Approximation Algorithms for the Minimum Bends Traveling Salesman Problem (Extended Abstract)

Cliff Stein*
David P. Wagner*
Dartmouth College Computer Science Technical Report TR2000-367

May 9, 2000

Abstract

The problem of traversing a set of points in the order that minimizes the total distance traveled (traveling salesman problem) is one of the most famous and well-studied problems in combinatorial optimization. It has many applications, and has been a testbed for many of the must useful ideas in algorithm design and analysis. The usual metric, minimizing the total distance traveled, is an important one, but many other metrics are of interest.

In this paper, we introduce the metric of minimizing the number of turns in the tour, given that the input points are in the Euclidean plane. To our knowledge this metric has not been studied previously. It is motivated by applications in robotics and in the movement of other heavy machinery: for many such devices turning is an expensive operation. We give approximation algorithms for several variants of the traveling salesman problem for which the metric is to minimize the number of turns. We call this the *minimum bend traveling salesman problem*.

For the case of an arbitrary set of n points in the Euclidean plane, we give an $O(\lg z)$-approximation algorithm, where z is the maximum number of collinear points. In the worst case z can be as big as n, but z will often be much smaller. For the case when the lines are restricted to being either horizontal or vertical, we give a 2-approximation algorithm. If we have the further restriction that no two points are allowed to have the same x- or y-coordinate, we give an algorithm that finds a tour which makes at most two turns more than the optimal tour. Thus we have an approximation algorithm with an additive, rather than a multiplicative error bound. Beyond the additive error bound, our algorithm for this problem introduces several interesting algorithmic techniques for decomposing sets of points in the Euclidean plane that we believe to be of independent interest.

*Dartmouth College, Department of Computer Science {cliff,dwagn}@cs.dartmouth.edu. Research partially supported by NSF Career Award CCR-9624828, NSF Grant EIA-98-02068, a Dartmouth Fellowship, and an Alfred P. Sloane Foundation Fellowship.

1 Introduction

The problem of traversing a set of points in the order that minimizes the total distance traveled (traveling salesman problem) is one of the most famous and well-studied problems in combinatorial optimization [17]. It has many applications [20, 11, 12, 21, 15], and has been a testbed for many of the most useful ideas in algorithm design and analysis. The usual metric, minimizing the total distance traveled, is an important one, but many other metrics including maximum distance [16, 10], minimum latency(traveling repairman problem) [1, 6], minimizing the shortest edge length[17], maximum scatter TSP [4], and minimizing the total angle traversed [2] are also of interest.

In this paper, we introduce the metric of minimizing the number of turns in the tour, given that the input points are in the Euclidean plane. Equivalently, given a set of points in the plane, we wish to find a tour through the points, consisting of straight lines, so that the number of lines is minimized. To our knowledge this metric has not been studied previously. It is motivated by applications in robotics and in the movement of other heavy machinery: for many such devices turning is an expensive operation. Imagine that a robot needs to visit a set of locations distributed over a relatively small physical space (room or building sized). If the locations are fairly dense throughout the region, there will be many tours whose total length is close to the minimum. If this robot is slow to turn, the actual time spent visiting all the points will be dominated by the number of turns that are made.

Further applications appear in VLSI, especially with the 2-layer chip model. This model allows vertical wires to be placed on one layer, and horizontal wires on another. Turns in a circuit are made by connecting the two layers, but each connection introduces resistance. Thus minimizing the number of turns is a desirable objective[19].

Over larger geographic areas, both distance and number of turns will contribute to the total time spent traversing a tour. We view our work in this paper as an important first step towards understanding this more general problem [22]. Metrics involving minimizing the number of turns or some function of the number of turns and total distance are well studied for the shortest paths problem [19, 25, 26, 18, 23, 24].

For the regular traveling salesman problem, Christofedes' algorithm gives a $3/2$-approximation [7], and the algorithm of Arora [5] gives a PTAS for the problem in the case when the points are in the Euclidean plane. Objectives such as the traveling repairman problem, maximum length tour and maximum scatter also have constant-factor approximation schemes [16, 4, 6] and, for longest tour, in the case when the points are in R^d for some fixed d, the problem is actually solvable in polynomial time. In contrast, for the angular-metric TSP, the best known approximation algorithm is $O(\log n)$ [2].

In this paper, we give approximation algorithms for several variants of the traveling salesman problem for which the metric is to minimize the number of turns. For the case of an arbitrary set of n points in the Euclidean plane, we give an $O(\lg z)$-approximation algorithm, where z is the maximum number of collinear points. In the worst case z can be as big as n, but z will often be much smaller. We call this problem the *minimum bends traveling salesman problem*. We also study interesting restricted cases and find better approximation ratios. We introduce the *rectilinear minimum bends traveling salesman problem*, in which the lines are restricted to being either horizontal or vertical. In this case, we give a 2-approximation algorithm. The algorithms for both this and the more general case are similar. They involve forming a relaxation which we call the *line cover* problem, where the line cover is the minimum-sized set of lines covering all the input points. We then solve this relaxation and use it to guide the choice of a tour. The differences in the two algorithms arise in the choice of potential lines to include in the cover and the ability to approximate the resulting line cover instance. For the general case, we obtain a set-cover problem, hence the logarithmic approximation ratio. For the rectilinear case, we obtain a bipartite vertex cover problem, which can be solved in polynomial time.

Finally, we consider a more restrictive case, the rectilinear minimum bends traveling salesman problem in which we have the further restriction that no two points are allowed to have the same x- or y-coordinate.

