Efficient Algorithms for Finding Optimal Meeting Point on Road Networks

Da Yan, Zhou Zhao and Wilfred Ng The Hong Kong University of Science and Technology Clear Water Bay, Kowloon, Hong Kong {yanda, zhaozhou, wilfred}@cse.ust.hk

ABSTRACT

Given a set of points Q on a road network, an *optimal meeting point* (OMP) query returns the point on a road network G = (V, E) with the smallest sum of network distances to all the points in Q. This problem has many real world applications, such as minimizing the total travel cost for a group of people who want to find a location for gathering. While this problem has been well studied in the Euclidean space, the recently proposed state-of-the-art algorithm for solving this problem in the context of road networks is still not efficient. In this paper, we propose a new baseline algorithm for the OMP query, which reduces the search space from $|Q| \cdot |E|$ to |V| + |Q|. We also present two effective pruning techniques that further accelerate the baseline algorithm. Finally, in order to support spatial applications that involve large flow of queries and require fast response, an extremely efficient algorithm is proposed to find a high-quality near-optimal meeting point, which is orders of magnitude faster than the exact OMP algorithms. Extensive experiments are conducted to verify the efficiency of our algorithms.

1. INTRODUCTION

Applications ranging from location-based services to computer games require *optimal meeting point* (OMP) query as a basic operation. For example, a travel agency may issue this query to decide the location for a tourist bus to pick up the tourists, so that the tourists can make the least effort to get to the meeting point. This is also true for numerous other scenarios such as an organization that wants to find a place for its members to hold a conference. In strategy games like *WorldofWarcraft*, a computer player may need this query as part of the artificial intelligence program, to decide the routes of its warriors.

There are two popular ways to define the OMP of a set of points $Q = \{q_1, q_2, \ldots, q_n\}$, based on two commonly used cost functions:

- min-sum: Find the point $\overline{x} = \arg \min_x \sum_i d(q_i, x)$.
- min-max: Find the point $\overline{x} = \arg \min_x \max_i d(q_i, x)$.

where $d(p_1, p_2)$ is the distance between the points p_1 and p_2 . The metric of distance can be the Euclidean distance (for a Euclidean

VLDB '11, August 29- September 3, 2011, Seattle, WA

Copyright 2011 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

space) or the network distance (for a road network). The network distance between two points on a road network is the length of the shortest path connecting them. Figure 1(a) illustrates the idea of OMPs using a road network with six people at the six black points, who want to meet together at some location on the road network. The upward (left) triangle in Figure 1(a) is the *min-max* OMP, and the downward (right) one is the *min-sum* OMP.

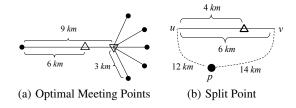


Figure 1: Example of Concepts

A *min-sum* OMP minimizes the total travel distance of all the people, while a *min-max* OMP minimizes the total travel time. For the example in Figure 1(a), the person at the black point on the left has to walk for 9 km to reach the *min-sum* OMP, and those on the right have to wait for him after they reach the meeting point. On the other hand, all the people will walk for 6 km to get to the *min-max* OMP, which is faster than the *min-sum* one.

As transportation is getting more and more convenient nowadays, *min-sum* OMPs are often preferred over *min-max* OMPs. Consider a multi-national corporation that plans to hold a meeting to let all its executive officers in China report to its CEO from the headquarter in USA. The ideal location for the meeting is in China since most of the participants are in China, while the *min-max* OMP may be within some European country on the path between USA and China. If the meeting is held in China, only the CEO from USA need to set out earlier to fly to the meeting location, while the other participants can set out at a later time, and the travel cost is minimized. On the other hand, if the meeting is held in the *min-max* OMP, all the participants have to set out early and the total travel cost is huge. Therefore, we study the *min-sum* OMP query in this paper, and whenever OMP (or "optimal meeting point") is mentioned later, we are referring to the *min-sum* OMP.

While the OMP query has been extensively studied in the Euclidean space, the current state-of-the-art algorithm of it in road networks is still not efficient. In this paper, we identify an interesting property of this problem, which greatly prunes the search space compared with the best-known technique. Two effective pruning rules are proposed to further accelerate query processing. Finally, in order to support spatial applications that require fast response, we propose another algorithm to find a high-quality near-optimal

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

 Table 1: Summary of Notations

Notation	Meaning
$\overline{p_1,p_2}$	the shortest path between points p_1 and p_2
$p_1 \sim p_2$	the line segment of an edge with endpoints p_1
	and p_2 (p_1 and p_2 are on the same edge)
$d(\ell)$	the length of the path/segment ℓ
sd(p,Q)	the sum of distances of point p to the points in
	query set Q (Q is omitted when it is clear from
	the context)

meeting point in considerably less time.

The rest of this paper is organized as follows: Section 2 reviews the previous studies that are highly relevant to the OMP query. Then, we introduce our efficient algorithms, describe the underlying idea, and analyze the time complexity in Section 3. Extensive experiments are presented in Section 4 to show the efficiency of our algorithms for the OMP queries. Finally, we conclude our paper in Section 5.

Table 1 summarizes the notations used in this paper.

2. RELATED WORK

Like the window query [13] and the various nearest neighbor queries [14, 16, 17, 18, 19], the OMP query is also fundamental in spatial databases. The studies of *min-sum* OMP query in the context of Euclidean space date back to the 60s-70s [8, 9, 10, 11]. When the Euclidean distance is adopted as the metric of distance, the OMP query is called the *Weber problem* [8], and the OMP is called the geometric median of the query point set Q.

Cooper [8] extended the Weber problem by posing the problem of minimizing the weighted sums of powers of the Euclidean distances, which was further generalized to handle radial cost functions by Reuven Chen [10]. However, it has been shown that no closed form formula can exist for the Weber problem and its generalizations, and these problems are usually solved by gradient descent methods, with initial point chosen as the center of gravity of the query point set Q. Fortunately, the sum of Euclidean distances is a convex function, since it is the composite of linear-norm-sum functions, all of with preserves convexity. As a result, the gradient descent method is able to approach the global minimum without the worry of being stuck at local minimal values.

On the other hand, the OMP query is not well explored in terms of road networks, where the network distance is adopted as the metric of distance. However, compared with the Weber problem, this is a more realistic scenario for location-based services. Recently, [1] proposed a solution to this problem by checking all the *split points* on the road network. For a point p on a road network, its *split point* on edge (u, v) is define to be the point x such that $d(\overline{p, u}) + d(u \sim x) = d(\overline{p, v}) + d(v \sim x)$ (cf. see Table 1 for the notations). Figure 1(b) illustrates the idea of split point, where the dotted curves denote the shortest paths between the end points. The location marked by triangle in Figure 1(b) is the split point xof p on edge (u, v). The shortest path from p to any point on the left (or right) of x on edge (u, v) passes through u (or v).

It is proved in [1] that the OMP must be a split point, which leads to an algorithm that checks the split point of each query point in Qon each edge in the road network G = (V, E), and picks the split point with the smallest sum of network distances as the OMP. As a result, the search space is $|Q| \cdot |E|$, which is huge. Although [1] included a pruning technique to skip some split points that are guaranteed not to be the OMP, the search space after pruning is still very large. Therefore, a novel road network partitioning scheme is proposed in [1] to further prune the search space, based on the property that the OMP is strictly restrained inside the partition where all the objects in the query set Q are located. This leads to the algorithm which first obtains the smallest partition that encloses all the basic network partitions where the points in Q belong, and then checks the split points in this partition.

A highly relevant but different type of query is the group nearest neighbor query [2, 3]. Given two sets of points P and Q, a group nearest neighbor (GNN) query retrieves the point(s) of Pwith the smallest sum of distances to all the points in Q. GNN queries can be applied, for instance, when n users at locations Q = $\{q_1, q_2, \ldots, q_n\}$ want to choose a restaurant to have dinner together, among a set of restaurants at locations $P = \{p_1, p_2, \dots, p_m\}$ in the city. The GNN query is different from the OMP query in that the candidate result locations of the former is the set P while the candidate result locations of the latter is all the possible locations on the road network. Therefore, the OMP query is more difficult than the GNN query due to its infinite search space. The OMP query is also more general than the GNN query in that it does not require users to determine the kind of place to meet at in advance. For a travel agency that wants to decide the location for a tourist bus to pick up the tourists, the set P does not even exist.

3. ALGORITHM

It is proved in [1] that the OMP must be a split point. As a result, [1] proposed to check all the $|Q| \cdot |E|$ split points for query set Q on the road network G = (V, E), and to pick the one with the smallest sum of distances to all the points in Q as the OMP. However, the $|Q| \cdot |E|$ search space is still very huge. In Section 3.1, we improve the search space to |V| + |Q| by proving that $V \cup Q$ must contain the OMP, and propose our baseline algorithm that only checks all the vertices in V and all the points in Q, and picks the one with the smallest sum of distances to all the points in Q as the OMP.

Then, in Section 3.2, we improve the performance of our baseline algorithm by two online convex-hull-based pruning techniques, which restrains the search space to a small region of the whole road network. This region is always smaller than the partition obtained by [1] that uses the off-line road network partitioning scheme. As a result, our pruning technique achieves better pruning effect.

To further support spatial applications that involve simultaneous evaluation over many queries and require fast response, we propose another algorithm in Section 3.3 that finds a high-quality nearoptimal meeting point in considerably less time.

3.1 Baseline Algorithm

For a query point set Q, the baseline algorithm of [1] treats all the $|Q| \cdot |E|$ split points in the road network G = (V, E) as candidates for the OMP. However, it is not necessary to compute all the split points and evaluate the sums of distances for all of them. In fact, only the vertices in V and the points in Q are the potential candidates for the OMP, as proved in the following:

LEMMA 1. Given a query point set Q, let sd(p) denote the sum of distances of point p to the points in Q. Suppose that no point in Q is on edge (u, v) except for the two end points u and v, then for any point x on edge (u, v), we have $sd(p) \le \max\{sd(u), sd(v)\}$.

PROOF. For a point x on edge (u, v), we denote Q_u as the set of query points whose shortest paths to x pass through u. Accordingly, $Q_v = Q - Q_u$ is the set of query points whose shortest paths to x pass through v. Without loss of generality, let us assume that $|Q_u| \ge |Q_v|$. Figure 2(a) illustrates this scenario, where the hollow points are the query points and the dotted lines are part of their shortest paths to x.

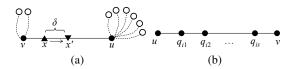


Figure 2: Illustration of Lemma 1 and Theorem 1

Now consider the point x' on edge (u, v) which is δ closer to u than x. Let Q_{ab} $(a, b \in \{u, v\})$ denote the set of query points that belong to Q_a when the meeting point is x and belong to Q_b when the meeting point is x'. Therefore, we can classify the points in Q into four disjoint sets: Q_{uu}, Q_{vv}, Q_{uv} and Q_{vu} .

For these four point sets, we have the following properties:

- $\forall p \in Q_{uu}, d(\overline{p, x'}) = d(\overline{p, x}) \delta.$ **Proof:** $d(\overline{p, x'}) = d(\overline{p, u}) + d(u \sim x') = d(\overline{p, u}) + [d(u \sim x) - \delta] = [d(\overline{p, u}) + d(u \sim x)] - \delta = d(\overline{p, x}) - \delta.$
- $\forall p \in Q_{vv}, d(\overline{p, x'}) = d(\overline{p, x}) + \delta.$ **Proof:** $d(\overline{p, x'}) = d(\overline{p, v}) + d(v \sim x') = d(\overline{p, v}) + [d(v \sim x) + \delta] = [d(\overline{p, v}) + d(v \sim x)] + \delta = d(\overline{p, x}) + \delta.$
- $Q_{uv} = \emptyset$.

Proof: For any $p \in Q_u$ when the meeting point is x, we have $d(\overline{p,v}) + d(v \sim x') = d(\overline{p,v}) + [d(v \sim x) + \delta] > d(\overline{p,v}) + d(v \sim x) \ge d(\overline{p,x}) = d(\overline{p,u}) + d(u \sim x) = d(\overline{p,u}) + [d(u \sim x') + \delta] > d(\overline{p,u}) + d(u \sim x')$, which implies that the shortest path from p to x' cannot pass through v (i.e. $p \notin Q_v$) when the meeting point is x'.

•
$$\forall p \in Q_{vu}, d(\overline{p, x'}) < d(\overline{p, x}) + \delta.$$

Proof: $d(\overline{p, x'}) \le d(\overline{p, v}) + d(v \sim x') = d(\overline{p, v}) + d(v \sim x') + \delta = d(\overline{p, x}) + \delta.$

Therefore, we have

$$\sum_{q \in Q} d(\overline{q, x}) = \left(\sum_{q \in Q_{uu}} + \sum_{q \in Q_{vv}} + \sum_{q \in Q_{uv}} + \sum_{q \in Q_{vu}}\right) d(\overline{q, x})$$

$$> \sum_{q \in Q_{uu}} [d(\overline{q, x'}) + \delta] + \left(\sum_{q \in Q_{vv}} + \sum_{q \in Q_{vu}}\right) [d(\overline{q, x'}) - \delta]$$

$$= \left(\sum_{q \in Q_{uu}} + \sum_{q \in Q_{vv}} + \sum_{q \in Q_{vu}}\right) d(\overline{q, x'})$$

$$+ \delta(|Q_{uu}| - |Q_{vv}| - |Q_{vu}|)$$

As $Q_{uv} = \emptyset$, we have $\sum_{q \in Q_{uv}} d(\overline{q, x'}) = 0$. Besides, since $|Q_u| \ge |Q_v|$ when the meeting point is x, i.e. $|Q_{uu}| + |Q_{uv}| \ge |Q_{vu}| + |Q_{vv}|$, we have $|Q_{uu}| - |Q_{vv}| - |Q_{vu}| \ge -|Q_{uv}| = 0$. According to the above analysis,

$$\begin{split} \sum_{q \in Q} d(\overline{q, x}) &> \left(\sum_{q \in Q_{uu}} + \sum_{q \in Q_{vv}} + \sum_{q \in Q_{uv}} + \sum_{q \in Q_{vu}} \right) d(\overline{q, x'}) \\ &= \sum_{q \in Q} d(\overline{q, x'}) \end{split}$$

Thus, we can conclude that sd(x') < sd(x) for arbitrary x, x'and δ . If we set x to be u, we reach the conclusion that $\forall x'$ on edge (u, v), sd(x') < sd(u). Due to the symmetricity of u and v, if $|Q_v| \ge |Q_u|$ when the meeting point is x, we get: $\forall x'$ on edge (u, v), sd(x') < sd(v). To sum up, $\forall x'$ on edge (u, v), $sd(x') \le \max\{sd(u), sd(v)\}$. \Box Now, let us take into consideration the special case where there exist some query points on an edge, as illustrated by Figure 2(b). By Lemma 1, we can obtain the following theorem:

THEOREM 1. Given an OMP query with query point set Q on a road network $G = (V, E), V \cup Q$ must contain the OMP.

