented using a Fibonacci heap, where n is the number of vertices in G and m is the number of edges. By the same reasoning, the priority queue Q won't grow larger than $O(n)$ during the algorithm, so the space required is $O(n)$.

3 Properties of Round-Trip Voronoi Diagrams on Graphs

Using the sum distance function for a two-site graph-theoretic Voronoi diagram allows us to label each vertex in G with its two nearest neighbors. Such a labeling is appropriate, for example, for fire stations or police stations, where we might want agents from both locations to travel to our home, or if we need to take separate trips to different sites. If instead we want to leave our home, travel to two nearby sites on the same trip, and return home, then we will need to use the round-trip function. Before presenting a new algorithm for this function, we first prove several properties of the round-trip distance function diagram on graphs.

Our first lemma is relatively straightforward, but provides an important property for pruning searches in our algorithms.

Lemma 2. *Let v be any vertex in a geographic network G, (s,t) a pair of sites in G minimizing the round-trip distance function D_P from v. Then for any sites p, q in G:*

$$d(v,s) \leq (d(v,p) + d(v,q) + d(p,q))/2$$
$$d(v,t) \leq (d(v,p) + d(v,q) + d(p,q))/2$$

Proof of Lemma 2: By assumption, $D_P(v,(s,t)) \leq D_P(v,(p,q))$. By the triangle inequality, $2d(v,s) \leq d(v,s) + d(v,t) + d(s,t)) = D_P(v,(s,t))$. Combining these, we get, $d(v,s) \leq \frac{1}{2}D_P(v,(p,q)) = \frac{1}{2}(d(v,p) + d(v,q) + d(p,q))$. The argument for $d(v,t)$ is symmetric. **End Proof.**

What this means is that if we know of some tour from vertex v through sites p and q—that is, we have a *candidate* pair (p,q) to minimize the round-trip distance from v— then our algorithms can safely ignore any other site s that is further from v than $\frac{1}{2}D_P(v,(p,q))$ because s cannot be a part of a pair that minimizes the round-trip distance from v.

This lemma combined with the triangle inequality $d(p,q) \leq d(v,p) + d(v,q)$ leads to the following corollary, which is a weaker condition, but one easily implementable as a pruning technique on a SSSP search.

Corollary 1. *Let p, q be the two sites closest to some vertex v under normal graph distance, and (s,t) the pair of sites minimizing the round-trip function D_P from v. Then $d(v,s) \leq d(v,p) + d(v,q)$ and $d(v,t) \leq d(v,p) + d(v,q)$.*

The following *double distance* lemma provides a similar but less obvious condition that can also be used for pruning.

Lemma 3. ***(Doubling Distance Property)*** *For any pair of sites s, t in a geographic network G, if there exists any other sites $p, q \in G$ such that $d(s,t) > 2d(s,p)$ and $d(s,t) > 2d(t,q))$, then (s,t) cannot minimize the round-trip distance function for* any *vertex $v \in G$.*

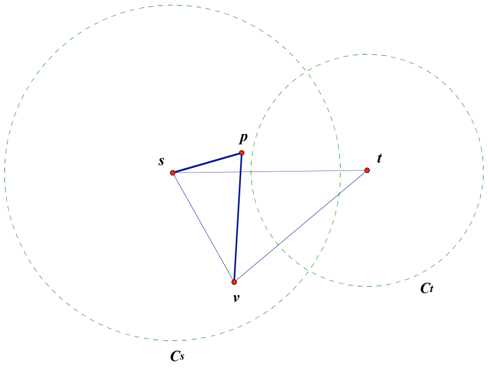

Fig. 4. Illustrating the proof of Lemma 3. The edges represent shortest paths, not single edges.

Proof of Lemma 3 (by contradiction): (See Figure 4.) Assume that there is some vertex v such that (s,t) is the closest pair of sites in the round-trip distance—that is, v is in the Voronoi region for (s,t). Assume also that there are sites $p, q \in G$ such that $d(s,t) > 2d(s,p)$ and $d(s,t) > 2d(t,q))$. Without loss of generality, let $d(v,s) \leq d(v,t)$. (Otherwise reverse the role of s and t.) We now consider the round-trip distance $D_P(v,(s,p))$. Applying the triangle inequality, we get: $D_P(v,(s,p)) = d(v,s) + d(s,p) + d(p,v) \leq d(v,s) + d(s,p) + (d(v,s) + d(s,p))$. By assumption, $2d(s,p) < d(s,t)$ and $d(v,s) \leq d(v,t)$ and thus: $D_P(v,(s,p)) < d(v,s) + d(v,t) + d(s,t) = D_P(v,(s,t))$, contradiction our assumption that (s,t) is the closest pair to v. **End Proof**

Note that Lemma 3 holds even if if s and t both meet the condition of Lemma 2—that is, even if for some vertex v, s and t are both closer to v than $\frac{1}{2}D_P(v,(p,q))$ for all sites p, q, the pair (s,t) cannot minimize the round-trip distance from v. Thus if the conditions of Lemma 3 hold, it follows immediately that the Voronoi region for (s,t) is empty in the two-site round-trip distance function Voronoi diagram.

We now state a final property of round-trip function Voronoi diagrams on graphs.

