CURVE SIMPLIFICATION UNDER THE L_2-NORM

BENJAMIN BERG

Abstract

With the increased production and availability of large data sets comes the opportunity to glean valuable information about behaviors that would previously have been difficult to study formally. One such data set describes the trajectory of various objects, such as that generated by a GPS. Naturally, we often wish to compare these trajectories. To simplify this process, we seek an efficient yet accurate summary of a trajectory. In this instance, we consider the problem of finding such a summary, a k-piece, continuous, piecewise linear function which minimizes distance from the underlying trajectory in the L_2-norm. We present an exact algorithm which is exponential in k, as well as a $(1+\epsilon)$ approximation algorithm which runs in polynomial time.

1. Introduction

With the increased production and availability of large data sets comes the opportunity to glean valuable information about behaviors that would previously have been difficult to study formally. One such data set is generated by the GPS tracking of various entities. Presented with many trajectories across a common geographic area, much of the useful information encoded in this data can be drawn from comparing the trajectories to one another. The input sizes of such trajectories, however, can be large, and such data sets are also plagued with uncertainty. Furthermore, to find methods which are both accurate and efficient in the sizes of the input trajectories is difficult. To overcome this, we can consider a piecewise linear function which serves as a succinct and accurate summary of an underlying trajectory, which will facilitate comparison of several trajectories [11]

To overcome this, one can consider a preprocessing step, where trajectories are simplified to a piecewise linear function of at most k pieces, and then processed with the guarantee of being of size $O(k)$. Of course, such a simplification will result in some deviation from the actual trajectory. The goal, then, is to perform some preprocessing which is both efficient in the size of an input trajectory, and which performs the simplification while generating as little deviation from the true data as possible.

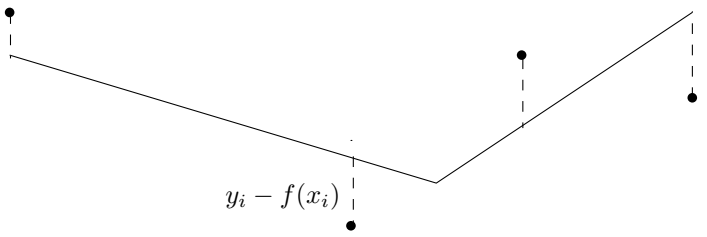

FIGURE 1. we take the sum of the squared distances between each point and the function, f, to calculate $\Delta_2(f, P)$

1.1. **The Problem.** Given a function $f : \mathbb{R} \to \mathbb{R}$, and a sequence of points, $P \subseteq \mathbb{R}^2$, let the **error** of a point set P be defined as

$$\Delta_2(f, P) = \sum_{(x,y)\in P} (f(x) - y)^2.$$

In general, let $\Delta_t(f, P)$ be error calculated in the same way, but measuring error in the L_t-norm.We are interested in continuous piecewise linear functions with at most k pieces for a given integer $k \geq 1$. Formally, we say that

$$f(x) = \begin{cases} f_1(x) = m_1x + c_1 & x < b_1 \\ f_2(x) = m_2x + c_2 & b_1 \leq x < b_2 \\ \vdots & \\ \vdots & \\ f_k(x) = m_kx + c_k & b_{k-1} \leq x \end{cases}$$

Since f is continuous, for all $1 \leq i < k$, $f_i(b_i) = f_{i+1}(b_i)$. We refer to the set $B = \{b_1, b_2, \ldots, b_{k-1}\}$ as the **breakpoints** of f. The graph of f, then, is an x-monotone polygonal chain. The goal is to find the function f^* which minimizes the error $\Delta_2(f, P)$ where the minimum is taken over all piecewise-linear functions of k pieces.