PROOF. For each edge (u, v) that contains some query points on it, but not at the end points u and v, let us denote these query points as $q_{i_1}, q_{i_2}, \ldots, q_{i_s}$, as illustrated in Figure 2(b). We introduce sdummy vertices $p_{i_1}, p_{i_2}, \ldots, p_{i_s}$ on the edge (u, v), where each dummy vertex p_{ij} , $(j = 1, 2, \ldots, s)$ is located at q_{ij} .

After the introduction of the dummy vertices for all the edges that contain some query points on it but not at its end points, we obtain another road network G' such that all the query points in Q are at its vertices. Since the vertex set of G' is $V \cup Q$, we can conclude that $V \cup Q$ must contain the OMP according to Lemma 1.

Theorem 1 is nice since its proof (including that of Lemma 1) relies only on the fact that the road network G is a graph, and is therefore general enough for road networks of any topology. In fact, we can obtain the following more general statement:

THEOREM 2. Given a point set $Q = \{q_1, q_2, \ldots, q_n\}$ on an arbitrary graph G = (V, E), where each point q_i is associated with a weight w_i . If all the weights are integers or rational numbers, then $V \cup Q$ must contain the point $\overline{x} = \arg \min_x \sum_i w_i \cdot d(\overline{q_i, x})$.

PROOF. See Appendix A.

Algorithm 1 Baseline Algorithm
1: given a query point set Q on a road network $G = (V, E)$
2: $opt \leftarrow NULL$
3: $minCost \leftarrow +\infty$
4: for each $q \in Q$ do
5: $cost \leftarrow sumOfDistance(q, Q, minCost)$
6: if $cost < minCost$ then
7: $cost \leftarrow minCost$
8: $opt \leftarrow q$
9: for each $v \in V$ do
10: $cost \leftarrow sumOfDistance(v, Q, minCost)$
11: if $cost < minCost$ then
12: $cost \leftarrow minCost$
13: $opt \leftarrow v$
14: return opt

Alg	$gorithm \ 2 \ sum Of Distance(v, Q, minCost)$
1:	$sum \longleftarrow 0$
2:	for each $q \in Q$ do
3:	$sum \longleftarrow sum + d(\overline{v,q})$
4:	if $sum > minCost$ then
5:	return sum
6:	return sum

Based on Theorem 1, we design our baseline algorithm (Algorithm 1) to check all the points in Q and all the vertices in V, and pick the one with smallest sum of network distances to all the points in Q as the OMP. The function *sumOfDistance* in Lines 5 and 10 of Algorithm 1 computes the sum of network distances of q (or v) to all the points in Q, which is detailed in Algorithm 2.

Lines 4–5 in Algorithm 2 return the partially computed value of the sum of distances for v if it is already larger than minCost. Let

minCost be the smallest sum of distances that is already found for the time being, then Lines 4–5 act as a pruning step to stop the computation for v since it cannot be the optimal point. Lines 8 and 11 in Algorithm 1 will automatically filter out such points.

A basic operation in all our algorithms is to obtain the length of the shortest path between two points p_1 and p_2 , i.e. $d(\overline{p_1, p_2})$, (e.g. Line 3 in Algorithm 2). Since shortest path computation is not our focus, we simply run Dijkstra algorithm for each vertex in the road network, and write all the obtained information into a *index* file on the disk. Please refer to Appendix B for the details on the construction and organization of our *index* file.

After the shortest path *index* file is constructed, the length of the shortest path between two vertices $u, v \in V$ on the road network G = (V, E), i.e. $d(\overline{u, v})$, can be obtained by only one I/O operation, and the shortest path $\overline{u, v} = (p_0 = u, p_1, p_2, \ldots, p_\ell = v)$ can be obtained by ℓ I/O operations. Further, we can utilized the above operation to obtain $d(\overline{p, v})$ for any point p on the road network and any vertex $v \in V$ by two I/O operations, and $d(\overline{p_1, p_2})$ for any two points p_1 and p_2 on the road network by at most four I/O operations. The reasoning of the above statements is presented in Appendix C. To sum up, only O(1) I/O operations are required to obtain the the length of the shortest path between two arbitrary points on the road network, while ℓ I/O operations are required to obtain a shortest path of length ℓ between two points.

As there are |V| vertex to check (Lines 9–13 in Algorithm 1), each of which requires O(|Q|) I/O operations to compute the value of the sum of distances (Lines 2 and 3 in Algorithm 2), the total number of I/O operations required is $O(|Q| \cdot |V|)$. Similarly, we can obtain that Lines 4–8 in Algorithm 1 take $|Q| \cdot O(|Q|) = O(|Q|^2)$ I/O operations. To sum up, the time complexity of Algorithm 1 is $O(|Q| \cdot |V| + |Q|^2)$.

3.2 Pruning Based on Convex Hull

Compared with the split-point-based method in [1], our baseline algorithm in Section 3.1 has already significantly reduced the search space of the OMP query. However, there is still room for the further pruning of the search space. For example, if all the query points are in California, then there is no need to check the vertices in Utah on the road network. Based on this rationale, [1] cuts the whole road network into partitions, and checks only those split points that are in the smallest partition enclosing all the basic network partitions where the query points belong. For the partitioned road network shown in Figure 3(a), where the four black points are the query points, [1] only checks the split points in the gray area.

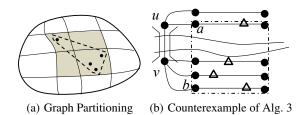


Figure 3: Illustration of Pruning Techniques

However, the correctness of that pruning technique relies on the assumption that whenever two roads cross with each other, there is an intersection vertex at the crossing point. This assumption may not be true in reality, e.g. when one road is a viaduct or a tunnel. Appendix D shows an example road network where the pruning technique of [1] makes mistakes.

Furthermore, the pruning effectiveness of that network partition-

ing scheme is still not sufficient. For example, in Figure 3(a), it is intuitive that the optimal meeting point must appear in the region surrounded by the dotted curve, and it is not necessary to check the remaining part of the gray area, which is the area check by the network partitioning scheme. Besides, there is no strict underlying principle on how to partition a road network in [1].

Now, we propose two online convex-hull-based pruning techniques that are more effective. Although our online pruning techniques do not rely on a pre-computed index, they provide higher efficiency since the dominating factor of the query processing time is the number of points/vertices to check, and the time of convex hull computation required by our techniques is negligible.

Before describing our pruning techniques, we first define the format of a query point q in the query point set Q. Since a query point on a road network G = (V, E) must be on some edge $(u, v) \in E$, we define the format of a query point as follows:

DEFINITION 1. Given a road network G = (V, E) and a query point q on edge $(u, v) \in E$, we define the format of q as the triplet (u, v, λ) such that $\overrightarrow{uq} = \lambda \cdot \overrightarrow{ub}$.

According to Definition 1, q is at the vertex u when $\lambda = 0$, and at v when $\lambda = 1$.

The first phase of our pruning techniques is to collect into a set P those end points of all the edges which the query points in Q are on, and then compute the convex hull of the point set P. Algorithm 3 details this process, where *convexHull*(P) computes the convex hull of the point set P using Andrew's Monotone Chain algorithm [12], which takes $O(|P| \log |P|)$ time.

Algorithm 3 *hullPhase1(Q)*

1: given a query point set Q on a road network G = (V, E)2: $P \longleftarrow \emptyset$

3: for each $q = (u, v, \lambda) \in Q$ do

 $\begin{array}{ll} 4: & P \longleftarrow P \cup u \\ 5: & P \longleftarrow P \cup v \end{array}$

6: return *convexHull*(*P*)

The first phase pruning simply checks the points in Q, and the vertices in the region surrounded by the convex hull computed by Algorithm 3 to find the optimal meeting point. However, this is not sufficient as we are going to illustrate it.

Consider the road network in Figure 3(b) where a bridge crosses over a river. The query points are marked by the triangles. The convex hull returned by Algorithm 3 is the one drawn with dotted lines. It is easy to check that v is the OMP, but it is outside of the region surrounded by the convex hull.

Algorithm 4 hullPhase2(H)
1: given the convex hull $H = (h_1, h_2,, h_{\ell}, h_{\ell+1} = h_1)$ re-
turned by $hullPhase1(Q)$
2: $S \longleftarrow \emptyset$
3: for $i = 1$ to ℓ do
4: Get the shortest path \mathcal{L} between h_i and h_{i+1}
5: for each vertex p on \mathcal{L} do
6: $S \leftarrow S \cup p^{-1}$
7: return <i>convexHull</i> (S)

In order to avoid such false negatives in search space pruning, we propose a second phase of convex hull computation. Suppose that the convex hull returned by *hullPhase1(Q)* is represented as $H = (h_1, h_2, ..., h_\ell, h_1)$ where the points h_i on H are listed in clockwise order, we find the shortest path for each pair of neighboring points on H, insert all the points on these paths into a set S, and then compute the convex hull of S. The process of running the second phase pruning is shown in Algorithm 4, where we use our index file to obtain the shortest path between two vertices in Line 4.

For the previous example in Figure 3(b), the OMP v is now in the region surrounded by the convex hull returned by Algorithm 4, since it is on the shortest path between a to b, which are the two neighboring vertices on the convex hull returned by Algorithm 3.

Let |H| denote the number of vertices on H, and $|\mathcal{L}_{max}|$ be the maximum number of points on the shortest path between two neighboring points on H, then Algorithm 4 takes $O(|H| \cdot |\mathcal{L}_{max}| \cdot \log(|H| \cdot |\mathcal{L}_{max}|))$ time.

Now, we present Algorithm 5 that first performs our two-phase online convex hull computation, and then checks only the query points and the vertices in the region surrounded by the convex hull to find the OMP.

Algorithm 5 Two-Phase Online Convex-Hull-Based Pruning 1: given a query point set Q on a road network G = (V, E)

2: $H \leftarrow hullPhasel(Q)$ 3: $H' \leftarrow hullPhase2(H)$ 4: $opt \leftarrow NULL$ 5: $minCost \leftarrow +\infty$ 6: for each $q \in Q$ do 7: $cost \leftarrow sumOfDistance(q, Q, minCost)$ 8: if cost < minCost then 9: $cost \longleftarrow minCost$ 10: $opt \longleftarrow q$ 11: for each $v \in V$ that is in the region surrounded by H' do 12: $cost \leftarrow sumOfDistance(v, Q, minCost)$ 13: if cost < minCost then 14: $cost \longleftarrow minCost$ $opt \longleftarrow v$ 15: 16: return opt

If we use only the first phase of convex hull computation, Lines 2 and 3 in Algorithm 5 can be replaced by $H' \leftarrow hullPhase1(Q)$.

To support efficient range query evaluation in Line 11 in Algorithm 5, we organize all the vertices in V by a kd-tree. The query window is the minimum bounding box (MBR) of the convex hull H', and for the vertices in the MBR, a refinement step is performed to obtain the vertices that are really in the region surrounded by H'.

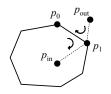


Figure 4: Points Inside/Outside the Convex Hull

We check whether a vertex is inside the region surrounded by the convex hull using the following property: given three points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ and $p_3 = (x_3, y_3)$, let us define $ccw(p_1, p_2, p_3) = (x_2 - x_1) * (y_3 - y_1) - (x_3 - x_1) * (y_2 - y_1)$ (or simply *ccw*). The angle $\overline{p_1 p_2 p_3}$ is in counter-clockwise order if ccw > 0, in clockwise order if ccw < 0, and p_1, p_2 and p_3 are on the same line if ccw = 0. To judge whether a point *p* is inside or on the boundary of the region surrounded by a convex hull H', we have the following theorem: THEOREM 3. A point p is inside or on the boundary of the region surrounded by a convex hull H', if and only if for any edge (p_0, p_1) on H' where p_0 and p_1 are listed in clockwise order, $ccw(p_0, p_1, p) \leq 0$.

Since the result convex hull H' returned by Andrew's Monotone Chain algorithm is a stack of points on H' that are pushed in counter-clockwise order, for some point p, we pop each edge (p_0, p_1) on H' (in clock-wise order) and check $ccw(p_0, p_1, p)$. Figure 4 illustrates the process. As long as we find ccw > 0 for an edge on H', p is outside the region surrounded by H' and thus pruned. Otherwise, we need to check p. Let |H'| denote the number of vertices on H', then this check takes O(|H'|) time.