Although this condition may not apply to all inputs, it allows us to study carefully an aspect of the problem which the previous approximation algorithms ignore. Once they find a line cover, the other algorithms patch the lines together in a fairly straightforward way. For this problem, however, the minimum sized line cover for these instances is exactly equal to the number of input points and so the line cover gives us essentially no helpful information. Instead we focus on how to carefully patch together non-collinear points into a tour. For this case, we give an algorithm that finds a tour which makes at most two turns more than the optimal tour. Thus we have an approximation algorithm with an additive, rather than a multiplicative error bound.

Beyond the additive error bound, our algorithm for this problem introduces several interesting algorithmic techniques. We introduce two different ways to decompose a set of points in the Euclidean plane. We call these decompositions a *9-division* and a *4-division*. We then show that any set of points can either be decomposed into a 9-division or a 4-division. Guided by the 9-division or the 4-division, we then repartition the points into a set of points that are monotonically increasing, a set of points that are monotonically decreasing, and a set of points that fall on the perimeters of a set of nested boxes. Using this second decomposition, we are able to find a tour that uses at most two turns more than the optimal tour. We believe that these decompositions may be of independent interest.

We omit many of the proofs in this extended abstract.

2 Preliminaries

In this section we first introduce three versions of the Minimum Bends TSP. We also define the Line Cover problem, and its rectilinear variant, both of which will be useful subroutines in our algorithms.

Throughout this paper, we use the convention that a point p_i has x and y coordinates x_i and y_i respectively. When a point p_i falls on line l_j, we will say that line l_j *covers* point p_i.

We will be concerned with approximation algorithms, and will define a ρ-approximation algorithm for a minimization problem to be one which, in polynomial time, finds a solution of value ρOPT$+O(1)$, where OPT is the value of the optimal solution to the problem.

In this paper, we will consider traveling salesman tours in the plane. Our tours will differ from conventional tours in that the endpoints of the segments need not be at input points.

Definition 2.1 (Segmented Tour (S-Tour)) *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean plane, define an S-Tour over P to be a sequence of line segments $\pi = l_0, l_1, \ldots, l_{m-1}$ such that:*

1. *There exists a set of points $Q = \{q_0, q_1, \ldots, q_{m-1}\}$ such that the endpoints of l_i are q_i and $q_{i+1 \bmod m}$.*
2. *Each point $p_i \in P$ falls along some line $l_j \in \pi$.*

The number of bends in an S-Tour is equivalent to the number of line segments. Thus our objective, minimizing bends, is characterized by minimizing m, the number of line segments in the tour.

Definition 2.2 (Rectilinear Segmented Tour (□S-Tour)) *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean plane, define a □S-Tour over P to be an S-Tour $\pi = l_0, l_1, \ldots, l_{m-1}$ over P with the following additional property:*

3. *$l_j \in \pi$ is a horizontal line segment if j is even and a vertical line segment if j is odd (or vise-versa).*

Although we are defining a tour as a sequence of line segments, given a set of points $Q = \{q_0, q_1, \ldots, q_{m-1}\}$, and a starting direction $d \in \{\text{horizontal}, \text{vertical}\}$, there is a natural tour associated with Q and d, in which we traverse the points in order. To go from point q_i to point q_{i+1}, we greedily travel in the path that minimizes the number of lines needed to travel from q_i to q_{i+1} (given that we approached q_i in a particular direction).

We now define the main problem which we study, along with two restricted versions.

Definition 2.3 (Minimum Bends TSP (MBTSP))
Given: *A set of points* $P = \{p_1, p_2, \ldots, p_n\}$ *in the Euclidean Plane.*
Find: *An S-Tour* $\pi = l_0, l_1, \ldots, l_{m-1}$ *over* P *such that* m *is minimized. Let MBTSP* (P) *be the optimal value for* m*.*
Definition 2.4 (Rectilinear Minimum Bends TSP (□MBTSP))
Given: *A set of points* $P = \{p_1, p_2, \ldots, p_n\}$ *in the Euclidean Plane.*
Find: *A □S-Tour* $\pi = l_0, l_1, \ldots, l_{m-1}$ *over* P *such that* m *is minimized. Let □MBTSP* (P) *be the optimal value for* m*.*
Definition 2.5 (Non-Collinear Rectilinear Minimum Bends TSP (NC-□MBTSP))
Given: *A set of points* $P = \{p_1, p_2, \ldots, p_n\}$ *in the Euclidean Plane such that for any* $p_i, p_j \in P$ *with* $i \neq j$ *we have* $x_i \neq x_j$ *and* $y_i \neq y_j$ *(in the future we will call this the non-collinearity property).*
Find: *A □S-Tour* $\pi = l_0, l_1, \ldots, l_{m-1}$ *over* P *such that* m *is minimized. Let NC-□MBTSP* (P) *be the optimal value for* m*.*

Our algorithms will also use, as subroutines, several related problems. In the *Line Cover Problem (LC)*, we are given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean Plane. We wish to find a set of lines $L = \{l_1, l_2, \ldots, l_m\}$ such that each $p_i \in P$ falls along some $l_j \in L$ and such that m is minimized. We let LC(P) be the optimal value for m. The *Rectilinear Line Cover Problem (□LC)* is the variant in which the lines are restricted to being either horizontal or vertical.

3 A 2-approximation algorithm for □MBTSP

In this section we give a 2-approximation algorithm for the Rectilinear version of the Minimum Bends Traveling Salesman Problem (□MBTSP). As described in Definition 2.4, we are given an arbitrary set of points in the Euclidean plane, and we are looking for a rectilinear tour which minimizes the number of turns in the tour. We will show how to find a tour which has at most twice the optimal number of turns, plus a constant number of extra turns. Our approach uses a series of reductions through related problems. In particular, we will show that we can form an instance of the rectilinear line cover problem. The solution to this line cover instance will guide our 2-approximation. Further, as we will show in Section 3.1, the rectilinear line cover problem can be solved in polynomial time.