Lemma 4. *Let s be any site in a geographic network G, and v, w any vertices in G such that a shortest path from s to v goes through w. If there exist any sites $p, q \in G$ such that $d(w,s) > \frac{1}{2}(d(w,p) + d(w,q) + d(p,q))$ then s cannot be part of a nearest round-trip pair to v.*

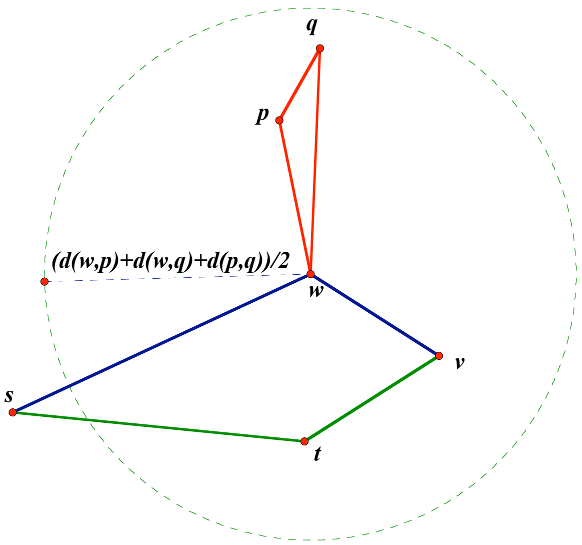

Fig. 5. Illustrating the proof of Lemma 4. The edges represent shortest paths in the graph, and not single edges. The edges (s,w) and (w,v) represent (by assumption) the *shortest* path from s to v.

Proof of Lemma 4: (See Figure 5.) For any t we know the following by the triangle inequality and the fact that w is on the shortest path from s to v:

$$d(v,t) + d(t,s) \geq d(v,w) + d(w,s) \tag{1}$$

Also by assumption

$$2d(w,s) > d(w,p) + d(w,q) + d(p,q). \tag{2}$$

Using Equations 1 and 2, we now show that $D_P(v,(p,q)) < D_P(v,(s,t))$ for s and any other site t.

$$D_P(v,(p,q)) = d(v,p) + d(v,q) + d(p,q)$$

$$\begin{aligned}
&D_P(v,(p,q)) \le [d(v,w)+d(w,p)] + [d(v,w)+d(w,q)] + d(p,q)\\
&D_P(v,(p,q)) \le 2d(v,w)+d(w,p)+d(w,q)+d(p,q)\\
&D_P(v,(p,q)) < 2d(v,w)+2d(w,s)\\
&D_P(v,(p,q)) < 2d(v,t)+2d(t,s)\\
&D_P(v,(p,q)) < d(v,t)+d(t,s)+d(v,s)\\
&D_P(v,(p,q)) < D_P(v,(s,t))
\end{aligned}$$

End Proof.

We could rephrase this in the contrapositive, in a form similar to that of Lemma 2. Let v be any vertex in a geographic network G, (s,t) a pair of sites in G minimizing the round-trip distance function D_P from v, w a vertex on a shortest path from s to v, and p,q any sites in G, then $d(w,s) \le (d(w,p) + d(w,q) + d(p,q))/2$. What this Lemma means is that even if the pair of sites (s,t) meet the condition of Lemma 2 for some vertex v—that is, s is a candidate site to be part of the closest pair to v—if the shortest path from s to v goes through some vertex w for which s does not meet that condition, then not only is s not part of a closest pair for w, s also cannot be part of a closest round-trip pair to v. Together, these three lemmas are sufficient to prove the correctness of the algorithms in the following section.

3.1 Two-Color Variants

These lemmas can all be extended to apply to the two-color variant. The two color versions are given below. In the follow lemmas, we let $G = (V,E,S,T)$ be a geographic network, with $S \subset V$ and $T \subset V$ two disjoint sets of sites (of different colors). The two-color round-trip distance is from a vertex in V to a pair of sites (s,t) with $s \in S$ and $t \in T$. The proofs of these lemmas are directly analogous to the proofs above.

Lemma 5. *Let v be any vertex in G. Let $s \in S$ and $t \in T$ be a pair of sites such that (s,t) minimizes the two-color round-trip distance function D_P from v. Let $p \in S$ and $q \in T$ be sites in G. Then:*

$$d(v,s) \le (d(v,p)+d(v,q)+d(p,q))/2$$
$$d(v,t) \le (d(v,p)+d(v,q)+d(p,q))/2$$

Corollary 2. *Let $p \in S$ and $q \in T$ be the sites in S and T respectively that are closest to some vertex v under normal graph distance, and let (s,t) (with $s \in S$ and $t \in T$) be the pair of sites minimizing the two-color round-trip function D_P from v. Then $d(v,s) \le d(v,p)+d(v,q)$ and $d(v,t) \le d(v,p)+d(v,q)$.*

Lemma 6. *For any pair of sites $s \in S$ and $t \in T$ in a geographic network G, if there exists any other sites $p \in T$ and $q \in S$ such that $d(s,t) > 2d(s,p)$ and $d(s,t) > 2d(t,q))$, then (s,t) cannot minimize the round-trip distance function for* any *vertex $v \in G$.*

Lemma 7. *Let s be any site in a geographic network G, and v, w any vertices in G such that a shortest path from s to v goes through w. If there exist any sites $p \in S$ and $q \in T$ such that $d(w, s) > \frac{1}{2}(d(w, p) + d(w, q) + d(p, q))$ then s cannot be part of a nearest round-trip pair to v.*

4 Round-Trip Voronoi Diagram Algorithms

We now provide algorithms to compute the round-trip function Voronoi diagram for a geographic network and set of sites $G = (V, K, E)$. Specifically, the algorithm labels each vertex $v \in V$ with a pair of sites in K minimizing the two-site round-trip distance function from v.

4.1 A Brute Force Algorithm

An algorithm for this problem was first presented in [12]. The algorithm, in Step 1, performs a complete SSSP algorithm on G from each of the k sites in K. (Unlike in the Sum function algorithm above, these searches do not need to be interleaved—that is, performed in parallel—as the algorithm searches the entire graph from each site.) It records the distances from v to every site in K, and then creates a table of distances between all pairs of sites $(p, q) \in K$, allowing constant time access to $d(p, q)$. Then, in Step 2, for each vertex $v \in V$ and each pair of sites $(p, q) \in S$, the algorithm explicitly computes the round-trip distance

$$D_P(v, (p, q)) = d(v, p) + d(v, q) + d(p, q)$$

and labels each v with pair (p, q) minimizing this function.