Algorithm 5 is able to find the OMP on almost all real road networks except for very unusual cases as illustrated in Appendix D. Actually, we have done extensive experiments on the road network datasets of 46 states in US [20], and the dataset of "city of Oldenburg" (OL) [21]. Altogether 1700 randomly generated OMP queries are performed on each dataset. Algorithm 5 only fails to return the OMP in 5 of the 79900 queries in total, and the sum-ofdistances values of these result meeting points are all within 0.1% more than the smallest sum-of-distances value.

3.3 Fast Greedy Algorithm for OMP Queries

Although the convexity property of the "sum of Euclidean distances" function no longer holds for the sum of network distances, the breach of this property is not significant since both types of distances are defined over a metric space.

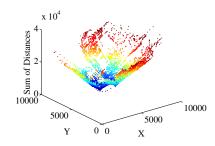


Figure 5: "Sum of Distances" Values at All Network Vertices

For example, Figure 5 shows the values of the sums of network distances at all the points in the road network of the OL dataset for an OMP query. From the shape of the function, we can observe that "convexity" is preserved to some extent. Inspired by this observation and the gradient descent methods of [8, 9], we propose a greedy algorithm as shown in Algorithm 6, where we denote the sum-of-distances value of a vertex u as sd(u), and the set of neighboring vertices of u as NB(u).

Algorithm 6 Greedy Algorithm

- 1: given a query point set Q on a road network G = (V, E)
- 2: Compute the center of gravity of Q as (x_c, y_c)
- 3: Obtain the vertex v_{nn} that is nearest to (x_c, y_c) by a nearest neighbor query on the vertex kd-tree
- 4: $opt \leftarrow v_{nn}$

5: repeat

- 6: $\min \leftarrow \arg \min_{u \in NB(u)} sd(u)$
- 7: **if** sd(min) > sd(opt) **then**
- 8: return opt
- 9: else $opt \leftarrow min$

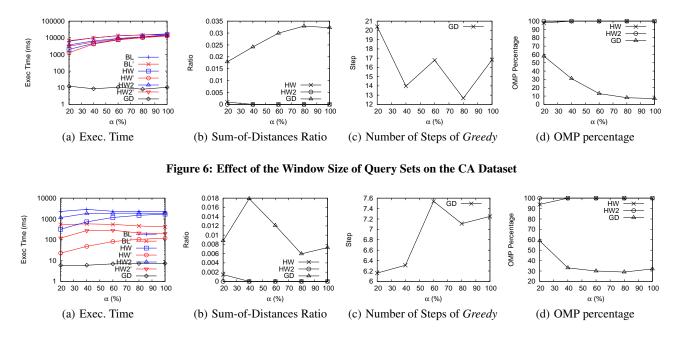


Figure 7: Effect of the Window Size of Query Sets on the OL Dataset

Algorithm 6 first computes the center of gravity (x_c, y_c) of the query point set Q (Line 2), which is the common choice of the initial point for the gradient descent methods of the Weber problem. As our initial point should be a vertex rather than an arbitrary point in the 2D space, Algorithm 6 picks the vertex that is closest to (x_c, y_c) as the initial point (Line 3) by a nearest neighbor query on the vertex kd-tree. In each iteration, Algorithm 6 finds the neighbor *min* of the current vertex *opt* that has the smallest sum of distances among all the neighbors (Line 6). If the neighbor *min* has a smaller sum of distances than the current vertex *opt*, we update the current vertex to be *min* (Line 9). Otherwise, Algorithm 6 terminates and the current point is returned (Line 8).

As we will see in Section 4, although Algorithm 6 may get stuck in a local optimal point (i.e. all its neighbors have larger sums of distances), its sum-of-distances value is very close to the minimum value. Importantly, Algorithm 6 is often able to find the exact OMP and runs orders of magnitude faster than the algorithms described in Sections 3.1 and 3.2. Thus, the algorithm is extremely suitable for large-scale query processing in real time, and the upper bound estimation of sum-of-distances for accelerating location constraint evaluation, such as those applications mentioned in [1].

4. EXPERIMENTS

In this section, we evaluate the performance of our algorithms for the OMP queries by using the road network datasets of 46 states in US from [20], and the datasets of "city of Oldenburg" (OL) and "California" (CA) from [21]. Table 2 in the Appendix describes all the datasets we use, which contains the number of nodes and edges in each dataset. The datasets of the remaining states in US from [20] are not used either because their sizes are too small, or because the datasets are composed of many small connected components. We require that all the vertices belong to the same component since we randomly generate OMP query points on the road networks. Appendix E details the experimental platform configuration and the preprocessing of the datasets.

For each dataset, we generate queries by imposing a rectangular window on the dataset, and all the query points are randomly generated on the part of the road network in the window. Let W denote the distance between the *x*-coordinates of the leftmost vertex and the rightmost vertex on the road network, and H denote the distance between the *y*-coordinates of the highest vertex and the lowest vertex on the road network. We parameterize the size of a window by the parameter $\alpha(< 1)$ so that the window has size $\alpha W \times \alpha H$. Appendix F presents the details of our query generator.

The configuration of a query set Q can be represented as a triple (α, N_p, N_w) , where N_w denotes the number of windows used to generate the query set, N_p denotes the number of query points generated in each window, and α decides the size of each window (which is $\alpha W \times \alpha H$). The total number of query points is $|Q| = N_p \times N_w$. We introduce the parameter N_w to generate query sets whose query points belong to several groups, and the points in each group are close to each other. For each dataset and each query set configuration, we randomly generate 100 query sets for evaluation.

In the sequel, we denote our baseline algorithm as *Baseline* (BL), the algorithm that uses only the first phase of convex-hull-based pruning as *HullWindow* (HW), the one that uses two-phase convex-hull-based pruning as *HullWindow2* (HW2), and the greedy algorithm as *Greedy* (GD).

To fully utilize the pruning in Lines 4–5 of Algorithm 2, we can use the result of *Greedy* to initialize the current minimum sum-ofdistances value of the other algorithms, i.e. Line 3 of Algorithm 1 and Line 5 of Algorithm 5. We denote the algorithm versions that use *Greedy* for initialization by appending the apostrophe to the original algorithm names, e.g. *Baseline* becomes *Baseline*' (BL').

We evaluate the performance of our algorithms based on the following four criteria: (1) query processing time; (2) the sum-ofdistances ratio of the other algorithms to *Baseline*; (3) the number of steps (i.e. iterations) of *Greedy*; and (4) the number of times that each algorithm finds the exact OMP among the 100 queries (for each configuration on each dataset). The reported value of the first three criteria are averaged over the 100 queries.

Due to the space limitation, we only show our results on the "CA" and "OL" datasets from [21], and the overall results of the

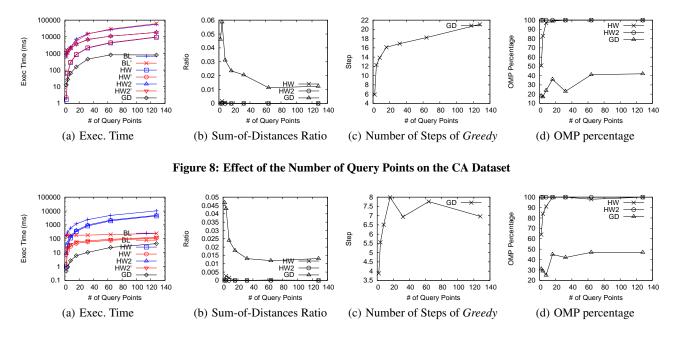


Figure 9: Effect of the Number of Query Points on the OL Dataset

47 datasets in this section. Appendix G shows part of the results on all the datasets, and the complete results are available online 1/2.

4.1 Effect of the Window Size of Query Sets

Figure 6(a) (Figure 7(a)) shows the average execution time of our algorithms (over 100 queries) for different window sizes (determined by α), where we set $N_w = 1$ and $N_p = 20$ for each query. We can see that *Baseline* takes the most time, *HullWindow2* the second most, *HullWindow* the third, and finally *Greedy*. Besides, initialization using *Greedy* does improve the performance of the algorithms, and the improvement on OL is more obvious than that on CA. While the query processing time increases with the increment of window size for the other algorithms, *Greedy* shows rather stable performance for all values of α , and usually takes tens of milliseconds. On the average of the 47 datasets in this set of experiments, *HullWindow2* runs 2.14 times faster than *Baseline*, and *Greedy* runs 142.02 times faster than *HullWindow2*.

Let sd(u, Q) denote the sum-of-distances value of vertex u for query set Q. As *Baseline* guarantees to return the OMP *opt*, for some other algorithm that returns the meeting point v, we define its *sum-of-distances ratio* to be sd(v, Q)/sd(u, Q) - 1. Figure 6(b) (Figure 7(b)) shows the average sum-of-distances ratio of our algorithms. We can see that *HullWindow2* always returns the OMP, while *HullWindow* may return a near-optimal meeting point when $\alpha < 40\%$. The result of *Greedy* is less optimal but the *sum-ofdistances ratio* is always within 3.5%.

Figure 6(c) (Figure 7(c)) shows the average number of iterations run by *Greedy* for different window sizes, and Figure 6(d) (Figure 7(d)) shows the number of times that the algorithms return the exact OMP among the 100 queries. *HullWindow2* always returns the OMP, while *HullWindow* returns the OMP for over 90% of the queries, the percentage of which reaches 100% as α increases to 40%. On the other hand, *Greedy* manages to return the OMP for only a small fraction of the queries, and the percentage goes down with the increment of α .

4.2 Effect of the Number of Query Points

Figures 8 (Figures 9) (a)–(d) show the results of our algorithms for different query set sizes $(N_p = 2^i, i \in \{1, 2, ..., 7\})$. In this set of experiments, we set $N_w = 1$ and $\alpha = 40\%$. From Figures 8 (Figures 9) (a) and (d), we can obtain similar observations as in Section 4.1. We find that *HullWindow2* can be tens of times faster than *Baseline* for small N_p , but the gap reduces to only several times when N_p becomes larger. On the average of the 47 datasets in this set of experiments, *HullWindow2* runs 5.88 times faster than *Baseline*, and *Greedy* runs 54 times faster than *HullWindow2*.

We can see from Figure 8(b) (Figure 9(b)) that the average sumof-distances ratio of Greedy gets smaller as there are more query points in the query set, which stabilizes at between 1% and 1.5% when $|Q| \ge 60$. Figure 8(c) (Figure 9(c)) shows that Greedy requires more iterations when |Q| becomes larger.

4.3 Effect of Multiple Windows

In this set of experiments, we study the scenario where the query points belong to several groups, and the points in each group are close to each other. Specifically, we study the case where there are two groups and each group contains 10 query points. We set $N_w = 2$, $N_p = 10$, and vary the parameter α to see the performance of our algorithms for different window sizes.

Figures 10 (Figures 11) (a)–(d) show the results of our algorithms for different query set sizes (determined by N_p). We find that the performance of all the algorithms are quite stable: As α increases, the query processing time does not increase, the *sum-of-distances* ratio decreases, and the number of iterations of *Greedy* increases. On the average of the 47 datasets in this set of experiments, *HullWindow2* runs 2.51 times faster than *Baseline*, and *Greedy* runs 129.9 times faster than *HullWindow2*.

Among the OMP 79900 queries in total, *HullWindow2* only fails to return the OMP for 5 queries, and the *sum-of-distances ratio* of these result meeting points are all within 0.1%. Therefore, *HullWindow2* is a good alternative to *Baseline*, especially for small

¹http://www.cse.ust.hk/~yanda/datasets/part1.pdf

²http://www.cse.ust.hk/~yanda/datasets/part2.pdf

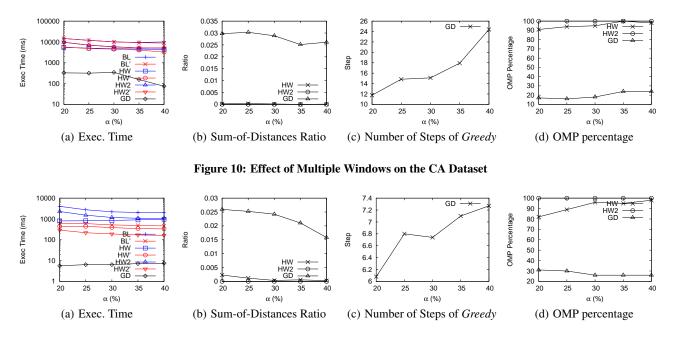


Figure 11: Effect of Multiple Windows on the OL Dataset

query sets. *Greedy* is over two orders of magnitude faster than the other algorithms, and usually takes only tens of milliseconds. Besides, its *sum-of-distances ratio* is usually around 3%. Therefore, the meeting point returned by *Greedy* is of high quality, and *Greedy* is the most practical method to support large-scale meeting point queries on real-world spatial database servers.

5. CONCLUSION

In this paper, we study the *optimal meeting point* query that returns the point on a road network G = (V, E) with the smallest sum of network distances to all the query points in a given query set Q. Our baseline algorithm substantially reduces the search space of the OMP query from $|Q| \cdot |E|$ to |V| + |Q| according to the spatial property established in Theorem 1. We also design an effective two-phase convex-hull-based pruning technique to further prune the search space. Finally, we develop an extremely efficient greedy algorithm to find a high-quality near-optimal meeting point instead of an exact OMP. The efficiency of this algorithm makes it the most practical choice for spatial applications that involve large flow of queries and require fast response as the top priority.

Acknowledgements. This work is partially supported by RGC GRF under grant number HKUST 617610.