Finally, we will define for each piecewise linear function f the **coverage** of f as follows. Let $P_i(f) = \{p \in P | x(p) \in [b_i, b_{i+1}]\}$, and the set $\mathcal{P}(f) = \{P_1(f), \ldots, P_k(f)\}$. The coverage of f. Two functions f and g are then said to have the same coverage of a point set P if and only if $P(f) = P(g)$

1.2. **Related Work.** A significant amount of work has been done on the so-called curve-simplification problem which, given a point set, tries to find a set of points or lines such that the distance between each point in the point set and the closest point or line given by the algorithm is minimized. Much of this work has been concerned with curve simplification under the requirement that the vertices of the simplified curve are a subset of the vertices of the original curve. Imai and Iri [7] present a framework for solving this problem under the L_t-norm, considering each point in a point set P to be a vertex in a graph G_p. An edge

connects any two nodes p_i and p_j for which the error associated with the line connecting the vertices, $\Delta_m(p_i, p_j, P)$ is less than some error threshold ϵ. Finding the shortest path in this graph from the first to last point in P provides an optimal ϵ-simplification, and runs in $O(n^2 \log n)$ time. We will use a similar technique in computing the optimal discontinuous k-piece piecewise linear function over a point set P.

Jagadish et al. present an $O(n^2k)$ dynamic programming solution for the case where our approximation is not required to be continuous [9]. This essentially the same as the solution presented in section 2, however in this thesis we choose to approach the problem as a shortest path problem. Nonetheless, we acknowledge that a similar result exists, and present our formulation to give insight into principles used later in the thesis.

Douglas and Peucker originally proposed an $O(n^2)$ algorithm for curve simplification which produces a approximation of a curve which minimizes the Hausdorff error, a measure of error in the L_∞-norm. However, this method does not limit the size of the approximation, a goal we wish to accomplish in this formulation of the problem [3].

Agarwal et al. [1] improves upon this algorithm, describing a near linear time greedy algorithm for the above problem where error is measured over the L_∞-norm that gives an optimal ϵ-simplification. We opt in this instance to consider the L_2-norm, as comparisons between trajectories in this norm have desirable statistical properties.

Indyk et al. [8]describe a sub linear algorithm for constructing a histogram which is has additive error no more than ϵ in the L_2 distance from a given point set. This uses a greedy algorithm which proceeds by iteratively adding the "best" piece to the histogram in each iteration, until a histogram which which covers the entire point set is developed. A sub linear running time is achieved by searching over only a subset of potential intervals whose endpoints are close to points in P.

Guha et al. [5] describe a linear time algorithm for finding a $(1 + \epsilon)$ approximation of the optimal histogram covering a set of points. This result uses the assumption that larger intervals will be more costly to cover (intervals are monotonic under inclusion). This assumption allows for a linear time dynamic programming solution, and generalizes to include piecewise linear functions.

There has been extensive work on a similar problem in the area of computer graphics. Often, a set of points is provided by a user, and the goal is to fit a spline to these points. However, rather than attempting to produce a compressed summary of the underlying function, the goal of these algorithms is simply to reconstruct this underlying function. For a complete description of this topic, see *Curves and Surfaces in Geometric Modeling: Theory & Algorithms*[4]

This problem can also be viewed as a special case of segmented regression in which continuity is required. Segmented regression divides an input set into disjoint intervals, and then applies a separate regression line to each interval. Oosterban et al. [10] show an example of how segmented regression can be used to explain trends where the general conditions allowing the application of linear regression are not met. Furthermore, this discusses some of the statistical conclusions that can be drawn from performing such a regression.

To our knowledge, it is not known whether this problem is in P, and no efficient approximation algorithm exists for the formulation which requires the simplification to be a continuous, piecewise linear function of at most k pieces.

1.3. **contributions.** The contributions of thesis are as follows.