This brute force approach uses the SSSP algorithm to efficiently compute all distances between pairs of vertices, and then explicitly compares all round-trip distances. The algorithm requires $O(k^2 n + km + kn \log n)$ time and $O(nk)$ space when we implement it using Fibonacci heaps as discussed above.

4.2 Improving the Brute Force Method: A Revised Algorithm

We now show how the properties of the previous section can be used to prune the search depth of the brute force algorithm of [12]. Our new algorithm has three steps, or phases.

Step 1 corresponds to Step 1 of the brute force approach above, except that we interleave the SSSP searches (as is done with the sum function) and we bound the number of sites that visit each vertex using some value B. In practice, this bounds the SSSP search outward from each site in K. The specific value of B–possibly determined as a function of n, m, k –will be described in the next section; the algorithm is correct regardless of the value of B, but its run time will depend on B. Ideally, the SSSP of Step 1 provides enough information for most (or all) of the vertices to determine the pair of sites minimizing the round-trip distiance. However the pruning may result in some vertices having incomplete information.

In Step 2, we need to complete information for each of these vertices that still have incomplete information by preforming an addition SSSP search outward from that vertex until it reaches all the sites satisfying Lemma 2. In particular, the smaller the value of B, the less work is done in Step 1, but the more potential work will need to be done in Step 2.

By Step 3, we have all the distance information needed to compute $D_P(v,(s,t))$ for all pairs of sites (s,t) satisfying Corollary 1, and Lemmas 3 and 4. We need only explicitly compute these distances from the information computed in Steps 1 and 2. Note that the pruning of Step 1 reduces not only the time required by each SSSP search, but also the number of explicit distances computed.

We start with the basic three-phase revised algorithm to compute the round-trip distance function two-site Voronoi diagram on a graph $G = (V, K, E)$.

- **Step 1:** Perform parallel Dijkstra SSSP algorithm from each site $p \in K$. For each vertex $v \in V$, record the distances from the first $B+1$ sites whose SSSP search visits v. Any subsequent search (after the $B+1^{st}$) visiting v is not recorded and the search is terminated. (As we will show, the result of Step 1 is that for each vertex, we have a list of the $B+1$ closest sites in sorted order.
- **Step 2:** For each vertex $v \in V$, let p, q be two closest sites in K, and compute $d_v = d(v,p) + d(v,q)$. By Corollary 1, no site further than d_v from v is a candidate to be part of a pair minimizing the round-trip distance from v. So we consider two cases:
  case i: If the final site p on the sorted list of $B+1$ closest sites to v is further from v than d_v, then we have found all sites closer to v than d_v and no work needs to be done on vertex v in this Step; the list of sites at v contains all possible candidate sites that could be part of a closest pair in the round-trip function.
  case ii: If the final site on the sorted list for v is not further than d_v, then we cannot guarantee that v was visited by all the candidate sites. In this case, we perform a SSSP algorithm from v and halt when we reach any vertex further from v than d_v. (Note that this is done also for those vertices that are also sites in K. Since d_v is at least as great as twice the distance from v to its nearest site, we will compute distances between all pairs of sites satisfying Lemma 3.)
- **Step 3:** For each vertex $v \in V$, compute $D_P(v,(s,t)) = d(v,s) + d(v,t) + d(s,t)$ for all sites s, t for which $d(v,s)$ and $d(v,t)$ are stored at v and $d(s,t)$ is stored at either s or t. (If $d(s,t)$ was not computed, then (s,t) is not a candidate pair and may be ignored.) Store at v the pair (s,t) minimizing $D_P(v,(s,t))$.

Correctness. Since the first $B+1$ SSSP searches that reach any vertex will continue through the vertex, by induction each vertex is guaranteed to be reached by the SSSP from at least its closest $B+1$ sites in Step 1. In Step 2, therefore, by looking at the first two and the last site in the list for each vertex v, we can determine if all sites meeting Corollary 1 have visited v. If not, then an

SSSP from v (in case ii) will reach those sites. So by Corollary 1 and Lemma 4, any sites s, t for which the algorithm does not explicitly computer $D_P(v, (s, t))$ cannot be a candidate to minimize the round-trip distance from v.

Worst Case Analysis. We now analyze the algorithm. In Step 1, we visit each vertex $B+1$ times. (If a search arrives at a vertex v that has already been visited $B+1$ times, we count that work to the edge along which the SSSP came to v.) An edge can be traversed at most $B+1$ times from the vertices on each end, for a total of $O(B)$ visits. So Step 1 requires $O(Bm + Bn \log n)$ time because we are overlapping B SSSP searches. We are storing $B+1$ sites and distances at each vertex, as well as a list of $O(k^2)$ distances between each pair of sites, so the space required is $O(Bn + m + k^2)$.

In step 2, we need to store distances between pairs of sites s, t that are candidates to minimize round-trip distance for some vertex. If we use a table, we need worst case $O(k^2)$ space with $O(1)$ time access for any pair (s, t). We also store distances between vertex v and its candidate sites in sorted order; there are at most B sites per list in Step 1, and though in Step 2 the lists can grow to size $O(k)$ we only need to store one list at a time, and so space required is $O(Bn + m + k^2)$.

In step 2, if for a vertex v, the $B+1$ vertices on its list includes all the sites within distance d_v, then we are in case i, and the total amount of work for that vertex in step 2 is $O(1)$ and in step 3 is $O(B^2)$ to explicitly compute all possible round-trip distances of candidate pairs (since for each pair of sites (s, t) the distance $d(s, t)$ has already been computed and can be retrieved in $O(1)$ time.) The total run time for these sites is thus $O(B^2 n)$.