6. REFERENCES

- [1] Z. Xu and H.-A. Jacobsen. "Processing Proximity Relations in Road Networks". In *SIGMOD*, 2010.
- [2] D. Papadias, Q. Shen, Y. Tao and K. Mouratidis. "Group Nearest Neighbor Queries". In *ICDE*, 2004.
- [3] M. L. Yiu, N. Mamoulis and D. Papadias. "Aggregate Nearest Neighbor Queries in Road Networks". In *TKDE*, 2005.
- [4] H. Samet, J. Sankaranarayanan and H. Alborzi. "Scalable Network Distance Browsing in Spatial Databases". In *SIGMOD*, 2008.

- [5] D. Papadias, J. Zhang, N. Mamoulis and Y. Tao. "Query Processing in Spatial Network Databases". In VLDB, 2003.
- [6] Z. Xu and H.-A. Jacobsen. "Efficient Constraint Processing for Location-aware Computing". In MDM, 2005.
- [7] H. Hu, D. L. Lee and V. C. S. Lee. "Distance Indexing on Road Networks". In VLDB, 2006.
- [8] L. Cooper. "An Extension of the Generalized Weber Problem". *Journal of Regional Science*, vol. 8, issue 2, pp. 181–197, Dec. 1968.
- [9] L. Cooper. "The Mulitfacility Location Problem: Applications and Descent Theorems". *Journal of Regional Science*, vol. 17, issue 3, pp. 409–419, Dec. 1977.
- [10] R. Chen. "Solution of Location Problems with Radial Cost Functions". *Computers & Mathematics with Applications*, vol. 10, issue 1, pp. 87–94, Dec. 1984.
- [11] R. Chen. "Location Problems with Costs Being Sums of Powers of Euclidean Distances". *Computers & Mathematics with Applications*, vol. 10, issue 1, pp. 87–94, Dec. 1984.
- [12] F. P. Preparata and M. I. Shamos. "Computational Geometry: An Introduction". *Springer-Verlag*, 1985.
- [13] N. Beckmann, H. Kriegel, R. Schneider and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
- [14] G. Hjaltason and H. Samet. "Distance Browsing in Spatial Databases". In *TODS*, 1999.
- [15] Y. Tao, D. Papadias and Q. Shen. "Continuous Nearest Neighbor Search". In VLDB, 2002.
- [16] M. R. Kolahdouzan and C. Shahabi. "Voronoi-Based k Nearest Neighbor Search for Spatial Network Databases". In *VLDB*, 2004.
- [17] H. Cho and C. Chung. "An Efficient and Scalable Approach to CNN Queries in a Road Network". In VLDB, 2005.
- [18] K. Mouratidis, M. L. Yiu, D. Papadias and N. Mamoulis. "Continuous Nearest Neighbor Monitoring in Road Networks". In VLDB, 2006.
- [19] Z. Chen, H. T. Shen, X. Zhou and J. X. Yu. "Monitoring Path

Nearest Neighbor in Road Networks". In SIGMOD, 2009.

[20] http://data.geocomm.com/catalog/US

[21] http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

APPENDIX

A. PROOF OF THEOREM 2

Theorem 2: Given a point set $Q = \{q_1, q_2, \ldots, q_n\}$ on an arbitrary graph G = (V, E), where each point q_i is associated with a weight w_i . If all the weights are integers or rational numbers, then $V \cup Q$ must contain the point $\overline{x} = \arg \min_x \sum_i w_i \cdot d(\overline{q_i, x})$.

PROOF. It is straightforward to convert the rational number weights into integer weights with the same weight distribution among all the points in Q. For example, suppose $Q = \{q_1, q_2, q_3\}$, $w_1 = 0.15$, $w_2 = 1.11$ and $w_3 = 0.8$, then we can re-assign the weights to be $w_1 = 15$, $w_2 = 111$ and $w_3 = 80$. Clearly, this transformation does not change the result point \overline{x} .

Now, let us assume that all the weights are integers, and we replace each point q_i with w_i new points at the same location of q_i , each of which has weight 1. The resulting new query point set Q' can be treated as unweighted, and thus $Q' \cup V$ contains \overline{x} according to Theorem 1. It is straightforward to see that the transformation from Q to Q' does not change the result point \overline{x} , and the locations in Q' is exactly the locations in Q. Therefore, Theorem 2 is proved. \Box

B. SHORTEST PATH INDEX BETWEEN VER-TICES

Since the definition of OMP on a road network is directly built on shortest paths, we construct a disk-based index file to accelerate the shortest path computation between two vertices on the road network. Shortest path computation is actually the basic operation of many queries on the road network [1, 4, 7, 5, 19]. While a lot of efficient online algorithms have been proposed [5, 19] for shortest path computation on road networks, [4] presents a nice off-line index for shortest path computation on road networks. As road networks seldomly change, off-line indices are usually viable and more effective in shortening query execution time.

As shortest path computation is not our main focus, we simply run Dijkstra algorithm for each vertex in the road network, to compute $d(\overline{v_1, v_2})$ for each pair of vertices v_1 and v_2 . The result is a 2D array, which is written to a file. During query processing, to obtain $d(\overline{v_1, v_2})$, we set the file pointer to the location where it is stored and read the value. Therefore, only one I/O operation is required to obtain $d(\overline{v_1, v_2})$.

To find the shortest path between two vertices, we maintain another disk-based index which is also built by the Dijkstra algorithm, together with the shortest path length index mentioned above. Besides the length of the shortest path from vertex v_1 to v_2 , the Dijkstra algorithm also reports the vertex before v_2 on the shortest path from v_1 to v_2 , which can be used to find the shortest path by going back iteratively. We store another 2D array *Path* on the disk, which is appended at the end of shortest path length index in the *index* file, where $Path[v_1][v_2]$ stores the first vertex after v_1 that is on the shortest path from v_1 to v_2 . Algorithm 7 shows the way to find the shortest path \mathcal{L} between v_1 and v_2 , which takes ℓ I/O operations, where ℓ denotes the length of \mathcal{L} .

Note that the more sophisticated methods for shortest path computation mentioned in the beginning of Appendix B can also be used, since our algorithms for the OMP query do not have specific requirement on the shortest path computation.

Algorithm 7 shortestPath (v_1, v_2)

- 1: given two vertices v_1 and v_2
- $2 : \ \mathcal{L} \longleftarrow \emptyset$
- 3: $v \longleftarrow v_1$
- 4: while $v \neq v_2$ do
- 5: Append v to \mathcal{L}
- 6: $v \leftarrow Path[v][v_2]$
- 7: return \mathcal{L}

C. SHORTEST PATH COMPUTATION BE-TWEEN ARBITRARY POINTS

The length of the shortest path from a query point q on edge (u, v) to a vertex p is computed as $d(\overline{q, p}) = \min\{d(q \sim u) + d(\overline{u, p}), d(q \sim v) + d(\overline{v, p})\}$, which requires two I/O operations from the index.

The length of the shortest path from a query point q on edge (u, v) to another query point q' on edge (u', v') is computed as $d(\overline{q,q'}) = \min\{d(q \sim u) + d(\overline{u,u'}) + d(u' \sim q'), d(q \sim u) + d(\overline{u,v'}) + d(v' \sim q'), d(q \sim v) + d(\overline{v,v'}) + d(u' \sim q'), d(q \sim v) + d(\overline{v,v'}) + d(v' \sim q')\}, if q and q' are on different edges, which requires four I/O operations from the index.$

If q and q' are on the same edge, then according to the triangular inequality of the road network edges, the shortest path length $d(\overline{q,q'}) = d(q \sim q')$, which requires no I/O operation.

D. COUNTEREXAMPLES

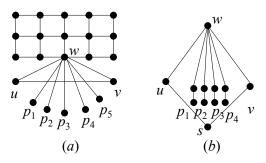


Figure 12: Counterexamples

Figure 12(a) shows an example where the road network partitioning scheme of [1] fails to obtain the OMP. Let us assume that a query point is at each vertex in the set $S = \{u, p_1, p_2, p_3, p_4, p_5, v\}$, and that the edge (u, v) cuts the graph into two partitions. Since all the query points belong to one partition (below (u, v)), the partitioning scheme of [1] will not check the vertex w (above (u, v)). However, by simple reasoning, we can see that w is the OMP.

Our two-phase online convex-hull-based pruning technique is able to find the OMP for the above example. This is because p_1 and p_2 are two neighboring points on the convex hull of the point set S, and the shortest path between them is (p_1, w, p_2) . As a result, w will be included in the second phase for further checking.

However, our technique may also have some limitation in some extreme cases. Consider the example road network in Figure 12(b), where we assume that a query point is at each vertex in the set $S' = \{u, p_1, p_2, p_3, p_4, v, s\}$. In the first phase pruning, we first obtain the convex hull of S', which contains u, s and v. In the second phase pruning, w will then not be included as none of the three shortest paths between the points in $\{u, v, s\}$ goes through w. As a result, w is not checked by our technique, although it is easy

data	vertex #	edge #									
CA	21048	21693	FL	6879	7303	LA	5886	6133	NY	7503	7766
OL	6105	7035	GA	6757	7038	MA	1530	1595	OH	4738	4986
AR	7454	7695	IA	4833	5168	MD	1222	1270	OK	6877	7300
AZ	11050	11381	ID	8108	8310	ME	4536	4656	PA	6384	6619
CO	9796	10175	IL	6117	6520	MI	5928	6198	RI	203	212
CT	923	958	IN	4300	4592	MN	7718	8184	SC	2868	2966
DE	250	257	KS	5615	6054	MO	9500	9889	SD	6733	7165
DR	10513	10738	KY	4745	4952	MS	6022	6297			

Table 2: DataSet Description

to check that w is the OMP.

We assume in the above example that the query point at u is represented as on an edge other than (u, w) (See Definition 1), e.g. (u, s), since otherwise, w will be included in the first phase. There is a similar assumption for v. Note that the intermediate point(s) between p_i and w is also important for the construction of the counterexample, since otherwise, p_i is represented as on edge (p_1, w) and w will be included. As real road networks are usually of regular structure rather than the weird topology as given in Figure 12(b), in this sense we claim that our two-phase online convex-hull-based pruning technique is able to find the OMP on almost all real road networks.

E. EXPERIMENTAL SETTING AND DATASET PREPROCESSING

All the experiments are done on a computer with Intel(R) Core(TM) i5 CPU and 4GB memory. All our programs are written in JAVA, and run in Eclipse on Windows 7 Enterprise.

We use the road network datasets of 46 states in US from [20], and the datasets of "city of Oldenburg" (OL) and "California" (CA) from [21] for experiments. As the "CA" dataset is contained in both sources, we only use the one from [21] for the experiment. Since we randomly generate OMP query points on the road networks, if two query points are in two different connected components, they can never reach each other and the OMP does not exist. Therefore, we require that all the vertices belong to the same component, and do not use the datasets of the remaining states in US from [20] that are composed of many small connected components.

While the datasets from [21] are directly usable, the datasets from [20] are in E00 file format. We use the "Global Mapper" software³ to convert the datasets into readable raw data files. After merging the duplicate vertices, we get the largest connected components of each dataset (which usually contains over 99% of the original vertices). The preprocessed datasets are available from

http://www.cse.ust.hk/~yanda/datasets/roadData.rar

F. QUERY GENERATOR

To generate a query point in a window, we find all the edges of the road network that intersect with the window, and randomly pick an edge from them. After picking the edge, we randomly generate a query point on the segment of the edge that is in the window.

To generate a query set for window parameter α , we randomly position a $\alpha W \times \alpha H$ rectangular window in the whole 2D space for the road network. If there are less than ϵ edges covered by the window, we drop it and generate a new window. This process is repeated until a window with at least ϵ edges is generated, and then we generate |Q| query points using the method described in the

previous paragraph. The parameter ϵ is used to avoid generating a window in a sparse area of the space of the road network, and is set to 20 throughout our experiments.

G. ADDITIONAL EXPERIMENTAL RESULTS

Due to the space limitation, in Section 4, we only show our experimental results on two of the 47 datasets, i.e. CA and OL. As the complete results still contain too many entries to be put in the appendix, we make them available online:

• The running time of our various algorithms and the number of steps executed by *Greedy* under different query configurations on all the datasets are recorded at:

http://www.cse.ust.hk/~yanda/datasets/part1.pdf

• The *sum-of-distance ratio* of our various algorithms and the percentage of queries for which each algorithm returns the exact OMP (OMP percentage) under different query configurations on all the datasets are recorded at:

http://www.cse.ust.hk/~yanda/datasets/part2.pdf

In the sequel, we show part of our results on *sum-of-distance* ratio and *OMP percentage* for all the datasets in all the 3 set of experiments we conduct. In the following tables, r(.) in the table heads denotes the *sum-of-distance ratio* of the algorithm, and o(.) denotes the *OMP percentage* of the algorithm.

- Table 3 shows our results for the experimental setting in Section 4.1, where we only show the results for $\alpha = 20\%$ and $\alpha = 100\%$.
- Table 4 shows our results for the experimental setting in Section 4.2, where we only show the results for $N_p = 2$ and $N_p = 128$.
- Table 5 shows our results for the experimental setting in Section 4.3, where we only show the results for $\alpha = 20\%$ and $\alpha = 40\%$.