3.1 The equivalence of Rectilinear Line Cover and Bipartite Vertex Cover

The Rectilinear Line Cover problem will be used in our approximation algorithm for □MBTSP . Hence, in this section, we show how to solve the Rectilinear Line Cover problem, by formulating it as a Bipartite Vertex Cover problem, which can be solved efficiently. Given a graph $G = (V, E)$, a *vertex cover* is a set of vertices $V' \subseteq V$ such that for any edge $e \in E$ with endpoints $v_1, v_2 \in V$ we have either $v_1 \in V'$ or $v_2 \in V'$. In the Vertex Cover problem, we wish to find vertex cover with the minimum number of vertices. In the Bipartite Vertex Cover problem, we have the further constraint that the graph is bipartite. Let BVC(G) be the size of the minimum vertex cover over the bipartite graph G.

Notice the similarity between bipartite vertex cover and rectilinear line cover. A point in the input to rectilinear line cover must be covered by either the horizontal line or the vertical line at that location, while an edge in the input graph to Bipartite Vertex Cover must be covered by either its left endpoint or its right endpoint. This leads to a straightforward translation from an instance of Rectilinear Line Cover to an instance of Bipartite Vertex Cover.

Given any set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean Plane, we define $T(P)$ to be the bipartite graph $G = (V_l, V_r, E)$, where for each distinct x-coordinate $x_i \in P$, there exists a vertex $v_{x_i} \in V_l$, for each distinct y-coordinate $y_j \in P$, there exists a unique vertex $v_{y_j} \in V_r$, and for each point $p_k = (x_k, y_k) \in P$ there exists an edge $e \in E$ with endpoints $v_{x_k} \in V_l$ and $v_{y_k} \in V_r$.

Lemma 3.1 *Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the Euclidean plane. If P has a rectilinear line cover consisting of m lines then $T(P) = (V_l, V_r, E)$ has a vertex cover consisting of at most m vertices.*

Lemma 3.2 *Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the Euclidean plane. If $T(P) = (V_l, V_r, E)$ has a vertex cover of size m then P has a rectilinear line cover of size m.*

Theorem 3.3 *Let $P = \{p_1, p_2, \ldots, p_n\}$ be an arbitrary set of points in the Euclidean plane and let $G = T(P)$. It follows that LC(P) = BVC(G), and hence Rectilinear Line Cover can be solved in $O(n^{1.5})$ time.*

Proof. Together, Lemmas 3.1 and 3.2 imply that Rectilinear Line Cover can be solved in polynomial time. The algorithm consists of two parts. First, we must create the input to Bipartite Vertex Cover, which involves sorting the set of x and y coordinates in $O(n \log n)$ time and forming the appropriate graph. The graph has $O(n)$ edges (corresponding to input points) and $O(n)$ vertices (at most 2 per input point), and can be constructed in $O(n)$ time. The size of the minimum bipartite vertex cover equals the size of the maximum matching [3], and hence can be found in $O(n^{1.5})$ time, using the Hopcroft-Karp algorithm [14]. □

3.2 The relationship between □MBTSP and Rectilinear Line Cover

In this section, we give a 2-approximation algorithm for □MBTSP . The algorithm has 2 parts. First we give a translation from □MBTSP to rectilinear line cover such that the optimal number of lines in the rectilinear line cover instance is no more than the number of turns made in the optimal solution to the □MBTSP instance. Second, once the optimal set of lines is known, we show how to find a rectilinear tour having no more than twice as many turns as there are lines in the optimal rectilinear line cover. The resulting tour is a 2-approximation since the number of lines in the rectilinear line cover instance defines an obvious lower bound on the □MBTSP instance.

Lemma 3.4 *If □MBTSP $(P) = m$, then LC$(P) \leq m$.*

Lemma 3.5 *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean plane, if LC(P)= m, then □MBTSP (P) $\leq 2m + O(1)$.*

Proof. Let $L = l_1, l_2, \ldots, l_m$ be a rectilinear line cover over the points in P. Let v be the number of vertical lines in L and let h be the number of horizontal lines in L.

Constrain the tour as follows. Define a bounding box B such that all points in P are contained within the boundaries of B. Connect all horizontal lines along the boundaries of B, in such a way that a tour may travel in a S-like fashion along the union of all horizontal line and the boundaries of B.

This adds $2h - 1$ lines to the tour. Repeating this process in the vertical direction adds $2v - 1$ lines. Joining the horizontal and vertical sections at both ends adds 4 more lines. Thus the total number of lines is $2h - 1 + 2v - 1 + 4 = 2h + 2v + 2 = 2m + O(1)$. Figure 3, in the appendix, shows a sample tour. □

Theorem 3.6 *A 2-approximation algorithm for the Minimum Bends Rectilinear Traveling Salesman Problem can found in $O(n^{1.5})$ time.*

Proof. This follows from the Lemmas 3.4 and 3.5, Theorem 3.3, and the fact that given a Rectilinear Line Cover, the tour can be constructed in $O(n)$ time. □

We observe that the previous theorem is tight in the following sense. Any algorithm which uses Rectilinear Line Cover as a lower bound can be a factor of 2 off from optimal. That is, there exist sets of points P for which □MBTSP $(P) \geq 2$ □LC(P). Consider the following class of examples. Choose $k >> n$, and create a set of nk points with Cartesian coordinates (i, j), one for each integer pair with $1 \leq i \leq n, 1 \leq j \leq k$. The optimal line cover for this set of points consists of n horizontal lines, yet there is no way to find a tour through these points using fewer than $2n$ lines. (Our algorithm will find such a tour).