For the rest of the vertices v, those in case ii, we must do a new SSSP from v. This requires $O(m + n \log n)$ time per vertex for the search and $O(k^2)$ time per vertex to look at all pairs of sites. Let A be the number of sites processed in case ii. The run time for all of them is $O(Am + An \log n + Ak^2)$.

The overall run time is thus $O((A + B)(m + n \log n) + B^2 n + Ak^2)$ and the space required is $O(nB + m + k^2)$.

In the next section, we formalize this and also provide some experimental data on values of A and B. First, however, we provide a further revision showing how for many real world networks such as road networks, we can make fuller use of Lemma 2 for an algorithm whose run time is significantly better.

4.3 Further Revisions: A Dynamic Variation

It is possible that we can further reduce the depth of our SSSP searches, and thus the number of candidate pairs examined in our algorithm. Lemma 2 gives a stronger condition than Corollary 1 that must be met by any site that is a candidate to minimize the round-trip distance from a vertex v.

Specifically, instead of using a static bound that prunes the depth of our searches in Step 1, and then simply computing the distance from vertex v to its two nearest sites, we would like to keep an updated minimal value of $D_P(v, (s, t))$ for *all* sites s, t whose SSSP searches have visited v. By Lemmas 2 and 4, we can then prune any search that reaches v from any site further away than the minimum value of $\frac{1}{2}D_P(v, (s, t))$.

Unfortunately, using this stronger condition requires that we dynamically update the minimum value of $D_P(v,(s,t))$ which in turn requires that we precompute or preprocess the values of $d(s,t)$ for all pairs of sites meeting the condition of Lemma 3. This leads to the following two-step algorithm.

- **Step 1:** Perform a SSSP algorithm from each site $p \in K$, terminating the search at the first vertex whose distance from p is greater than $2d(p,q)$ where q is the closest other site to p (discovered in the SSSP). Store the values of $d(p,q)$ for all pairs of sites reached in all of the searches.
- **Step 2:** Perform interleaved SSSP searches from each site $p \in K$, as in Step 1 of the previous algorithm. At each vertex v, store the sites s whose searches reach v along with the distance $d(v,s)$. Using this information and the table from Step 1, once a second site search has visited v, also compute and maintain the distance $D_P(v,(s,t)) = d(v,s) + d(v,t) + d(s,t)$ that minimizes this function among all pairs of sites s,t which have visited v (as well as the pair (s,t) minimizing that distance). Terminate the search from any site farther from v than $\frac{1}{2}D_P(v,(s,t))$ for the minimum value of $D_P(v,(s,t))$ seen so far.

Worst Case Analysis. In the worst case, Step 1 will require $O(m + n\log n)$ time and $O(n+m)$ space for each SSSP for a total of $O(km + kn\log n)$ time, plus an extra $O(k^2)$ space to store the table of distances between pairs of sites, for a total of $O(k^2 + m + n)$ space.

Similarly, in the worst case in Step 2, each of the k SSSP algorithm may require $O(m + n\log n)$ time, but since the searches are interleaved we may need extra $O(nk)$ space to have k searches active at once. We also need to compute $O(k^2)$ distances at each vertex in the worst case, but $k \leq n$ and so we have a total of $O(km + kn\log n)$ time and $O(nk + m)$ space.

As we will see in the following section, however, road networks and many types of POI sites have properties that result in a much more efficient algorithm.

4.4 The Two-Color Variant

The algorithms of the previous section can be extended to the two-color variant, where for each vertex v we want to find the pair of sites (or POIs) of two *different* types–say a grocery store and a post office–that minimize the distance of the shortest round-trip from v. The same basic approaches of both the revised algorithm and the dynamic variant of the revised algorithm work for the two-color version. Lemmas 5, 6, and 7 suffice as proof.

Other than the obvious change that the two-color versions of the algorithms compute and minimize the round-trip distances to pairs of sites of different types, there are only two other primary changes that are necessary. In the first stage, we still perform the interleaved SSSP algorithms from all sites (of both types). However at each vertex v we store separate lists of the sites of the two different types that visit v. This doubles the worst-case memory requirement.

Second, the application of Lemma 6 two-color variant is slightly different than that of Lemma 3 to the standard round-trip distance function. In the dynamic version we need to pre-compute only the distances between sites of different type. In particular, if our two sets of sites are S and T, we need to compute the distance from each $t \in T$ to all the sites in S no more than twice the distance of the closest site in S to t, and symmetrically from each $s \in S$ to all the sites in T no more than twice the distance of the closest site in T to s.

In terms of run-times, what this means for the two-color variant of the round-trip distance function Voronoi diagram is that we care about the *doubling density* of sites in T with respect to sites in S and vice versa–rather than the doubling density of sites in one set to other sites in the same set, as is the case with the standard round-trip two-site distance function.

5 Empirical Analysis on Doubling Density and Dynamic Pruning on Road Networks

The actual run time of our algorithms when they are run on real world data such as road networks with sites coming from standard points-of-interest (POI) files (fire departments, educational institutions, etc.) may be much better than the worst case asymptotic analyses presented in the previous section. In particular, the Lemmas of Section 3 enable each SSSP search in both of our algorithms to be terminated (pruned) well before a linear number of edges and vertices have been visited.

In the first algorithm, for example, the algorithm balances work between the first two phases by a careful choice of the value of B, where B is an expected number of sites that are "close" to most vertices, where "close" for a vertex v is defined by the value of d_v: the sum of the distances to the two nearest sites to v. If the distance to the two nearest sites is, in general, a good indication of how densely packed sites are in the proximity to a vertex v, then the first algorithm—the static version—will perform well.

When the first algorithm does not do well is when there are large rural or wilderness areas that have roads (vertices and edges on the network) but no sites. Consider, for example, a southwest desert area on the edge of an urban area, or a large northeastern or northwestern forest near a city. It might be 50 miles to the nearest gas station site, which sits in a city on the edge of the dessert, but there might be several dozen or even a few hundred gas stations within 100 miles of that rural vertex. In terms of the notation of the previous section, distance from v to the nearest two sites is very large, resulting in a large value of d_v, thus also resulting in a large number of sites—more than B—within a distance of d_v of v.