³http://www.globalmapper.com

Table 3: Effect of the Window Size of Query Sets

Table 4:	Effect of t	the Num	ber of Q	Query 1	Points
----------	-------------	---------	----------	---------	--------

data α $r(HW)$ $r(HW$	
CA 100% 0% 3.225% 100% 100% 7% OL 100% 0,147% 0% 0,732% 100% 100% 23 AR 100% 0% 0,734% 100% 100% 23 AZ 20% 0,75% 0,76% 1,215% 100% 100% 22 CO 20% 0,76% 0,76% 1,215% 100% 100% 23 CO 20% 0,76% 0,76% 1,25% 100% 100% 69 CT 20% 0,02% 0% 1,353% 99% 100% 66 DE 20% 0,03% 0% 1,439% 95% 100% 63 DR 20% 0,03% 0% 1,325% 97% 100% 63 CA 20% 0,03% 0% 1,325% 95% 100% 643 DA 20% 0,325% 10% 100% 63 1,55% <td< td=""><td></td></td<>	
CL 20% 0.14% 0% 0.73% 94% 100% 599 AR 20% 0% 0.73 100% 100% 669 AR 100% 0.96 0.13745 100% 100% 669 AZ 20% 0.018% 0% 1.525% 90% 100% 699 CO 20% 0.6% 0.67% 1.558% 100% 100% 697 CO 100% 0.62% 0.64% 1.455% 99% 100% 669 CT 100% 0.021% 0.64% 1.455% 99% 100% 669 DR 0.026 0.038 0.5% 1.5258 100% 100% 669 CA 0.036 0.63 0.75 1.5258 100% 100% 459 A 100% 0.032% 0.75 1.3254 100% 100% 439 L 100% 0.103% 0.5% 1.3254 100% <	
OL 100% 0.9% 0.732% 100% 100% 232 AR 100% 0.9% 0.732% 100% 100% 223 AZ 200% 0.9% 0.2156% 100% 100% 223 Z 100% 0.9% 0.067% 1.155% 100% 100% 203 CO 100% 0.9% 0.087% 1.258% 100% 100% 669 CT 100% 0.9% 0.034% 100% 100% 669 DR 20% 0.031% 0% 0.041% 100% 669 DR 20% 0.033% 0% 1.192% 9%% 100% 663 DR 20% 0.033% 0% 1.333% 100% 673 GA 20% 0.033% 0% 1.192% 9%% 100% 673 GA 20% 0.032% 0% 1.335% 100% 100% 633 IA 100%	
AR 20% 0% 1.374% 100% 100% 669 AZ 20% 0.018% 0% 1.525% 90% 100% 227 AZ 100% 0.95 0.067% 1.558% 100% 99% 557 CO 100% 0.02% 0.358% 96% 100% 203 CD 0.02% 0.04% 0.9% 0.01% 99% 100% 669 DE 0.00% 0.02% 0.358% 96% 100% 669 DE 0.00% 0.02% 1.439% 95% 100% 669 DE 1.00% 0.03% 0.9% 1.03% 95% 100% 669 L 0.00% 0.03% 0.5% 1.03% 90% 100% 669 L 0.00% 0.03% 0.5% 1.03% 90% 100% 669 L 100% 0.013% 0.5% 1.03% 100% 100% 100% <th< td=""><td></td></th<>	
AR 100% 00% 0% 1.55% 9% 100% 100% 227 CO 20% 0% 0.067% 1.55% 100% 100% 297 CO 20% 0% 0.067% 1.55% 100% 100% 297 CO 100% 0.067% 0.338% 96% 100% 697 CT 100% 0.004% 0.614% 99% 100% 648 DE 100% 0.6 0.644% 100% 100% 636 DR 0.03% 0% 1.439% 95% 100% 675 CA 0.03% 0% 1.554% 97% 100% 675 IA 100% 0.02% 0% 1.554% 98% 100% 638 ID 100% 0.02% 0% 1.554% 98% 100% 638 ID 100% 0.02% 0% 1.554% 98% 100% 638	
AZ 20% 0.018% 0% 1.525% 96% 100% 229 CO 20% 0% 0.067% 1.538% 100% 99% 550 CO 100% 0% 0.358% 96% 100% 209 CT 100% 0.021% 0% 0.031% 99% 100% 666 DE 100% 0.021% 0% 0.034% 100% 100% 666 DR 100% 0.033% 0% 1.439% 95% 100% 667 GA 0.033% 0% 1.349% 95% 100% 677 GA 0.0303% 0% 1.3524% 100% 100% 673 GA 0.0303% 0% 1.3524% 100% 100% 643 IA 0.0303% 0% 1.3544% 100% 100% 438 ID 0.002% 0% 1.3544% 100% 100% 143 IL 20%<	
AZ 100% 0% 0% 0.124% 100% 100% 20% CO 100% 0% 0.667% 1.558% 100% 100% 0.09% CT 100% 0.02% 0.95% 0.38% 96% 100% 0.99% DE 100% 0.004% 0% 0.644% 100% 60% DE 100% 0% 0.644% 100% 60% 1.649% 100% 60% DR 100% 0% 0.444% 100% 100% 63% DR 00% 0.58 1.585% 100% 100% 57% DR 0.00% 0% 1.585% 100% 100% 455 CA 100% 0.002% 0% 1.546 98% 100% 643 DA 100% 0.002% 0% 1.546 98% 100% 143 DA 0.002% 0% 1.576 97% 100% 633	
CO 20% 0.06% 1.558% 100% 99% 555 CT 20% 0.02% 0% 2.028% 100% 100% 2.09 CT 100% 0.021% 0% 0.611% 99% 100% 666 DE 100% 0.03 0% 1.439% 95% 100% 667 DR 0.03 0.03 0% 1.649% 100% 100% 647 DR 0.03 0% 0.585% 97% 100% 647 CA 0.03 0% 1.554% 98% 100% 677 GA 100% 0.032% 0% 1.354% 100% 100% 459 IA 100% 0.002% 0% 1.354% 100% 100% 459 ID 100% 0.002% 0% 1.354% 100% 100% 149 IL 100% 0.002% 0% 1.324% 100% 100% 101%	
CO 100% 00% 0% 2.082% 100% 100% 609 CT 100% 0.024% 0% 0.384% 96% 100% 699 CT 100% 0.021% 0% 0.601% 99% 100% 699 DE 100% 0% 0.601% 99% 100% 695 DR 20% 0.03% 0% 1.439% 98% 100% 636 DR 00% 0.032% 0% 1.395% 100% 675 L0 0.032% 0% 1.554% 98% 100% 619 L1 20% 0.027% 0% 1.554% 99% 100% 635 L1 100% 0.0026 0% 1.318% 100% 100% 635 L1 100% 0.0026 0% 1.215% 99% 100% 637 L1 100% 0.0026 0% 1.215% 95% 100% 105	
CT 20% 0.02% 0% 0.38% 96% 100% 69% DE 20% 0.021% 0% 0.601% 99% 100% 666 DR 100% 0.03% 0% 1.192% 9%% 100% 666 DR 100% 0.03% 0% 1.439% 95% 100% 667 TL 100% 0.03% 0% 1.535% 97% 100% 667 GA 100% 0.03 0% 1.532% 100% 100% 677 GA 100% 0.03 0% 1.354% 98% 100% 673 GA 100% 0.03 0% 1.354% 100% 100% 243 D 0.00 0.079% 0.5 1.130% 100% 149 IL 100% 0.002% 0% 1.554% 99% 100% 163 IL 100% 0.002% 0% 1.215% 99%	
CT 100% 0.004% 0% 1.459% 99% 100% 646 DE 100% 0.03 0% 0.0814% 100% 100% 645 DR 20% 0.03% 0% 1.1924% 98% 100% 645 DR 100% 0% 1.439% 95% 100% 57% FL 100% 0% 0.325% 10% 105% 10% 57% GA 20% 0.032% 0% 1.554% 98% 100% 643 IA 100% 0% 0.554% 99% 100% 583 ID 100% 0.002% 0% 1.37% 100% 100% 583 IL 20% 0.002% 0% 1.21% 99% 100% 635 IL 100% 0.002% 0% 1.21% 95% 100% 635 IL 100% 0.002% 0% 1.21% 9% 100% <t< td=""><td></td></t<>	
DE 20% 0.01% 0.601% 99% 100% 66% DR 20% 0.03% 0% 1.192% 98% 100% 649 DR 100% 0% 0.63% 1649% 100% 100% 649 DR 100% 0% 0.85% 97% 100% 649 GA 100% 0.032% 0% 1.352% 100% 100% 677 GA 100% 0.032% 0% 1.354% 98% 100% 661 IA 20% 0.032% 0% 1.354% 99% 100% 424 ID 100% 0.002% 0% 1.354% 99% 100% 63 IL 100% 0.002% 0% 1.554% 98% 100% 163 IS 100% 0.002% 0% 1.215% 97% 100% 649 IL 20% 0.074% 1.148% 100% 100% 149	
DE 100% 0% 0% 1.02% 9% 100% 657 DR 100% 0% 1.649% 100% 0% 366 FL 20% 0.03% 0% 1.439% 95% 100% 376 GA 20% 0.032% 0% 1.325% 100% 100% 475 GA 20% 0.325% 0% 1.554% 98% 100% 455 IA 20% 0.039% 0% 1.554% 98% 100% 455 IL 20% 0.002% 0% 1.57% 99% 100% 645 IL 20% 0.002% 0% 1.191% 100% 100% 645 IN 100% 0% 0.577 97% 100% 643 IN 100% 0% 1.187% 100% 100% 183 KY 100% 0% 1.215% 95% 100% 163 LA	76
DR 20% 0.03% 0% 1.192% 98% 100% 649 DR 100% 0% 1.439% 95% 100% 369 GA 20% 0.032% 0% 1.352% 97% 100% 677 GA 100% 0% 0.532% 100% 100% 677 GA 100% 0.032% 0% 1.352% 100% 100% 673 GA 0.039 0% 1.354% 98% 100% 613 IA 20% 0.02% 0% 1.306% 100% 100% 588 ID 100% 0.002% 0% 1.57% 99% 100% 613 IL 100% 0.006% 0% 1.318% 100% 109% 138 IN 20% 0.0074 0% 1.215% 97% 100% 595 IN 100% 0.066% 0.5% 1.215% 97% 100% 595	70
DR 100% 0% 0% 1.649% 100% 100% 577 FL 100% 0% 0.532% 0% 1.358% 100% 100% 375 GA 20% 0.032% 0% 1.352% 100% 100% 455 IA 20% 0.032% 0% 1.534% 98% 100% 64 IA 20% 0.179% 0% 1.021% 90% 100% 244 ID 20% 0.022% 0% 1.01% 99% 100% 655 IL 100% 0% 0% 1.78% 100% 100% 653 IN 100% 0% 0% 1.318% 100% 100% 598 KY 20% 0.027% 0% 1.215% 95% 100% 598 KY 20% 0.074% 0% 1.807% 100% 100% 598 KY 20% 0.074% 0% 1.4	70
FL 100% 0% 1.439% 95% 100% 57% GA 20% 0.032% 0% 1.352% 100% 100% 679 GA 100% 0% 0% 1.352% 100% 100% 679 GA 100% 0.030% 0% 1.554% 98% 100% 619 ID 0.06% 0.002% 0% 1.011% 100% 100% 588 ID 100% 0.002% 0% 1.59% 99% 100% 659 IL 20% 0.002% 0% 1.57% 99% 100% 659 IN 20% 0.069% 0% 1.215% 95% 100% 108 KY 20% 0.072% 0% 1.215% 95% 100% 599 KY 20% 0.072% 0% 1.215% 95% 100% 599 LA 20% 0.066% 0.6426% 100% 100%	70
FL 100% 0% 1.43% 95% 100% 57% GA 20% 0.032% 0% 1.352% 100% 100% 679 GA 100% 0% 0% 1.352% 100% 100% 679 GA 100% 0.030% 0% 1.354% 98% 100% 613 ID 0.0% 0.017% 0% 1.0121K 90% 100% 588 ID 100% 0.002% 0% 1.59% 99% 100% 659 IL 100% 0% 0% 1.59% 99% 100% 659 IN 100% 0% 0% 1.215% 95% 100% 699 IN 100% 0% 0% 1.215% 95% 100% 599 KY 100% 0% 0% 1.215% 95% 100% 599 KY 100% 0% 0% 1.4180% 100% 100%	70
FL 100% 0% 0.5% 1.358% 100% 100% 40% GA 100% 0% 0% 1.352% 100% 100% 459 IA 20% 0.039% 0% 1.534% 98% 100% 459 ID 20% 0.179% 0% 1.021% 90% 100% 549 ID 100% 0.002% 0% 1.59% 99% 100% 669 IL 100% 0% 0% 1.59% 99% 100% 663 IN 100% 0% 0% 1.59% 100% 100% 675 IN 100% 0% 0% 1.13% 100% 100% 59 IN 100% 0% 0.072% 0% 1.13% 100% 100% 59 IX 100% 0% 0.074% 0% 1.14% 100% 100% 59 IA 100% 0% 0.016%	
GA 20% 0.032% 0% 1.985% 97% 100% 6% IA 20% 0.039% 0% 1.352% 100% 100% 645 IA 100% 0% 0% 1.354% 98% 100% 100% 645 ID 20% 0.179% 0% 1.354% 98% 100% 645 IL 20% 0.002% 0% 1.59% 99% 100% 655 IL 100% 0% 0% 1.78% 100% 100% 619 IN 100% 0% 0% 1.597% 97% 100% 619 KS 100% 0% 0% 1.215% 95% 100% 619 LA 20% 0.074% 0% 1.265% 100% 100% 65 LA 20% 0.106% 0% 1.445% 100% 100% 58 MD 20% 0.106% 0% 1.445% <td></td>	
GA 100% 0% 0% 1.352% 100% 100% 619 IA 100% 0% 0% 1.554% 98% 100% 649 ID 20% 0.179% 0% 1.021% 90% 100% 243 ID 100% 0.002% 0% 1.554% 99% 100% 649 IL 20% 0.002% 0% 1.59% 99% 100% 639 IN 20% 0.% 0% 1.597% 97% 100% 643 KS 100% 0% 0% 1.215% 95% 100% 599 KY 20% 0.074% 0% 1.189% 97% 100% 653 LA 100% 0% 0.666% 95% 100% 699 MA 100% 0% 0.666% 95% 100% 663 LA 100% 0% 0.666% 95% 100% 669	
IA 20% 0.039% 0% 1.554% 9% 100% 6.4 ID 10% 0.179% 0% 1.306% 100% 104% 249 ID 100% 0.002% 0% 3.571% 99% 100% 168 IL 100% 0.002% 0% 1.597% 99% 100% 653 IL 100% 0% 0% 1.597% 97% 100% 633 IN 100% 0% 0% 1.517% 90% 100% 633 KS 100% 0% 0% 1.215% 95% 100% 163 KY 20% 0.072% 0% 1.215% 95% 100% 159 LA 20% 0.106% 0% 1.425% 97% 100% 635 LA 20% 0.106% 0% 1.445% 100% 100% 75 MA 20% 0.016% 0% 1.445% 1	
IA 100% 0% 0% 1.306% 100% 20% ID 20% 0.179% 0% 1.021% 90% 100% 589 IL 20% 0.002% 0% 1.55% 99% 100% 169 IL 100% 0% 0% 1.191% 100% 100% 169 IN 20% 0.069% 0% 1.597% 97% 100% 169 KS 100% 0.074% 0.6 1.318% 100% 100% 159 KY 100% 0.074% 0% 1.139% 97% 100% 59 LA 100% 0.074% 0% 1.807% 100% 100% 59 LA 100% 0% 0.666% 95% 100% 69 59 LA 100% 0% 0.666% 95% 100% 69 59 ML 20% 0.164% 0% 1.645% 100% 100	
D 20% 0.179% 0% 1.021% 90% 100% 149 IL 100% 0.002% 0% 1.59% 99% 100% 165 IL 100% 0% 0% 1.91% 100% 100% 165 IL 100% 0% 0% 1.73% 100% 100% 633 IN 100% 0% 0.69% 1.597% 97% 100% 167 KS 100% 0% 0.606 0.606 1.818% 100% 100% 159 LA 20% 0.074% 0% 1.807% 100% 100% 755 MA 100% 0% 0% 1.448% 100% 755 MA 100% 0% 0.606 956 100% 792 MA 100% 0% 0.606 956 100% 792 MA 100% 0% 0.606 956 100% 1006 792 <td></td>	
D 100% 0.002% 0% 3.571% 99% 100% 149 IL 20% 0% 0% 1.59% 99% 100% 659 IN 20% 0% 0% 1.78% 100% 100% 639 IN 100% 0% 0% 1.597% 97% 100% 619 KS 100% 0% 0% 1.215% 95% 100% 639 KY 100% 0% 0% 1.215% 95% 100% 557 LA 20% 0.074% 0% 1.807% 100% 100% 579 MA 20% 0.106% 0% 0.425% 94% 100% 588 MD 20% 0.106% 0% 0.666% 95% 100% 599 MI 20% 0.014% 0% 1.254% 100% 100% 599 MI 100% 0% 0% 1.445% 100%	
IL 20% 0.002% 0% 1.59% 99% 100% 655 IL 100% 0% 0% 1.191% 100% 100% 657 IN 100% 0% 0% 2.254% 100% 100% 677 KS 20% 0.069% 0% 1.517% 97% 100% 657 KY 20% 0.072% 0% 1.215% 95% 100% 559 LA 20% 0.016% 0% 1.807% 100% 557 LA 20% 0.016% 0% 0.425% 94% 100% 558 MD 20% 0.109% 0% 1.144% 97% 100% 559 MD 20% 0.109% 0% 1.245% 97% 100% 559 ME 20% 0.109% 0% 1.244% 97% 100% 559 MI 20% 0.019% 0% 1.443% 99% <td< td=""><td></td></td<>	
IL 100% 0% 1.191% 100% 10% 199 IN 20% 0% 0% 1.78% 100% 100% 639 IN 100% 0% 0% 1.597% 97% 100% 163 KS 100% 0% 0% 1.215% 95% 100% 599 KY 20% 0.072% 0% 1.215% 95% 100% 599 LA 100% 0% 0% 1.89% 100% 100% 579 MA 20% 0.016% 0% 0.425% 94% 100% 579 MA 100% 0% 0% 1.645% 100% 100% 583 MD 100% 0% 0.616% 0.55% 100% 100% 599 MI 100% 0% 0.645% 100% 100% 593 MI 100% 0% 0.75% 100% 100% 593	
IN 20% 0% 1.78% 100% 100% 633 IN 100% 0% 0% 2.254% 100% 100% 633 KS 100% 0% 0% 1.318% 100% 100% 187 KY 20% 0.072% 0% 1.215% 95% 100% 1597 LA 20% 0.074% 0% 1.887% 97% 100% 55% LA 20% 0.106% 0% 0.425% 94% 100% 55% MA 100% 0% 0.666 95% 100% 393 ME 20% 0.109% 0% 1.144% 97% 100% 255 MI 20% 0.134% 0% 1.234% 97% 100% 643 MN 20% 0.031% 0% 0.834% 99% 100% 643 MN 100% 0% 1.434% 97% 100% 643 <t< td=""><td></td></t<>	
IN 100% 0% 2.24% 100% 10% 17 KS 100% 0% 0% 1.597% 97% 100% 161 KY 20% 0.072% 0% 1.215% 95% 100% 599 LA 20% 0.074% 0% 1.189% 97% 100% 659 LA 100% 0% 0% 1.498% 100% 100% 579 MA 100% 0% 0.425% 94% 100% 759 MA 100% 0% 0.425% 94% 100% 759 MD 100% 0% 0.425% 94% 100% 729 MD 20% 0.109% 0% 1.444% 97% 100% 599 MI 20% 0.144% 0% 1.254% 97% 100% 664 MN 100% 0% 0.834% 99% 100% 669 MN 00%	
SS 20% 0.069% 0% 1.597% 97% 100% 619 KS 100% 0% 0% 1.318% 100% 100% 619 KY 20% 0.072% 0% 1.215% 100% 100% 159 LA 20% 0.074% 0% 1.807% 100% 100% 557 LA 20% 0.016% 0% 0.425% 94% 100% 558 MA 20% 0.016% 0% 0.666% 95% 100% 588 MD 20% 0.016% 0% 1.245% 97% 100% 599 MI 20% 0.031% 0% 2.758% 100% 100% 699 MN 100% 0% 0.4746% 100% 100% 699 MN 20% 0.031% 0% 1.434% 99% 100% 699 MN 20% 0.001% 0% 1.4343% 99%	
KS 100% 0% 1.318% 100% 100% 188 KY 20% 0.072% 0% 1.215% 95% 100% 1599 LA 20% 0.074% 0% 1.189% 97% 100% 659 LA 100% 0% 0% 1.807% 100% 100% 657 MA 100% 0% 0.425% 94% 100% 757 MA 100% 0% 0.445% 100% 100% 399 MD 20% 0.016% 0% 1.645% 100% 100% 399 ME 100% 0% 0.725% 100% 100% 595 MI 20% 0.144% 0% 1.254% 97% 100% 643 MN 20% 0.0140% 0% 1.659% 100% 609 MI 100% 0% 0.8334% 99% 100% 609 MN 20% 0	