4 An approximation algorithm for MBTSP

We now turn to the general minimum bends TSP problem. Consider what happens if we try to apply the algorithm of the previous section to this problem. Although the details are more involved, we can formulate a line cover problem, by choosing the candidate lines to be those which cover maximal colinear subsets of the input points, together with the degenerate "lines" formed by single points. We can then solve the line cover problem, and use this to obtain a tour, paying roughly a factor of 2 in the process. The only problem in this approach is that the resulting line cover problem is no longer equivalent to a bipartite vertex cover problem. Instead, it is now a set cover problem, and so our approximation bound will not be as good.

The details of our algorithm appear in Figure 1. In the first seven lines, we compute the set T, which contains all lines which might be in the optimal tour. We include in this set all lines going through two or more points. The optimal tour may also consist of lines that go through only one input point, hence we include singleton points as degenerate lines. The set T can clearly be computed in polynomial time, and in this extended abstract we omit the discussion of the data structures needed to compute it efficiently.

We now form a set-cover instance as follows. The points are the initial input points, while the sets are the sets in T. By arguments similar to those in Lemma 3.4, it is clear that the optimal set cover is a lower bound on the optimal tour. Further, by arguments similar to those in Lemma 3.5, if we take the line segments in T', a line cover of the points P, order them, and connect the two endpoints of each two consecutive segments with an additional line segment, we do indeed get a tour. The code in lines 9 through 13 achieves this. Thus we have shown:

Theorem 4.1 *Algorithm Find-MBTSP, given an input to the Minimum Bends TSP problem, computes a tour which is a $2\rho + 2$-approximation, where ρ is the approximation bound for Set Cover.*

In general, the best set cover approximation is $\ln n$ [8, 13]. However, in the case when each set is of size no more than z, the approximation ratio is roughly $\ln z$, and tighter bounds are known for small values of z[9]. The maximum set size corresponds to the maximum number of collinear points and thus we have the following:

Corollary 4.2 *Given a set of points P among which no more than z are collinear, Algorithm Find-MBTSP is an $O(\ln z)$-approximation algorithm.*

5 An approximation of NC-□MBTSP using OPT+2 bends

In this section we consider □MBTSP with the additional constraint that no two points share an x-coordinate or a y-coordinate. In this case, no two points may lie along the same line of the tour, and hence n is a lower bound on the number of bends in the tour. Also the number of lines in any rectilinear tour must be even, since the tour must have the same number of horizontal and vertical lines. Thus if n is odd, then $n + 1$ is a lower bound on the number of bends.

Our approximation algorithm finds a tour with $n + 2$ lines if n is even and $n + 3$ lines if n is odd. Thus, the algorithm finds a tour with at most OPT+2 bends.

5.1 Box Points and Diagonal Points

Our algorithm depends heavily on the division of the points into two categories: diagonal points, and box points. Here we define these two sets. In the following section we show that the input to NC-□MBTSP can always be partitioned into one set of box points and one set of diagonal points.

We say that a set of points $P = \{p_1, p_2, \ldots, p_n\}$ is *monotonically increasing* if, for any two points $p_i = (x_i, y_i)$, and $p_j = (x_j, y_j) \in P$ we have $x_i > x_j$ if and only if $y_i > y_j$. Similarly, P is *monotonically decreasing* if for any two points $p_i, p_j \in P$ we have $x_i > x_j$ if and only if $y_i < y_j$.

Function Find-MBTSP(P)

```
T = ∅
for all (p_i ∈ P)
        for all (p_j ∈ P)
                let S_k = the set of all points in P along the line through (p_i, p_j)
                if S_k is not in T then add S_k to T.
for all (p_i ∈ P)
        let T = T ∪ {{p_i}}
let T′=SET-COVER(T, P) // SET-COVER returns a ln n-approximation for Set Cover
for all (S_k ∈ T′)
        let (q_{2k−1}, q_{2k}) = the two extremum points in S_k ⊆ P, in either the x or the y direction.
        ( if S_k is a singleton p, then q_{2k−1} = q_{2k} = p)
let Q = ⋃ q_i
for all (q_i ∈ Q)
        let l_i = the line segment (q_i, q_{(i mod |Q|)+1})
return π = l_1, l_2, ..., l_{|Q|}
```

Figure 1: An approximation algorithm for MBTSP

Definition 5.1 (Diagonal Points) *A set of points may be considered diagonal points if they can be partitioned into two sets $\mathcal{I}$ and $\mathcal{D}$ such that the points in $\mathcal{I}$ are are monotonically increasing and the points in $\mathcal{D}$ are monotonically decreasing.*

We will need a few preliminary definitions before defining box points.

Definition 5.2 (Smallest enclosing rectangle) *Given any set of points, P, we define the smallest enclosing rectangle to be the rectangular region containing all points of P, bounded by horizontal and vertical lines, having the smallest possible area.*

Definition 5.3 (4-box) *We define a 4-box to be any set of four points $P = p_1, \ldots, p_4$, such that a single point falls along each of the four boundaries of the smallest enclosing rectangle around P, and no point falls at any corner of that rectangle.*

Definition 5.4 (Box Points) *A set of points may be considered box points if they can be partitioned into subsets of cardinality 4, such that each subset forms a 4-box. Additionally, for any two of these rectangles, one will lie entirely within the other.*

Ultimately we want to show that for any given set of points with the non-collinearity property, all points can be partitioned into a single set of diagonal points and single set of box points as defined above.

5.2 Planar Subdivisions

We will classify the input points using a method we call the *Planar Subdivision Method*. We will define two different ways to divide the Euclidean plane, a 4-division and a 9-division. Then we will show that among any set of points in the Euclidean plane with the non-collinearity property there always exists a way to divide the plane into a 4-division or a 9-division. See Figure 4 in the Appendix for examples of the two divisions.