By contrast, the second algorithm—the dynamic variant—uses the stronger version of the lemma to handle even instances of these poorly distributed sites efficiently, though at the cost of more overhead and a possibly costly preprocessing phase. Consider a vertex v in a rural area that is a large distance from its nearest sites. In real world applications of read networks, however, it is likely that its two nearest sites p and q are close to one another, even though they are far from v.

We thus prune using the dynamically updated value of $\frac{1}{2}(d(v,p)+d(v,q)+d(p,q))$ which is never greater than $d(v,p)+d(v,q)$ and in the example described above is likely to be much smaller.

We ran a variety of experiments to determine empirical performance of both the static and dynamic versions of the algorithm, for both one-color and two-color variants of the problem. In particular, we ran experiments to determine the empirical values of A and B in the static variant, and to determine the doubling densities and the expected number of edge and vertex visits in the dynamic variant. Our experiments included 22 different U.S. states: AK, CT, DC, DE, HI, ID, IL, IN, LA, MA, MD, ME, ND, NH, NJ, NY, OH, RI, TN, UT, VT and WY. The state road networks ranged in size from Hawaii, with only 64892 vertices and 76809 edges to New York, with 716215 vertices and 897451 edges. They also varied greatly in terrain, urban areas, and presence of large areas of wilderness with sparse roads.

Between the one- and two-color variants, we also experimented on a variety of POIs as our sites including: *educational institutions*, *recreational sites*, *hospitals*, *shopping centers*, *fire stations*, and *religious institutions* accessed from a publicly available collection of POIs. (Multiple POIs of the same type at the same address were combined into one site. However POIs in close proximity but at different vertices were treated separately.) The number of sites in a file ranged to a maximum of 7640 (educational institutions in TN). In addition to being publicly available POIs, the variety of sites also made a good choice because some of them are intuitively distributed in a way that could lead to poor performance. Educational institutions—unlike, for example, post offices or fire stations—are unevenly distributed; a large campus for a single institution may contribute to the POI file numerous buildings in close proximity but with different addresses.

We report first on the empirical values for A and B, and then on *doubling densities* of these POIs for both the one- and two-color variants. We also report on the depths to which the SSSP searches need to go before they can be pruned by Lemmas 4 and 7.

5.1 Empirical Values for A and B on Road Networks

To study the distributions impacting the run time of our first algorithm, we ran ten trials that tested four northeastern states (VT,ME,NJ,NH), plus Hawaii (HI), testing each state on two available data sets of POI sites: where fire department data was publicly available (ME and VT), we used fire departments and religious institutions; in the other three states we used educational institutions and religious institutions. The size of the data sets had the number n of vertices ranging 64892 (HI) to 330386 (NJ), with the number m of edges ranging from to 76809 (HI) to 436036 (NJ), and the number k of POI sites ranging from 144 (educational sites in HI) to 759 (religious institutions in ME). For various values of B (the bound at which the first phase of the search is pruned) we computed the value of A (the number of vertices whose information is incomplete after the first phase).

Results were somewhat divergent and seemed to depend considerably on the geographic nature of the road network as described in the introduction to this section. With respect to the efficiency of our algorithm, Maine was the worst state, and performance for religious institutions in New Hampshire was also bad. For both fire departments and religious institutions in Maine, and religious institutions in New Hampshire, setting $B = 5 \log n$ results in values of $A = 11.6\sqrt{n}$ (NH, religious institutions), $A = 12.3\sqrt{n}$ (ME, religious institutions), and $A = 4.47\sqrt{n}$ (ME, fire departments). This is easily explained because Maine has thousands of square miles (known as the North Maine Woods) that are tracked by hundreds of miles of lumber roads, but without any real towns or villages, and thus no POIs. New Hampshire also has a large wilderness area in the White Mountains and northern forests. However on the edge of these large wilderness areas are population centers (e.g. Gorham and Conway, NH, Bangor, ME), with relatively dense distributions of POIs. If we generalize these results to other states comparable in nature to Maine and New Hanpshire, we have an algorithm with run time $O(mn^{\frac{1}{2}} + n^{\frac{3}{2}} \log n + k^2\sqrt{n})$ and the space required is $O(n \log n + m + k^2)$.

By contrast, for the seven other data sets (which use road networks on Hawaii, Vermont, New Hampshire, and New Jersey), if we set $B = \log n \log k$, then we find that $A = 0$—that is, the first step is sufficient to provide complete information for all vertices, and no additional work is required in the second step. The first algorithm therefore requires $O(m \log n \log k + n \log^2 n \log^2 k)$ time and $O(n \log n \log k + m + k^2)$ space, which is a significant asymptotic time improvement over the previous best known algorithm of [12].

5.2 Doubling Density

As noted above, the preprocessing in Step 1 of the dynamic variant of the algorithm must compute a table of distances between pairs of sites that define potential Voronoi regions. In the worst case, this may take $O(km + kn \log n)$ time to compute and additional $O(k^2)$ space to store the table, where k is the number of sites. However by Lemmas 3 and 6, we only need to store pairs of sites (s, t) if s is no more than twice as far from t as the nearest other site to t, or vice versa. This improved efficiency for Step 1 thus depends on a property we call the *doubling density*, which is defined as follows: *for a given vertex v and set of sites S, the doubling density of v is the number of sites in S no further from v than twice the distance to the nearest other site to v (not counting v if $v \in S$.)*

For the one-color two-site tour-distance problem, Lemma 3 indicates that the space and time required by our dynamic algorithm depend on the average doubling density from all sites in the current POI data set to other sites of the same type. For the two-color version where we have one set of sites K_1 and another set K_2, Lemma 6 tells us that the algorithm's space and time depend on the average doubling density of sites of type K_1 from sites of type K_2, and the average doubling density of sites of type K_2 from sites of type K_1. Empirical results of average doubling density for for both the one-color and two-color version are promising.