KY 20% 0.072% 0% 1.215% 95% 100% 599 KY 100% 0% 0.6% 2.6% 100% 100% 159 LA 100% 0% 0.74% 0% 1.189% 100% 100% 579 MA 20% 0.106% 0% 0.425% 94% 100% 579 MD 20% 0.016% 0% 0.606% 95% 100% 589 MD 100% 0% 0% 1.645% 100% 100% 255 MI 20% 0.144% 0% 1.254% 97% 100% 649 MN 100% 0% 0.834% 99% 100% 649 MN 20% 0.031% 0% 1.443% 97% 100% 649 MS 20% 0.058% 0% 1.013% 100% 649 MS 100% 0% 1.438% 100% 100%	
KY 100% 0% 0.6% 1.6% 100% 100% 15% LA 100% 0% 0.1180% 100% 100% 100% 577 MA 100% 0% 0.425% 94% 100% 758 MA 100% 0% 0.666% 95% 100% 100% 488 MD 20% 0.106% 0% 1.645% 100% 100% 399 ME 100% 0% 0% 1.254% 97% 100% 598 MI 100% 0% 0% 1.254% 97% 100% 649 MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 0% 1.659% 100% 100% 169 MS 100% 0% 0% 1.815% 89% 100% 649 MS 100% 0% 0% 1.815% 100% 100%	
LA 20% 0.074% 0% 1.189% 97% 100% 655 LA 100% 0% 0.807% 100% 100% 575 MA 20% 0.106% 0% 0.425% 94% 100% 575 MA 100% 0% 0.606% 95% 100% 589 MD 100% 0% 0.616% 00% 1.144% 97% 100% 729 ME 100% 0% 0.758% 100% 100% 599 MI 20% 0.144% 0% 4.746% 100% 100% 649 MN 20% 0.031% 0% 0.834% 99% 100% 609 MN 20% 0.058% 0% 1.134% 97% 100% 649 MS 20% 0.478% 0% 1.815% 89% 100% 539 MT 100% 0% 0% 1.443% 99% 100% <t< td=""><td></td></t<>	
LA 100% 0% 1.807% 100% 100% 579 MA 20% 0.106% 0% 0.425% 94% 100% 759 MD 20% 0.016% 0% 0.666% 95% 100% 589 MD 100% 0% 0.645% 100% 100% 389 ME 20% 0.109% 0% 1.244% 97% 100% 599 ME 20% 0.144% 0% 1.254% 97% 100% 649 MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 1.659% 100% 100% 169 MO 20% 0.001% 0% 1.834% 99% 100% 649 MT 20% 0.478% 0% 1.835% 99% 100% 649 MS 100% 0% 0% 1.831% 97% 100% 539	
MA 20% 0.106% 0% 0.425% 94% 100% 759 MA 100% 0% 0.439% 100% 100% 449 MD 20% 0.016% 0% 1.645% 100% 100% 399 ME 20% 0.109% 0% 1.144% 97% 100% 599 MI 20% 0.144% 0% 1.254% 97% 100% 694 MN 20% 0.031% 0% 4.746% 100% 100% 109 MN 100% 0% 0.6534% 99% 100% 604 MN 100% 0% 0% 1.013% 97% 100% 749 MS 100% 0% 0% 1.815% 89% 100% 593 MT 100% 0% 0% 1.815% 897 100% 593 MT 100% 0% 0% 1.815% 896 100% 069 </td <td>%</td>	%
MA 20% 0.106% 0% 0.425% 94% 100% 75% MD 20% 0.016% 0% 1.448% 100% 100% 449 MD 20% 0.016% 0% 1.645% 100% 100% 399 ME 20% 0.016% 0% 1.254% 97% 100% 599 MI 20% 0.0144% 0% 1.254% 97% 100% 643 MN 20% 0.031% 0% 4.746% 100% 104 107 MO 20% 0.031% 0% 1.655% 100% 100% 643 MN 20% 0.01% 0% 1.483% 99% 100% 643 MS 100% 0% 0% 1.655% 100% 100% 643 MS 100% 0.017% 0% 1.313% 100% 100% 100% 100% 100% 100% 100% 100% 100% <td>70</td>	70
MA 100% 0% 1.498% 100% 100% 449 MD 20% 0.016% 0% 0.660% 95% 100% 389 ME 20% 0.109% 0% 1.144% 97% 100% 729 ME 100% 0% 0% 2.758% 100% 100% 259 MI 20% 0.144% 0% 1.254% 97% 100% 669 MN 100% 0% 0% 0.834% 99% 100% 669 MN 100% 0% 0.834% 99% 100% 669 MO 20% 0.001% 0% 1.83% 100% 100% 67 MS 20% 0.058% 0% 1.815% 89% 100% 539 MT 100% 0% 0% 1.311% 93% 100% 597 ND 20% 0.078% 0% 1.314% 100% 100% 597<	
MD 20% 0.016% 0% 0.606% 95% 100% 58 MD 100% 0% 0% 1.645% 100% 100% 399 ME 100% 0% 0.758% 100% 100% 225 MI 100% 0% 0.758% 100% 100% 599 MI 20% 0.031% 0% 0.834% 99% 100% 649 MN 20% 0.001% 0% 1.659% 100% 100% 699 MO 20% 0.001% 0% 1.483% 99% 100% 699 MS 20% 0.038% 0% 1.613% 97% 100% 249 MT 20% 0.478% 0% 1.815% 89% 100% 533 MT 100% 0% 0% 1.311% 93% 100% 20% NC 100% 0% 0% 1.548% 100% 100% 56	
MD 100% 0% 0% 1.645% 100% 10% 399 ME 20% 0.109% 0% 1.144% 97% 100% 72 MI 20% 0.144% 0% 1.254% 97% 100% 599 MI 100% 0% 0.834% 99% 100% 649 MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 0.658% 0% 1.659% 100% 100% 649 MN 100% 0% 0.96% 1.999% 100% 100% 649 MS 20% 0.058% 0% 1.013% 97% 100% 649 MT 20% 0.478% 0% 1.815% 89% 100% 395 NC 100% 0% 0% 1.311% 93% 100% 565 ND 100% 0% 0% 1.314% 100%	
ME 20% 0.109% 0% 1.144% 97% 100% 729 ME 100% 0% 0% 2.758% 100% 100% 599 MI 20% 0.144% 0% 1.254% 97% 100% 599 MI 100% 0% 0.834% 99% 100% 607 MN 20% 0.001% 0% 1.659% 100% 100% 607 MO 20% 0.001% 0% 1.483% 99% 100% 607 MO 100% 0% 1.815% 89% 100% 749 MS 100% 0% 1.815% 89% 100% 533 MT 100% 0% 0% 1.815% 89% 100% 599 NC 100% 0% 0% 1.341% 100% 100% 279 ND 100% 0% 0% 1.342% 96% 100% 609	
ME 100% 0% 2.758% 100% 100% 259 MI 20% 0.144% 0% 1.254% 97% 100% 599 MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 0.4746% 100% 100% 649 MN 20% 0.001% 0% 1.483% 99% 100% 649 MO 100% 0% 0.97 1.00% 100% 649 MS 20% 0.058% 0% 1.013% 97% 100% 649 MT 20% 0.478% 0% 1.815% 89% 100% 399 NC 100% 0% 0.717% 0% 1.311% 93% 100% 539 NC 100% 0% 0.717% 0% 1.311% 100% 100% 297 ND 20% 0.078% 0% 1.744% 100%	
MI 20% 0.144% 0% 1.254% 97% 100% 599 MN 20% 0.031% 0% 4.746% 100% 100% 643 MN 20% 0.031% 0% 0.834% 99% 100% 643 MN 100% 0% 0% 1.659% 100% 100% 643 MO 100% 0% 0.99% 100% 100% 643 MS 20% 0.058% 0% 1.013% 97% 100% 744 MS 100% 0% 0.833% 100% 100% 303 NC 20% 0.177% 0% 1.311% 93% 100% 309 NC 100% 0% 0% 1.444% 100% 100% 206 ND 20% 0.078% 0% 1.314% 93% 100% 309 NE 100% 0% 0.546% 97% 100% 309	
MI 100% 0% 0% 4.746% 100% 100% 109% MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 1.659% 100% 100% 669 MO 20% 0.001% 0% 1.483% 99% 100% 669 MO 100% 0% 0.584% 0% 1.013% 97% 100% 669 MS 100% 0% 0.682 4522% 100% 100% 243 MT 20% 0.478% 0% 1.815% 89% 100% 599 NC 100% 0% 0% 1.311% 93% 100% 609 ND 100% 0% 0% 1.548% 100% 100% 199 NE 20% 0.078% 0% 1.548% 100% 100% 199 NH 100% 0% 0% 1.713% 100	
MN 20% 0.031% 0% 0.834% 99% 100% 649 MN 100% 0% 0.6 1.659% 100% 100% 699 MO 100% 0% 0.7 1.483% 99% 100% 699 MS 20% 0.058% 0% 1.013% 97% 100% 749 MS 100% 0% 0.6 2.452% 100% 100% 539 MT 100% 0.478% 0% 1.815% 89% 100% 539 NC 20% 0.177% 0% 1.311% 93% 100% 569 ND 20% 0.77% 0% 1.449% 100% 100% 569 ND 20% 0.078% 0% 1.342% 96% 100% 199 NE 20% 0.007% 0% 0.546% 100% 100% 139 NH 20% 0.003% 0% 0.187%	
MN 100% 0% 1.659% 100% 100% 179 MO 20% 0.001% 0% 1.483% 99% 100% 609 MO 100% 0% 0% 1.999% 100% 100% 744 MS 20% 0.058% 0% 1.013% 97% 100% 744 MS 100% 0% 0% 1.815% 89% 100% 533 MT 100% 0% 0% 1.811% 93% 100% 599 NC 100% 0% 0% 1.149% 100% 100% 569 ND 100% 0% 0% 1.342% 96% 100% 609 NE 100% 0% 0% 1.342% 96% 100% 199 NH 20% 0.007% 0% 0.546% 97% 100% 769 NH 20% 0.003% 0% 2.381% 100% 100%	
MO 20% 0.001% 0% 1.483% 99% 100% 609 MO 100% 0% 0% 1.999% 100% 100% 9% MS 100% 0% 0% 1.013% 97% 100% 749 MT 20% 0.478% 0% 1.815% 89% 100% 539 MT 100% 0% 0% 1.311% 93% 100% 539 NC 100% 0% 0% 1.311% 93% 100% 569 ND 20% 0% 0% 1.548% 100% 100% 199 NE 20% 0.007% 0% 1.342% 96% 100% 199 NH 20% 0.007% 0% 0.546% 97% 100% 1419 NJ 20% 0.003% 0% 0.862% 99% 100% 100% 1639 NV 100% 0% 0% 1.846%	
MO 100% 0% 0.99% 100% 100% 9% MS 20% 0.058% 0% 1.013% 97% 100% 749 MS 100% 0% 0% 1.815% 89% 100% 533 MT 20% 0.478% 0% 1.815% 89% 100% 539 NC 20% 0.177% 0% 1.311% 93% 100% 599 NC 100% 0% 0% 1.44% 100% 100% 565 ND 20% 0.78% 0% 1.342% 96% 100% 609 NE 100% 0% 0.07% 0.546% 97% 100% 73 NH 20% 0.003% 0% 0.862% 99% 100% 769 NJ 100% 0% 0% 1.817% 97% 100% 769 NM 20% 0.003% 0% 2.831% 100% 100% <td></td>	
MS 20% 0.058% 0% 1.013% 97% 100% 749 MS 100% 0% 0% 2.452% 100% 100% 249 MT 20% 0.478% 0% 1.815% 89% 100% 339 MT 100% 0% 0% 1.815% 89% 100% 339 NC 20% 0.177% 0% 1.311% 93% 100% 599 ND 20% 0% 0% 1.548% 100% 100% 599 NE 20% 0.078% 0% 1.548% 100% 100% 199 NE 100% 0% 0% 0.546% 97% 100% 769 NJ 20% 0.093% 0% 0.862% 99% 100% 769 NJ 100% 0% 0% 2.351% 100% 100% 269 NM 100% 0% 0% 2.351% 100%	
MS 100% 0% 0.478% 0% 1.815% 89% 100% 249 MT 20% 0.478% 0% 1.815% 89% 100% 539 MT 100% 0% 0% 1.811% 93% 100% 509 NC 20% 0.177% 0% 1.311% 93% 100% 569 ND 20% 0% 0% 1.744% 100% 100% 569 ND 100% 0% 0% 1.713% 100% 100% 699 NE 20% 0.078% 0% 1.342% 96% 100% 699 NH 20% 0.007% 0% 0.5466% 97% 100% 739 NH 100% 0% 0% 2.381% 100% 100% 265 NJ 100% 0% 0% 2.381% 100% 100% 329 NV 20% 0.257% 0% 1.227% </td <td></td>	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
NC 20% 0.177% 0% 1.311% 93% 100% 599 NC 100% 0% 0% 1.744% 100% 100% 279 ND 20% 0% 0% 1.548% 100% 100% 569 ND 100% 0% 0% 1.548% 100% 100% 699 NE 20% 0.078% 0% 1.342% 96% 100% 609 NE 100% 0% 0% 0.546% 97% 100% 739 NH 20% 0.007% 0% 0.187% 97% 100% 709 NJ 20% 0.003% 0% 2.831% 100% 100% 269 NM 20% 0.003% 0% 2.053% 100% 100% 329 NV 20% 0% 0% 1.227% 93% 100% 649 NY 20% 0.257% 0% 1.227% 93% <td></td>	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	70
ND 20% 0% 0% 1.149% 100% 100% 569 ND 100% 0% 0% 1.548% 100% 100% 69 NE 20% 0.078% 0% 1.342% 96% 100% 69 NE 100% 0% 0% 1.713% 100% 100% 199 NH 20% 0.007% 0% 0.546% 97% 100% 739 NH 100% 0% 0% 1.832% 96% 100% 419 NJ 20% 0.003% 0% 0.862% 99% 100% 709 NM 20% 0.003% 0% 2.053% 100% 100% 633 NV 100% 0% 0% 1.227% 93% 100% 649 NY 20% 0.257% 0% 1.227% 93% 100% 649 NY 100% 0% 0% 2.739% 100%	70
ND 100% 0% 0.548% 100% 100% 199 NE 20% 0.078% 0% 1.342% 96% 100% 609 NE 100% 0% 0% 1.713% 100% 100% 609 NH 20% 0.007% 0% 0.546% 97% 100% 419 NJ 20% 0.093% 0% 0.187% 97% 100% 769 NJ 100% 0% 0% 2.381% 100% 100% 769 NM 20% 0.003% 0% 0.862% 99% 100% 709 NM 100% 0% 0% 1.227% 93% 100% 619 NY 20% 0.257% 0% 1.227% 93% 100% 649 OH 100% 0% 0% 2.348% 100% 649 OH 20% 0.04% 0% 1.281% 98% 100% 649 </td <td>70</td>	70
ND 100% 0% 0.548% 100% 100% 199 NE 20% 0.078% 0% 1.342% 96% 100% 609 NE 100% 0% 0% 1.713% 100% 100% 609 NH 20% 0.007% 0% 0.546% 97% 100% 419 NJ 20% 0.093% 0% 0.187% 97% 100% 769 NJ 100% 0% 0% 2.381% 100% 100% 769 NM 20% 0.003% 0% 0.862% 99% 100% 709 NM 100% 0% 0% 1.227% 93% 100% 619 NY 20% 0.257% 0% 1.227% 93% 100% 649 OH 100% 0% 0% 2.348% 100% 649 OH 20% 0.04% 0% 1.281% 98% 100% 649 </td <td>70</td>	70
NE 20% 0.078% 0% 1.342% 96% 100% 609 NE 100% 0% 0% 1.713% 100% 100% 199 NH 20% 0.007% 0% 0.546% 97% 100% 739 NH 100% 0% 0% 0.187% 97% 100% 769 NJ 20% 0.003% 0% 0.862% 99% 100% 769 NM 20% 0.003% 0% 2.053% 100% 100% 639 NV 20% 0% 0% 2.127% 93% 100% 649 NY 100% 0% 0% 2.348% 100% 100% 109 NY 100% 0% 0% 2.799% 100% 100% 259 OK 20% 0.101% 0% 0.864% 97% 100% 639 OK 100% 0% 0% 0.719% 100%	70
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
NH 20% 0.007% 0% 0.546% 97% 100% 739 NH 100% 0% 0% 1.929% 100% 100% 419 NJ 20% 0.093% 0% 0.187% 97% 100% 769 NJ 100% 0% 0% 2.381% 100% 100% 769 NM 20% 0.003% 0% 0.862% 99% 100% 709 NM 100% 0% 0% 2.053% 100% 100% 329 NV 100% 0% 0% 3.179% 100% 100% 329 NY 20% 0.257% 0% 1.221% 93% 100% 649 OH 20% 0.04% 0% 1.281% 98% 100% 599 OK 20% 0.101% 0% 0.864% 97% 100% 599 OK 100% 0% 0% 0.337% 94% <td>70</td>	70
NH 100% 0% 0% 1.929% 100% 100% 419 NJ 20% 0.093% 0% 0.187% 97% 100% 769 NJ 100% 0% 0% 2.381% 100% 100% 769 NM 20% 0.003% 0% 0.862% 99% 100% 709 NM 100% 0% 0% 2.053% 100% 100% 633 NV 20% 0.257% 0% 1.227% 93% 100% 649 NY 20% 0.257% 0% 1.2217% 93% 100% 649 OH 100% 0% 0% 2.348% 100% 649 OH 100% 0% 0% 1.2217% 93% 100% 649 OH 100% 0% 0.644% 07% 100% 599 OK 100% 0% 0.799% 100% 100% 599	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	
NJ 100% 0% 0.862% 99% 100% 269 NM 20% 0.003% 0% 0.862% 99% 100% 709 NM 100% 0% 0% 2.053% 100% 100% 709 NV 20% 0% 0% 2.053% 100% 100% 639 NV 20% 0% 0% 1.846% 100% 100% 639 NY 20% 0.257% 0% 1.227% 93% 100% 649 OH 100% 0% 0% 2.348% 100% 649 OH 100% 0% 0% 2.799% 100% 649 OH 100% 0% 0% 1.29% 100% 100% 219 PA 20% 0.011% 0% 0.937% 94% 100% 549 RI 100% 0% 0% 1.237% 100% 100% 549	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	%
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	%
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	76
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	76
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
UT 20% 0.318% 0% 1.709% 92% 100% 57% UT 100% 0% 0% 1.56% 100% 100% 24% VA 20% 0.054% 0% 1.094% 95% 100% 57% VA 100% 0% 0.054% 0% 1.952% 100% 57% VA 100% 0% 0% 1.952% 100% 100% 24% VT 20% 0.067% 0% 0.386% 95% 100% 73% VT 100% 0% 0% 2.134% 100% 100% 45% WA 20% 0.039% 0% 1.462% 96% 100% 62% WA 100% 0% 0% 3.397% 100% 100% 8%	
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$	
VA 20% 0.054% 0% 1.094% 95% 100% 579 VA 100% 0% 0% 1.952% 100% 100% 249 VT 20% 0.067% 0% 0.386% 95% 100% 739 VT 100% 0% 0% 2.134% 100% 100% 459 WA 20% 0.039% 0% 1.462% 96% 100% 629 WA 100% 0% 0% 3.397% 100% 100% 8%	
VA 100% 0% 0% 1.952% 100% 100% 249 VT 20% 0.067% 0% 0.386% 95% 100% 739 VT 100% 0% 0% 2.134% 100% 100% 459 WA 20% 0.039% 0% 1.462% 96% 100% 629 WA 100% 0% 0% 3.397% 100% 100% 8%	
VT 20% 0.067% 0% 0.386% 95% 100% 73% VT 100% 0% 0% 2.134% 100% 100% 45% WA 20% 0.039% 0% 1.462% 96% 100% 62% WA 100% 0% 0% 3.397% 100% 8%	
VT 100% 0% 0% 2.134% 100% 100% 459 WA 20% 0.039% 0% 1.462% 96% 100% 629 WA 100% 0% 0% 3.397% 100% 8%	
WA 20% 0.039% 0% 1.462% 96% 100% 62% WA 100% 0% 0% 3.397% 100% 100% 8%	
WA 100% 0% 3.397% 100% 100% 8%	
WI 20% 0.008% 0% 0.901% 99% 100% 769	
WI 100% 0% 1.727% 100% 100% 229	
WY 20% 0.035% 0% 1.17% 97% 100% 739	
WY 100% 0% 0% 2.068% 100% 100% 209	
TX 20% 0.088% 0% 1.78% 97% 100% 45%	%
TX 100% 0% 0% 1.632% 100% 100% 139	76