Definition 5.5 (4-division) *A 4-division is a division of the Euclidean plane via a single horizontal and a single vertical line, into 4 quadrants, NE, NW, SE, and SW, such that*

1. *Those points in the NW region and those points in the SE region are monotonically decreasing.*

2. *Those points in the NE region and those points in the SW region are monotonically increasing.*

Due to the relative positions of the quadrants, it is also the case that the union of the points in the NW and SE regions are monotonically decreasing. This will allow us to cover these points at a rate of one per line. Likewise the union of the points in the NE and SW regions are monotonically increasing. Note that in order to have a 4-division it is not necessary that a particular quadrant contain any points. In fact, among any arrangement of 0, 1, 2, or 3 points, there exists a 4-division. It is not not necessarily true that there exists a 4-division among 4 points. (A simple case analysis will show this.)

Definition 5.6 (9-division) *A 9-division is a division of the Euclidean plane into 9 regions, NE, NW, SE, SW, N, S, E, W, and C (the Center), defined by two horizontal and two vertical lines, satisfying the two properties of the 4-division along with:*

3. *The N, S, E, and W regions are all empty.*

4. *There exists exactly one point along each of the four boundaries of the center region, and no points at any of the corners. The interior of the center region may contain any arrangement of points.*

We now define a function *PLANAR-SUBDIVISION* which, when given a set of points with a 9-division, it returns NW, NE, SW, SE, C, and B, which are the points in the NW, NE, SW, SE, and C regions, and the 4 points along the boundary of C respectively. If no 9-division exists, the algorithm finds a 4-division, and returns NW, NE, SW, and SE, which are the sets of points in the respective quadrants.

The algorithm repeatedly finds the smallest enclosing rectangle around the set of points. If that box has 4 points, then a 9-division exists. If that box has 1-point, then a 4-division exists. Otherwise, the box must have a corner point. We remove that corner point, put it in the appropriate set, and continue the algorithm on the remaining points. Of the sets returned, B must contain exactly four points, but any of the other sets returned may be empty. Detailed pseudocode appears in Figure 5 in the Appendix.

In order to analyze the planar subdivision algorithm, we will need several additional properties. Note that if any edge of a smallest enclosing rectangle did not contain any points, then the region would not properly be a smallest rectangle, as we could shrink it on that side. Thus the smallest enclosing rectangle about a set of points with the non-collinearity property must have exactly one point along each of its edges. In the event that fewer than four points fall along the edges of a smallest enclosing rectangle, then at least one point must lie at a corner of that rectangle. We state this as the corner lemma:

Lemma 5.7 *(Corner Lemma) Given a set of points $P = p_1, p_2, ..., p_n$ such that fewer than 4 points lie along the boundary of the smallest enclosing rectangle, R, at least one point in P must lie at a corner of R.*

We now know that given a set of four points P with the non-collinearity property, the following are equivalent: 1) The set P forms a 4-box. 2) The smallest enclosing rectangle about P has four points along its boundary. 3) No point lies at the corner of the boundary of the smallest enclosing rectangle about P.

The 4-box plays an important role in the classification of box points. Here we show the uniqueness of an enclosing 4-box.

Lemma 5.8 *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ with the non-collinearity property, if there exists a 4-box in P, then there exists a unique 4-box which encloses all other 4-boxes in P.*

Lemma 5.9 *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ with the non-collinearity property, if there exists a 9-division in P, then* PLANAR-SUBDIVISION(P) *finds the 9-division, and the points returned in B form the unique enclosing 4-box in P.*

Proof. First, let us show that the algorithm finds the unique enclosing 4-box if it exists. The set P initially contains all points. Let us think of this as the set of points that could possibly be in a 4-box. A point is only removed from P when it is on the corner of the smallest enclosing rectangle around P. Such a point cannot be in a 4-box, since it would have to be at the corner of any 4-box in P. Thus, we do not remove any points from P that could be in a 4-box. The algorithm only terminates when all points are removed from P, or when it finds that the smallest enclosing rectangle contains four points along its boundary. Thus it follows that either there does not exist a 4-box in P, or the algorithm finds the unique enclosing 4-box.

Next we show that if there exists a 4-box in P, then the algorithm finds a 9-division. Assume there exists a 4-box in P. Then we know that the algorithm finds the unique enclosing 4-box in P. Assume, by way of contradiction, that this 4-box does not define a 9-division. Then one of the properties of a 9-division would have to be violated. We will attempt to violate each property, and then contradict each violation.

Properties 1 and 2: Let there be two points in a corner region (NW, NE, SW, SE) whose points do not follow the region's monotonicity property. Then these two points, together with the two far points from the 4-box would form a larger 4-box, and the given 4-box would not be the unique enclosing 4-box.

Property 3: Let there be a point in side region (N,W,E,W). Then this point together with the 3 far points of the given 4-box would form a larger 4-box.

Property 4 holds trivially and thus, our algorithm finds a 9-division if a 4-box exists. Since every 9-division contains a 4-box, B, our algorithm finds a 9-division if it exists. □

Lemma 5.10 *Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ with the non-collinearity property, if there does not exists a 9-division in P, then there exists a 4-division in P, and* PLANAR-SUBDIVISION *finds a 4-division.*

Proof Sketch. First let us show that if there does not exist a 9-division in P, then there exists a 4-division. We do this by induction on the cardinality of a set of points with no 9-division. Assume there does not exist a 9-division in P and consider the smallest bounding rectangle about P. There cannot be 4 points on this rectangle, or there would be a 4-box and thus a 9-division in P. Therefore, there must be a point, say p_c, at the corner of this rectangle. Now consider the set of points $P - \{p_c\}$. Assume inductively that all sets of $|P| - 1$ points which do not have a 9-division contain a 4-division. Thus $P - \{p_c\}$ contains a 4-division (since it contains no 9-division).