Let d be the total double density—that is, the number of "candidate pairs" of sites, one of each type or "color", that might have non-empty Voronoi regions. In the worst case, in the one-color case, d could be $\Omega(k^2)$ where k is the number of sites. However empirical results suggest that d is $O(k)$; or, rephrased, the average doubling density per site, d/k, is constant.

We ran nine trials for the one-color version using three POI types (religious and educational institutions and fire stations) on a mix of states. Results were very consistent. In all trials, the average doubling density for sites of a single type was found to be less than 4.1. As a result, for values of k tested up to 1783, the number of pairs of sites whose distances need to be stored for use in Step 2—that is, pairs that are candidates by Lemma 3 to minimize the round-trip distance for at least one vertex, and thus are candidates to have a non-empty Voronoi region–is empirically less than $5k$, or $O(k)$. These values appear to be independent of n. However the data does indicate a possible logarithmic relationship to k, the number of sites. In particular, for several different road networks and POI files, the average doubling density appears to be $\Theta(\log k)$, meaning the total number of pairs to be stored is $\Theta(k \log k)$ rather than $O(k)$, though for practical purposes on the POIs examined, the number of candidate pairs is $< 5k$. (Much larger POI files for the same sets of states would be needed to verify this result.) To avoid using a sparse direct table of size $\Theta(k^2)$, we can use a hash table to store these pairs in $O(k)$ or $O(k \log k)$ space, and provide $O(1)$ expected time access in Step 2.

Though the doubling density of sites (and, in particular, the size of the table computer in Step 1) impacts the size and run time of later steps, the run time of Step 1 itself is actually determined by the total number of edge visits in all of the SSSP searches used to compute the table. We kept track of the number of times each edge in the graph was visited in Step 1. In all trials for the one-color version, the average number of visits per edge was less than 5. In fact, while for fire department POIs the value was between 4 and 5, for religious and educational institutions, the number was less than 3 in all trials. These values appear to be true constants, independent of either n or k. As a result, all of the SSSPs in Step 1 of the second algorithm combined require only $O(m + n \log n)$ time—only a small constant factor more than a single SSSP search—and this holds whether the doubling density is $O(1)$ or $O(\log k)$).

For the two-color variant, we also ran trials on numerous states and a variety of different points-of-interest for the sites, to empirically determine the values of the double densities. The states (and district) tested included: AK, CT, DC, DE, HI, ID, MD, ME, ND, NH, RI, UT, VT, and WY. The types of POIs tested included education institutions, religions institutions, recreational centers, hospitals, and shopping malls.

Again, let d be the total double density—that is, the number of "candidate pairs" of sites, one of each type or "color", that might have non-empty Voronoi regions. In the worst case, as with the one-color case d could be $\Theta(k_1 k_2)$, or simply $\Theta(k^2)$ where k_1 and k_2 are the number of sites of each type, and k is the total number of sites. However for most combinations of types of sites, the

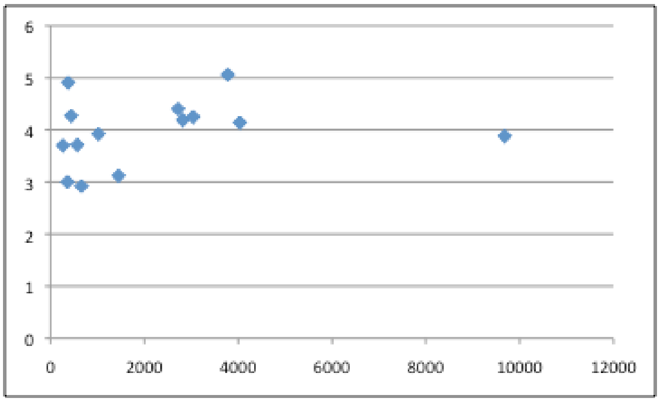

Fig. 6. The number of candidate Voronoi regions divided by k (the number of sites) as a function of k for fourteen states

average doubling density, given by the ratio d/k of the total number of candidate pairs to the total number of sites, ranged from 2.92 to 5.06. A graph of the number of candidate pairs of religious institutions and educational institutions, for the fourteen states listed above is shown in Figure 6.

For the same two types of sites, we also computed the number of times edges were visited on all of these searches, which determines the efficiency of the preprocessing step. The average number of times each edge is visited in computing this table, in the total of both types of searches, was less than 22 in all trials. See Figure 7.

Some combinations of POIs had a somewhat higher average double density. For churches and schools, the average doubling density was as high as 14. However it still appears to be a constant; the doubling density does not increase as the number of vertices or sites increases, but rather the constant seems related to the types of sites and the way they are distributed related to each other. Figure 8 shows the average doubling densities for all five types of POI sites and fourteen states described above.

Thus empirical results suggest a constant average doubling density, a table of candidate pairs that is linear in the number of input sites, and a total run time of $O(m + n \log n)$ and total space of $O(n + m)$ for all the SSSP searches to compute this information.

5.3 Dynamic Pruning on Road Networks

Results for Step 2 are equally promising for the one-color variant, though not for the two-color variant. For all vertices in all trials for one-color, the total number of sites whose SSSP visited the vertex—that is, the number of sites closer to each vertex v than half of the distance of the best known round-trip distance pair yet found—is bounded by 13.