		4: Effect					
data CA	$\frac{N_p}{2}$	$\begin{array}{c} r(HW) \\ 0\% \end{array}$	$\frac{r(HW2)}{0\%}$	r(GD) 4.62%	o(HW) 51%	o(HW2) 100%	o(GD) 18%
CA	128	0%	0%	1.231%	100%	100%	42%
OL	2	0%	0%	4.671%	64%	100%	31%
OL AR	128 2	0% 0%	0% 0%	1.311% 3.942%	100% 57%	100% 100%	47% 18%
AR	128	0%	0%	1.764%	100%	100%	48%
AZ	2	0%	0%	5.112%	56%	100%	15%
AZ CO	128 2	0% 0%	$0\% \\ 0\%$	2.203% 3.568%	100% 47%	100% 100%	48% 15%
СО	128	0%	0%	1.951%	100%	100%	45%
CT CT	2 128	0% 0.004%	0% 0%	6.89% 0.84%	68% 99%	100% 100%	42% 72%
DE	2	0.004%	0% 0%	0.84% 728.276		100%	72% 70%
DE	128	0%	0%	0.094%	100%	100%	56%
DR DR	2 128	0% 0%	0% 0%	3.643% 1.267%	57% 100%	100% 100%	12% 58%
FL	2	0%	0%	3.042%	49%	100%	23%
FL	128	0.054%	0%	1.593%	99%	100%	45%
GA GA	2 128	0% 0%	0% 0%	2.883% 1.354%	57% 100%	100% 100%	16% 50%
IA	2	0%	0%	2.751%	65%	100%	34%
IA ID	128 2	0% 0%	0% 0%	0.909% 8.708%	100% 50%	100% 100%	54% 18%
ID	128	0.412%	0%	1.487%	88%	100%	54%
IL	2	0%	0%	4.753%	66%	100%	24%
IL IN	128 2	0% 0%	0% 0%	1.687% 4.089%	100% 66%	100% 100%	47% 28%
IN	128	0%	0%	2.222%	100%	100%	42%
KS KS	2 128	0% 0%	0% 0%	2.152% 0.837%	66% 100%	100% 100%	23% 52%
KY	2	0%	0%	2.623%	100 <i>%</i> 59%	100%	32% 22%
KY	128	0.088%	0%	1.881%	95%	100%	48%
LA LA	2 128	0% 0.035%	0% 0%	21.9% 1.155%	63% 97%	100% 100%	24% 54%
MA	2	0%	0%	2.955%	61%	100%	30%
MA	128	0.001%	0%	0.424%	99%	100%	77%
MD MD	2 128	0% 0%	0% 0%	2.334% 0.665%	59% 100%	100% 100%	35% 64%
ME	2	0%	0%	3.832%	68%	99%	21%
ME MI	128 2	0% 0%	0% 0%	1.305% 3.576%	100% 56%	100% 100%	59% 28%
MI	128	0.046%	0%	1.068%	99%	100%	28% 55%
MN	2	0%	0%	2.751%	64%	100%	23%
MN MO	128 2	0.018%	0% 0%	1.828% 7.438%	99% 56%	100% 100%	40% 13%
MO	128	0%	0%	2.482%	100%	100%	32%
MS MS	2 128	0% 0%	0% 0%	6.329% 2.15%	51% 100%	100% 100%	20% 45%
MT	2	0%	0%	2.13% 3.046%	100% 51%	100%	43% 8%
MT	128	0%	0%	1.845%	100%	100%	39%
NC NC	2 128	0% 0.021%	0% 0%	4.936% 1.32%	65% 99%	100% 100%	26% 53%
ND	2	0%	0%	4.965%	70%	100%	31%
ND	128	0%	0%	0.8%	100%	100%	54%
NE NE	2 128	0% 0%	$0\% \\ 0\%$	2.548% 0.997%	58% 100%	100% 100%	23% 51%
NH	2	0%	0%	1.454%	55%	100%	40%
NH NJ	128 2	0% 0%	0% 0%	0.336% 3.946%	100% 71%	100% 100%	81% 42%
NJ	128	0%	0%	0.626%	100%	100%	71%
NM	2	0%	0%	3.594%	47%	100%	18%
NM NV	128 2	0% 0%	0% 0%	1.602% 3.234%	100% 54%	100% 100%	42% 14%
NV	128	0%	0%	1.372%	100%	100%	55%
NY NY	2 128	0% 0.009%	0% 0%	4.904% 1.131%	58% 98%	100% 100%	27% 43%
OH	2	0%	0%	2.82%	59%	100%	26%
OH	128	0% 0%	0%	2.128% 2.071%	100%	100%	47% 24%
OK OK	2 128	0%	0% 0%	2.071% 0.6%	61% 100%	100% 100%	24% 49%
PA	2	0%	0%	3.725%	55%	100%	21%
PA RI	128 2	0% 0%	0% 0%	1.342% 8.71%	100% 86%	100% 100%	55% 70%
RI	128	0%	0%	0.315%	100%	100%	71%
SC	2	0%	0%	3.999%	62%	100%	30%
SC SD	128 2	0% 0%	0% 0%	0.872% 3.862%	100% 68%	100% 100%	58% 22%
SD	128	0%	0%	0.524%	100%	100%	60%
TN TN	2 128	0% 0%	0% 0%	3.5% 0.781%	47% 100%	100% 100%	12% 53%
UT	2	0%	0%	3.943%	49%	100%	14%
UT	128	0%	0%	1.473%	100%	100%	44%
VA VA	2 128	0% 0.113%	0% 0%	17.958% 0.52%	55% 96%	100% 100%	27% 70%
VT	2	0%	0%	8.68%	55%	100%	31%
VT WA	128 2	0.005% 0%	0% 0%	0.996% 5.358%	99% 45%	100% 100%	74% 10%
WA	128	0.002%	0% 0%	1.563%	4 <i>3%</i> 99%	100%	46%
WI	2	0%	0%	6.235%	51%	100%	28%
WI WY	128 2	0% 0%	0% 0%	1.051% 4.05%	100% 55%	100% 100%	52% 14%
WY	128	0%	0%	1.834%	100%	100%	49%
TX TX	2 128	0% 0%	0% 0%	3.841% 1.563%	43% 100%	100% 100%	7% 26%
іЛ	120	070	070	1.30370	100 /0	100 /0	2070