Adding p_c to this set can only eliminate the 4-division if it breaks the monotonicity property in some region. Since p_c was a corner point in the set P, it follows that it is extremal in both the x and y direction. We can show that for at least one region R, p_c will satisfy the monotonicity property for that region. Consider that p_c is not located in R. Then we can show that it is possible to reassign the boundaries so that p_c is in R without changing the designation of any point in $P - \{p_c\}$.

Thus, we have our inductive hypothesis. Given any set of points of cardinality n which displays the non-collinearity property and has no 9-division, if there exists a 4-division on any set of points of cardinality $n-1$ which has no 9-division, then there exists a 4-division on any set of size n. For our base case we simply state that there is a 4-division on any set of points of cardinality 1, and we are done with the inductive proof.

It remains for us to show that our algorithm finds a 4-division, if there is no 9-division. Assume there is no 9-division on P. Then our algorithm can never find a 4-box in P. Thus at every iteration, it must find a corner point. As this point is extremal in two directions, it maintains a particular monotonicity with all remaining points in P. Thus, assigning it to the region which has that monotonicity property cannot violate the property in that region. Our algorithms does this. Furthermore, the regions remain properly defined, since, when a point is assigned to a region (and removed from P) it is extremal with respect to all remaining points in P in the proper direction. Thus for any two points in different regions, we guaranteed the proper directional relationship when the first of these was removed from P. Therefore, the 4 regions the algorithm returns will be properly defined and have the proper monotonicity property, and thus they will form a 4-division. □

The previous two lemmas imply the following theorem:

Theorem 5.11 *(Planar Subdivision Theorem) Given any set of points $P = p_1, p_2, ..., p_n$ such that no two points share an x-coordinate or a y-coordinate, there must exist either a 4-division or a 9-division among those points, and PLANAR-SUBDIVISION(P) returns a proper 4-division or 9-division among P.*

5.3 The Algorithm

Figure 2 describes the approximation algorithm which finds tour of length at most OPT+2 among a given set of points. In lines 1-13, it repeatedly applies the planar subdivision theorem to obtain a decomposition of the points. The loop will run for $O(n)$ iterations because each iteration, save the last, reduces the cardinality of C by at least 4. Recall that each decomposition partitions the points into sets NW, NE, SW, SE, B and C. The points in NE, SW, and SE are placed in the appropriate diagonals, either $\mathcal{I}$ or $\mathcal{D}$. In the case of a 9-division, the points in NW are indexed by the iteration in which they were found. The box points are indexed similarly.

The tour then includes the points in $\mathcal{I}$, the points in $\mathcal{D}$ from SE and then alternates between the points in $\mathcal{D}$ from NW and the various boxes. In order to get the exact bounds claimed, we have to be careful about the starting direction and we may also have to move points from one diagonal to another. These details are discussed in the proof of Theorem 5.13.

Our algorithm may require that we move diagonal points from one diagonal to another. We call the algorithm that performs this SWITCH-DIAGONAL.

Theorem 5.12 *(Diagonal Switching Theorem) For any set of diagonal points, derived using the Planar Subdivision method, there exists at least one point which may be either swapped between $\mathcal{D}$ and $\mathcal{I}$, without violating the monotonicity properties.*

Proof. Recall that we defined the diagonal points as those points in the NE, NW, SE, and SW quadrants. Also recall that those points were relegated to those region by virtue of being at the corners of a series of rectangles, or being the final point of a 4-division. Note further that for any two of these rectangles, one is completely enclosed within the other.

Consider the innermost rectangle of this series which has a diagonal point at its boundary. If this point is the only diagonal point on this box, then it may be placed in either $\mathcal{I}$ or $\mathcal{D}$ without violating the monotonicity properties. If there are 2 diagonal points on the boundary of this rectangle, then they can both be in the one set from $\mathcal{I}$ and $\mathcal{D}$ whose monotonicity property they share, or they can be split between the two sets $\mathcal{I}$ and $\mathcal{D}$. □

Theorem 5.13 *Algorithm Find-NC-□MBTSP , given an input to the Non-Collinear Minimum Bends TSP, finds a tour which contains at most 2 additional bends more than the optimal.*

Proof Sketch. As stated earlier, if n is even, then n is a lower bound on the number of lines in the optimal tour, by the non-collinearity property. Similarly, if n is odd, then $n + 1$ a lower bound.

Now consider the number of lines in a tour returned by Find-NC-□MBTSP . The values returned consist of a series of point sequences, together with a starting direction. The total number of lines will be the sum of the number of lines in each sequence, plus the sum of the additional lines used in connecting adjoining sequences. An additional line will be necessary between two sequences if the preceding sequence finishes its path heading away from the start of the subsequent sequence. More than one additional line would be

Function Find-NC-$\square$MBTSP (P)

```
let i = 1
while (P ≠ ∅)
    if ( (4-division,NW,NE,SW,SE)=PLANAR-SUBDIVISION(P) )
        I = I ∪ NE ∪ SW ; D' = D' ∪ NW ∪ SE ; P = ∅
    else if ( (9-division,NW,NE,SW,SE,B,C)=PLANAR-SUBDIVISION(P) )
        I = I ∪ NE ∪ SW ; D' = D' ∪ SE ; D_i = NW ; Box_i = B ; P = C ; i++
I_sort = SORT(I) // Along the y-coordinate
D'_sort = SORT(D') // Along the y-coordinate
if (|I_sort|) is odd
    Starting-Direction = East
else if (|I_sort|) is even
    Starting-Direction = North
π = (Starting-Direction,I_sort, D'_sort,Box_{i-1},D_{i-1},...,Box_1,D_1)
if (FINISHING-DIRECTION(π) == Starting-Direction)
    return SWITCH-DIAGONAL(π,I,D') // Apply Diagonal Switching Theorem
else return π
```

Figure 2: An Approximation algorithm for NC-$\square$MBTSP

necessary only if a particular entrance direction were required at a point. Our definition of a tour defined by points will not require this, except to rejoin the end of a tour to its starting point.