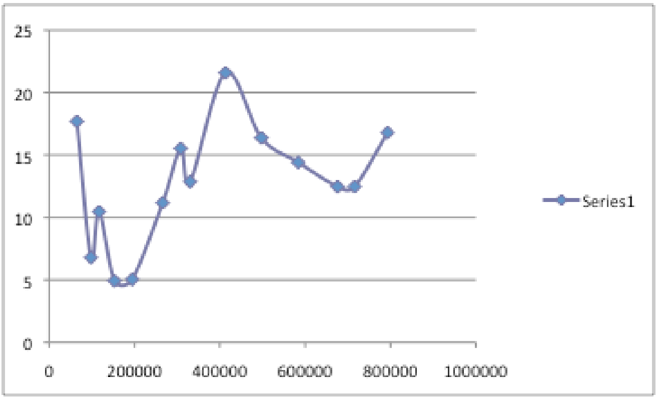

Fig. 7. Based on the doubling density, the average number of times each edge is visited (in the search for candidate pairs) visited as a function of n (for fourteen states)

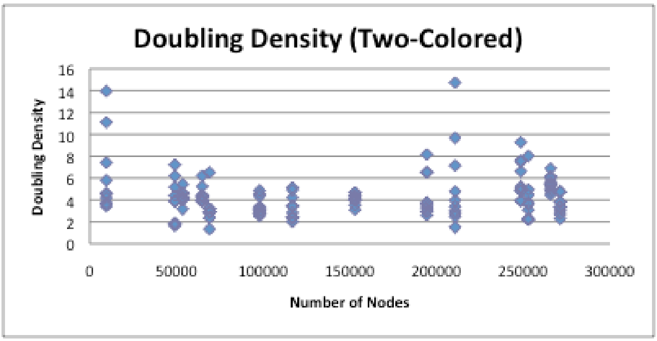

Fig. 8. The average doubling densities of five types of POIs for fourteen states

Thus the total run time of Step 2 is also $O(m + n \log n)$. Empirically, then, the overall run time of the dynamically pruned Algorithm 2 is $O(m + n \log n)$ and the space required is $O(m + n + k \log k)$ when used on road networks with standard POI files.

Results on the level of pruning are equally promising, though less immediately so; an amortized approach is required to see the efficiency. In particular, when sites of one type are much denser than sites of another—as is the case, for example, when there is a large educational campus that contributes numerous entries to a POI file in a small area—then the number of pairs of sites satisfying Lemma 2 and 5 may be large, prohibiting an early pruning of the searches and requiring distance calculations for numerous pairs of sites.

However in the two-color variant, there is no second phase of the algorithm when we must perform a SSSP search from each vertex with incomplete information. Instead, the SSSP searches from the original sites continue until all of them have been pruned. So we can bound the overall run time of these searches simply by the total number of times that a search continues through a vertex—or, equivalently, by the average number of times that each vertex is visited. In all trials except Hawaii, the average number of such vertex visits was less than 10 per vertex, which is linear on the size of the graph. (For Hawaii, which had the smallest graph, the number was 16.) That is, each vertex was visited an average of $O(c)$ SSSP searches for a total of $O(n)$ vertex visits, before further searches are dynamically pruned by Lemma 7.

Equally importantly for the efficiency of the algorithm, the number of pairs of sites of different colors that are are tested for each vertex empirically appears to converge to 10 as the number n of vertices grows. Thus, empirically the total number of distances explicitly computed to find the minimum was $O(n)$.

This immediately implies that the space complexity of all the SSSPs never exceeds $O(n)$. So empirical data suggests that Step 2 also requires $O(m+n \log n)$ time and $O(n+m)$ space.

6 Multi-site and Multi-color Sum Function Extensions

In this section, we discuss how to extend our two-site distance function algorithms to multi-site and multi-color variants, beginning with the multi-site sum function problem. The C-site distance function D_C from a vertex p to a set of C sites $v_1, v_2, \ldots, v_C$ is defined as:

$$D_S(p,(v_1,v_2,\ldots,v_C)) = d(p,v_1) + d(p,v_2) + \cdots + d(p,v_C)$$

Our next goal is to extend our sum function algorithm to solve the following problem.

Problem 2. Given a graph $G = (V, K, E)$ of n vertices V, m edges E, a subset $K \subset V$ of k special vertices called "sites", and an integer $C \leq S$, compute the C-site sum function Voronoi diagram of G; that is, label each vertex $v \in V$ with the closest C sites in K according to the C-site Sum distance function.

As we already showed to be the case when $C = 2$, it is simple to show that for general $C \geq 2$ that the C-site sum function Voronoi diagram is equivalent to the C nearest neighbors Voronoi diagram. This is sufficient argument for the correctness of the following algorithm.

6.1 The Multi-site Sum Algorithm

Our algorithm is a relatively straightforward extension of the two-site algorithm. Perform a Dijkstra single-source shortest-path (SSSP) algorithm from each site in K, in parallel. But visit each vertex only from the first C searches to reach it. Any search to reach a vertex that has already been visited C times by closer sites will not "visit" that vertex or proceed past it. Label each vertex v with the C sites from K that are the closest, namely the sites described above through which the SSSP algorithm from s first passed.

Since no vertex will be visited more than C times, no edge will be traversed more than $2C$ times (C times from each side). Essentially, the algorithm takes at most C times as many steps as the standard SSSP algorithm. The Dijkstra algorithm requires $O(m + n \log n)$ time in the worst cast when implemented using Fibonacci heaps and when visiting each vertex once. In the multi-site case, we visit each vertex C times. In order to avoid an $O(C^2)$ or $O(C \log C)$ term in our running time, we need to maintain, for each vertex, the number of nearest-neighbor sites that we have already completed. The total running time is $O(Cn \log n + Cm)$. By a similar technique we used in our two-site algorithm, we can keep a single copy of each vertex in our priority queue and just insert each vertex C times, but we keep C labels for each vertex, so the space required by our algorithm is $O(Cn)$.