	,	Table 5: 1	Effect of	Multipl	e Windo	ws	
data	α	r(HW)	r(HW2)		o(HW)	o(HW2)	o(GD)
CA	20%	0.023%	0%	2.974%	91%	100%	17%
CA	40%	0.012%	0%	2.615%	98%	100%	24%
OL	20%	0.23%	0%	2.594%	82%	100%	31%
OL	40%	0.028%	0%	1.583%	98%	100%	26%
AR	20%	0.033%	0%	1.789%	94%	100%	34%
AR	40%	0%	0%	1.41%	100%	100%	45%
AZ	20%	0.08%	0%	3.424%	95%	100%	25%
AZ	40%	0%	0%	2.088%	100%	100%	34%
CO	20%	0.01%	0%	3.246%	95%	100%	18%
CO	40%	0%	0%	2.42%	100%	100%	34%
CT	20%	0.321%	0%	1.819%	70%	100%	37%
CT	40%	0.201%	0%	2.103%	89%	100%	59%
DE	20%	0%	0%	0.103%	93%	100%	64%
DE	40%	0%	0%	0.164%	100%	100%	70%
DR	20%	0.017%	0%	2.154%	94%	100%	33%
DR	40%	0%	0%	2.245%	100%	100%	42%
FL	20%	0.237%	0%	2.95%	70%	100%	20%
FL	40%	0.13%	0%	2.531%	85%	100%	19%
GA	20%	0.058%	0%	2.554%	95%	100%	31%
GA	40%	0%	0%	1.989%	100%	100%	42%
IA	20%	0.026%	0%	1.876%	95%	100%	26%
IA	40%	0.020 %	0%	0.925%	100%	100%	40%
ID	20%	0.335%	0%	3.103%	63%	100%	19%
ID	40%	0.448%	0%	3.011%	81%	100%	29%
IL	20%	0.067%	0%	2.188%	96%	100%	23%
IL	40%	0%	0%	2.217%	100%	100%	28%
IN	20%	0.038%	0%	1.887%	95%	100%	28%
IN	40%	0.002%	0%	1.76%	99%	100%	35%
KS	20%	0.022%	0%	1.506%	93%	100%	19%
KS	40%	0%	0%	1.001%	100%	100%	37%
KY	20%	0.223%	0%	2.915%	85%	100%	34%
KY	40%	0.019%	0%	2.033%	99%	100%	42%
LA	20%	0.212%	0%	3.598%	68%	100%	22%
LA	40%	0.041%	0%	2.312%	92%	100%	38%
MA	20%	0.097%	0%	1.42%	84%	100%	28%
MA	40%	0.013%	0%	1.736%	99%	100%	43%
MD	20%	0.013%	0%	1.263%	83%	100%	41%
MD	40%	0.22170	0%	2.04%	100%	100%	51%
ME	20%	0.062%	0%	2.597%	88%	100%	28%
ME	40%	0%	0%	1.485%	100%	100%	48%
MI	20%	0.191%	0%	5.801%	65%	100%	24%
MI	40%	0.226%	0%	3.97%	78%	100%	28%
MN	20%	0.039%	0%	1.956%	94%	100%	22%
MN	40%	0%	0%	1.709%	100%	100%	36%
MO	20%	0.046%	0%	3.844%	87%	100%	12%
MO	40%	0%	0%	3.094%	100%	100%	24%
MS	20%	0.034%	0%	2.742%	95%	100%	22%
MS	40%	0%	0%	2.212%	100%	100%	38%
MT	20%	0.063%	0%	2.582%	90%	100%	19%
MT	40%	0%	0%	2.178%	100%	100%	27%
NC	20%	0.13%	0%	2.842%	82%	100%	18%
NC	40%	0.06%	0%	2.195%	96%	100%	42%
ND	20%	0.0010	0%	1.684%	94%	100%	30%
ND	40%	0.02170	0%	1.083%	100%	100%	44%
NE	20%	0.019%	0%	2.185%	97%	100%	21%
	40%			1.53%		100%	40%
NE		0%	0%		100%		
NH	20%	0.131%	0%	2.838%	86%	100%	32%
NH	40%	0.016%	0%	1.699%	98%	100%	50%
NJ	20%	0.09%	0%	1.436%	86%	100%	36%
NJ	40%	0%	0%	2.208%	100%	100%	45%
NM	20%	0.047%	0%	2.632%	95%	100%	18%
NM	40%	0%	0%	2.052%	100%	100%	31%
NV	20%	0.031%	0%	2.689%	92%	100%	21%
NV	40%	0%	0%	2.146%	100%	100%	48%
NY	20%	0.014%	0%	2.324%	93%	100%	25%
NY	40%	0%	0%	1.457%	100%	100%	34%
OH	20%	0.045%	0%	2.919%	90%	100%	23%
OH	40%	0.006%	0%	3.057%	99%	100%	28%
OK	20%	0.026%	0%	1.648%	90%	100%	33%
OK	40%	0%	0%	1.204%	100%	100%	32%
PA	20%	0.134%	0%	2.301%	87%	100%	28%
PA	40%	0%	0%	2.032%	99%	100%	33%
RI	20%	0.078%	0%	1.078%	88%	100%	62%
RI	40%	0.007%	0%	0.474%	98%	100%	63%
SC	20%	0.023%	0%	2.615%	94%	100%	29%
SC	40%	0%	0%	1.059%	100%	100%	53%
SD	20%	0.005%	0%	1.056%	97%	100%	36%
SD	40%	0.005 /0	0%	1.459%	100%	100%	39%
TN	20%	0.089%	0%	2.229%	79%	100%	19%
TN	40%	0.009%	0%	2.039%	99%	100%	37%
UT	20%	0.107%	0%	2.039%	99% 92%	100%	26%
					92% 99%		
UT VA	40%	0.008%	0%	1.66%		100%	36%
VA	20%	0.025%	0%	2.133%	78%	100%	21%
VA	40%	0%	0%	1.719%	96%	100%	43%
VT	20%	0.09%	0%	3.28%	86%	100%	42%
VT	40%	0.036%	0%	1.905%	98%	100%	56%
WA	20%	0.225%	0%	3.791%	73%	100%	18%
WA	40%	0.007%	0%	2.159%	97%	100%	26%
WI	20%	0.107%	0%	2.226%	89%	100%	36%
WI	40%	0%	0%	1.522%	100%	100%	44%
	20%	0.071%	0%	2.65%	91%	100%	34%
WY		0.001%	0%	2.077%	99%	100%	33%
WY WY	40%	0.001%					
WY WY TX	40%	0.001%	0%	1.891%	92%	100%	12%