The sequences $\mathcal{I}_{sort}$ and $\mathcal{D}'_{sort}$, because of their monotonicity, can be covered, starting from an extremal point, using a staircase-like path. This can be done at a rate of one line per point unless the starting direction requires that an extra turn be made in traveling from the first point to the second point.

Transferring from $\mathcal{I}_{sort}$ to $\mathcal{D}'_{sort}$ will incur an extra turn, unless the tour can proceed directly to $\mathcal{D}'_{sort}$, in which case an extra line is needed after the first point of $\mathcal{D}'_{sort}$. Either way the total number of line segments needed to cover $\mathcal{I}_{sort}$ and $\mathcal{D}'_{sort}$ will be $|\mathcal{I}|+|\mathcal{D}'|+1$. Note that this algorithm does not handle the degenerate cases where the sets may have 0 or 1 points, but optimal paths can be found on a case by case basis.

Upon completion of $\mathcal{D}'_{sort}$ the path is inside the innermost uncovered Box$_i$, heading in either the North direction or the West direction, and all remaining points in D_i are to the North and West of the most recently covered point. These conditions are invariants to maintain after each subsequent point sequence is completed. The conditions are sufficient to cover all points of the inner most uncovered Box$_i$ using exactly 4 additional lines. The invariants remain true upon completion of Box$_i$. The invariants are also sufficient to cover a particular D_i using $|D_i|$ lines, and remain true after such a covering. Thus all points in $\bigcup_i$ (Box$_i \cup D_i$) can be covered with $\sum_i(|\text{Box}_i| + |D_i|)$ lines.

Joining the end of the tour back to its beginning will require 1 or 2 or 3 additional lines, depending on the starting and finishing directions. If 3 additional lines are required, then the starting and finishing directions must both be North. If this is true, then we can apply the Diagonal Switching Theorem to modify the sizes of $\mathcal{D}'$ and $\mathcal{I}$ each by 1. This changes both the starting and finishing directions, allowing the tour to complete with 1 additional line.

The resultant tour will have $n + 2$ total lines, or $n + 3$ total lines. By the parity arguement, the tour can only have $n + 3$ lines if n is odd. Thus we achive the desired optimality bound. $\square$

The total number of iterations of both the *while* loops in Find-NC-$\square$MBTSP and in PLANAR-SUBDIVISION is at most n, since an iteration of either loop reduces the size of P. The SMALLEST-ENCLOSING-RECTANGLE is thus run at most n times, and if run in a brute force fashion takes $O(n)$ time to run. However, by using 2 binary search trees, one keyed on the x-coordinate and one keyed on the y-coordinate, and an amortized analysis of the work needed per iteration, we can show that this portion of the algorithm runs in $O(n \log n)$ time. The remainder of the algorithm runs in $O(n)$ time.

Acknowledgements

We thank Neal Young for many helpful discussions on this work. Early work on this problem, particularly that in Section 3, was done jointly with Neal. We also thank Robert Fitch and Daniela Rus for their discussions on the Rectilinear Minimum Bend Path with Obstacles problem.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *Informatique Theoretique et Applications*, pages 79–87, 1986.

[2] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29(3):697–711, June 2000.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[4] Esther M. Arkin, Yi-Jen Chiang, Joseph S. B. Mitchell, Steven S. Skiena, and Tae-Cheon Yang. On the maximum scatter TSP (extended abstract). In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211–220, New Orleans, Louisiana, 5–7 January 1997.

[5] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *JACM: Journal of the ACM*, 45, 1998.

[6] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–172, May 1994.

[7] N. Christofedes. Worst case analysis of a new heuristc for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.

[8] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979.

[9] R. Duh and M. Furer. Approximation of k-set cover by semi-local optimization. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 256–264, 1997.

[10] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for finding a maximum weight hamiltonian circuit. *Operations Research*, 27(4):799–809, July–August 1979.

[11] R.S. Garfinkel. Minimizing wallpaper waste, part i: a class of traveling salesman problems. *Operations Research*, 25:741–751, 1977.

[12] P.C. Gilmore and R.E. Gomory. Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. *Operations Research*, 12:655–679, 1964.

[13] Dorit Hochbaum, editor. *Approximation Algorithms*. PWS, 1997.

[14] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[15] L.J. Hubert and F.B. Baker. Applications of combinatorial programming to data analysis: the traveling salesman and related problems. *Pyschometrika*, 43:81–91, 1978.

[16] R. Kosaraju, J. Park, and C. Stein. Long tours and short superstrings. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 166–177, 1994.

[17] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley and Sons, 1985.

[18] D.T. Lee, C.D. Yang, and C.K. Wong. Problem transformation for finding rectilinear paths among obstacles in two-layer interconnection model. Technical Report 92-AC-104, Dept. of EECS, Northwestern University, 1992.

[19] D.T. Lee, C.D. Yang, and C.K. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70, 1996.

[20] J.K. Lenstra and A.H.G. Rinnooy Kan. Some simple applications of the travelling salesman problem. *Operations Research Quarterly*, 26:717–733, 1975.

[21] W.T. McCormick, P.J. Schweitzer, and T.W. White. Problem decomposition and datareorganization by a clustering technique. *Operations Research*, 20:993–1009, 1972.

[22] J.S.B. Mitchell, C. Piatko, and E.M. Arkin. Computing a shortest k-link path in a polygon. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 573–582, 1992.

[23] C.D. Yang, D.T. Lee, and C.K. Wong. On bends and lengths of rectilinear paths: a graph-theoretic approach. *Internat. J. Comp. Geom. Appl.*, 2:61–74, 1992.