6.2 The Multi-color Sum Algorithm

As discussed in our introduction, another extension of our algorithm is to a multi-color variant. Each of the K sites is colored with a color from 1 to C (for some $C \leq S$). Our goal for each vertex p is to compute the closest site of each color. The result for each vertex is a C-tuple $(v_1, \ldots, v_C)$ such that the C colors of the v_i are unique, and that minimizes the distance $D_S(p, (v_1, v_2, \ldots, v_C)) = d(p, v_1) + \cdots + d(p, v_C)$ among all such possibilities.

Again, our algorithm is a relatively straightforward extension of the standard graph Voronoi diagram algorithm. For each color c, we perform a Dijkstra single-source shortest-path (SSSP) algorithm from each site in K labeled c, visiting each vertex only from the first site to reach it. That is, we perform the Voronoi diagram algorithm once for each color. Label each vertex v with the C sites from K representing the closest site for each color. The result is that each vertex is colored with the closest site for each color; we have C overlapping Voronoi diagrams, one for each color.

Essentially, the algorithm takes at most C times as many steps as the standard Voronoi diagram graph algorithm discussed earlier. For each color, the algorithm requires $O(m + n \log n)$ time in the worst case. The total run time is therefore $O(Cn \log n + Cm)$. Since each search is conducted independently, the priority queue won't grow larger than $O(n)$ and only $O(C)$ information is stored at each vertex/edge, so the space required is $O(Cn)$.

7 Conclusion

We have given complete and efficient algorithms for multi-site and multi-color Voronoi diagrams, under both the Sum and round-trip combination functions. The only variants that we have omitted are the multi-site and multi-color Voronoi diagrams under the round-trip function. The reason we have omitted these variants here, beyond the two-site case, is that computing multi-site or multi-color distance values under the round-trip function requires that we solve miniature versions of the Traveling Salesperson Problem, which is NP-hard. For a small number p of “colors”, an extension to the pruning lemmas might provide some added efficiency in minimizing the number of candidate p-tuples (p-tuples of sites with potentially non-empty p-order Voronoi regions) but the overhead of computing the least-cost tour and dynamically updating the data set would be high. So it is unlikely that we will be able to solve these variants efficiently.

References

1. Abellanas, M., Hurtado, F., Sacristán, V., Icking, C., Ma, L., Klein, R., Langetepe, E., Palop, B.: Voronoi diagram for services neighboring a highway. Inf. Process. Lett. 86(5), 283–288 (2003)
2. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest paths, straight skeletons, and the city Voronoi diagram. In: SCG 2002: Proceedings of the Eighteenth Annual Symposium on Computational Geometry, pp. 151–159. ACM, New York (2002)
3. Aurenhammer, F.: Voronoi diagrams: A survey of a fundamental geometric data structure. ACM Comput. Surv. 23(3), 345–405 (1991)
4. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 201–290. Elsevier Science Publishers B.V., North-Holland, Amsterdam (2000)
5. Bae, S.W., Chwa, K.-Y.: Voronoi Diagrams with a Transportation Network on the Euclidean Plane. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 101–112. Springer, Heidelberg (2004)
6. Bae, S.W., Chwa, K.-Y.: Shortest Paths and Voronoi Diagrams with Transportation Networks Under General Distances. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1007–1018. Springer, Heidelberg (2005)
7. Barequet, G., Dickerson, M.T., Drysdale, R.L.S.: 2-point site Voronoi diagrams. Discrete Appl. Math. 122(1-3), 37–54 (2002)
8. Barequet, G., Scot, R.L., Dickerson, M.T., Guertin, D.S.: 2-point site Voronoi diagrams. In: SCG 2001: Proceedings of the Seventeenth Annual Symposium on Computational Geometry, pp. 323–324. ACM, New York (2001)
9. Chazelle, B., Edelsbrunner, H.: An improved algorithm for constructing kth-order Voronoi diagrams. IEEE Trans. Comput. C-36, 1349–1354 (1987)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
11. de Almeida, V.T., Güting, R.H.: Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In: SAC 2006: Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 58–62. ACM, New York (2006)
12. Dickerson, M.T., Goodrich, M.T.: Two-site voronoi diagrams in geographic networks. In: GIS 2008: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 1–4. ACM, New York (2008)
13. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
14. Dirichlet, G.L.: Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. J. Reine Angew. Math. 40, 209–227 (1850)
15. Erwig, M.: The graph Voronoi diagram with applications. Networks 36(3), 156–163 (2000)

16. Fortune, S.: Voronoi diagrams and Delaunay triangulations. In: Du, D.-Z., Hwang, F.K. (eds.) Computing in Euclidean Geometry, 1st edn. Lecture Notes Series on Computing, vol. 1, pp. 193–233. World Scientific, Singapore (1992)
17. Goodrich, M.T., Tamassia, R.: Algorithm Design: Foundations, Analysis, and Internet Examples. John Wiley & Sons, New York (2002)
18. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1973)
19. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB 2004: Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp. 840–851. VLDB Endowment (2004)
20. Lee, D.T.: On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput. C-31, 478–487 (1982)
21. Mehlhorn, K.: A faster approximation algorithm for the Steiner problem in graphs. Information Processing Letters 27, 125–128 (1988)
22. Patroumpas, K., Minogiannis, T., Sellis, T.: Approximate order-k Voronoi cells over positional streams. In: GIS 2007: Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems, pp. 1–8. ACM, New York (2007)
23. Safar, M.: K nearest neighbor search in navigation systems. Mob. Inf. Syst. 1(3), 207–224 (2005)
24. Sugihara, K.: Algorithms for computing Voronoi diagrams. In: Okabe, A., Boots, B., Sugihara, K. (eds.) Spatial Tesselations: Concepts and Applications of Voronoi Diagrams. John Wiley & Sons, Chichester (1992)
25. Voronoi, G.M.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites. J. Reine Angew. Math. 133, 97–178 (1907)