[24] C.D. Yang, D.T. Lee, and C.K. Wong. On minimum-bend shortest recilinear path among weighted rectangles. Technical Report 92-AC-122, Dept. of EECS, Northwestern University, 1992.

[25] C.D. Yang, D.T. Lee, and C.K. Wong. Rectilinear path problems among rectilinear obstacles revisited. *SIAM Journal on Computing*, 24:457–472, 1992.

[26] C.D. Yang, D.T. Lee, and C.K. Wong. On bends and distance paths among obstacles in two-layer interconnection model. *IEEE Transactions on Computers*, 43:711–724, 1994.

Appendix

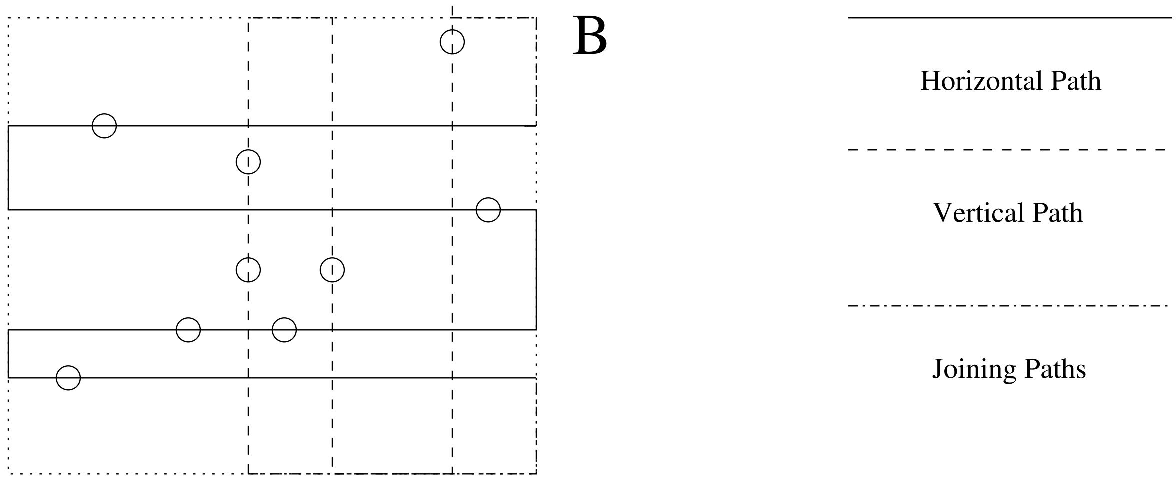

Figure 3: A sample tour formed by □MBTSP

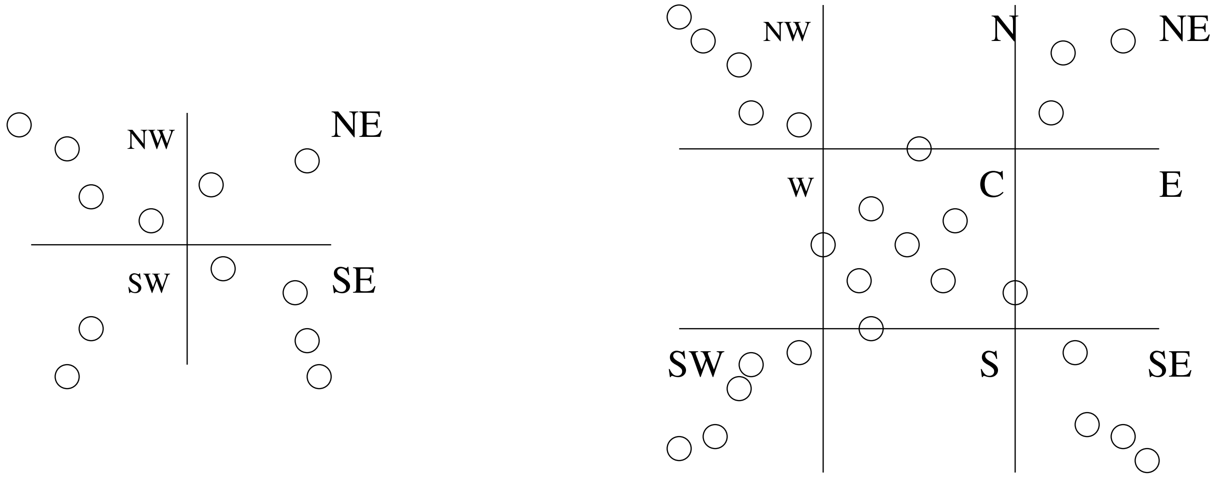

Figure 4: A 4-division and a 9-division

```
Function PLANAR-SUBDIVISION(P)
    let NW=NE=SW=SE=C=B=∅ // The values to return
    while (P ≠ ∅)
        let R=SMALLEST-ENCLOSING-RECTANGLE(P)
        if (|R| == 4)
            Division-Type=9
            C = P − R
            B = R
            return (Division-Type,NW,NE,SW,SE,C,B)
        else if (|R| == 1)
            Division-Type=4
            NW = NW ∪R
            return (Division-Type,NW,NE,SW,SE)
        else
            let maxx = p_i ∈ B such that x_i is maximized
            let minx = p_i ∈ B such that x_i is minimized
            let maxy = p_i ∈ B such that y_i is maximized
            let miny = p_i ∈ B such that y_i is minimized
            if (maxx == maxy)
                NE = NE ∪ maxx ; P = P - maxx
            else if (minx == maxy)
                NW = NW ∪ minx ; P = P - minx
            else if (maxx == miny)
                SE = SE ∪ maxx ; P = P - maxx
            else if (minx == miny)
                SW = SW ∪ minx ; P = P - minx
```

Figure 5: The Planar Subdivision Algorithm. It returns either a 4-division or a 9-division