(1) If f is not required to be continuous, we describe a known dynamic programming solution which runs in $O(n^2k)$. However, finding such a function is often a bad approximation of the true underlying trajectory, since these trajectories are often continuous.
(2) We will consider a technique for obtaining the continuous piecewise linear function, f, of least error over some sequence of points. To do this, we first give an algorithm to find an optimal solution for $k = 2$ (i.e. when f consists of two pieces). We will then extend this solution for to include cases where $k \geq 2$. This runs in $O(n^{O(k)}(k^5 + n))$.
(3) Finally, we will consider a $(1+\epsilon)$ approximation algorithm for finding the optimal k piece continuous piecewise linear function over a sequence of points. Our algorithm hinges on finding a discrete set of points such that any piece in our approximately optimal function must pass through some pair of these points. We will then search for the best k piece continuous piecewise linear function that meets these conditions. This runs in $O((\frac{1}{\epsilon}n \log n)^4 k)$.

2. The Discontinuous Case

Let us consider the case where f is not required to be continuous. We can observe that this formulation presents an optimal substructure, and we can use a dynamic programming approach to solve the problem efficiently. To solve this, we expand on the idea of building a graph and finding some shortest path in this graph, a technique presented in the Imai and Iri paper mentioned above.

2.1. **The Algorithm.** Given a point set, let $P = \{p_1, p_2, \ldots, p_n\}$ where $(x_i, y_i) = p_i \in P$ be the set of input points sorted in increasing order by their x-coordinate. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of candidate breakpoints.

(1) We will construct a graph, $G(V, E)$ with a vertex V_i corresponding to each candidate breakpoint x_i. For each $0 \leq i < j \leq n$, let the exist an edge (v_i, v_j).

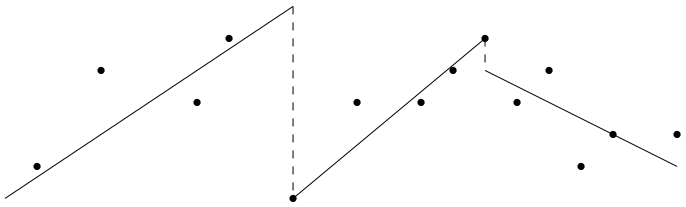

FIGURE 2. If the requirement that the function is continuous is dropped, we can easily find the optimal function for a given coverage of P. Simply select the optimal line covering each set P_i. We then try all possible coverages.

(2) For each edge $e_{ij} = (v_i, v_j$ where $i < j$, let the weight of e_{ij} equal $\Delta_2(f, \{p_i, p_{i+1}, \ldots, p_j\})$ for the line $f^*_{i,j}$ which minimizes this error.
(3) Run a modified version of the Bellman-Ford algorithm on this graph. Each iteration of edge relaxations will proceed almost as normal, but we will only perform k iterations of this process. In addition, in each iteration, updates will be made to a copy of the current graph to ensure that all paths grow by at most 1 edge in each iteration. When complete, the we will observe the shortest path from v_0 to v_n, which we will denote as the vertices of this path between V_0 and V_n as $B = \{b_1, b_2, \ldots b_{k-1}\}$. We now report the optimal k piece piecewise linear function as

$$f(x) = \begin{cases} f^*_1(x) & x < b_1 \\ f^*_2(x) & b_1 \leq x < b_2 \\ \vdots & \\ \vdots & \\ f^*_k(x) & b_{k-1} \leq x \end{cases}$$

Claim 1. *Given any set of points P and a set of breakpoints, B, for any piece of the optimal piecewise linear function covering these points,$f^*(x)$, $f^*_m(x) = f^*_{i,j}(x)$ where $b_m \leq x_i < x_j \leq b_{m+1}$ and $x_{j+} > b_{m+1}$*

Proof. This follows by simple contradiction. Assume the contrary, that for some optimal function f^*, the piece f^*_m is not the least error line that covers the subset of P falling in the interval $[b_m, b_{m+1}]$. Replace this piece by fi, j^* as described above. Then, by the definition of our error function, we see that we have decreased the error of f^*. Thus, since this contradicts the assumption that f^* was optimal, the above claim holds. □

Note that, given a set of points, P, the single (not piecewise) linear function f^* which minimizes $\Delta_2(f, P)$ is, by definition, the Least Squares Regression Line (LSRL) over these points. Thus, any time $f*$ was mentioned above, this line is actually the LSRL over the points in question.

Claim 2. *Given a sequence of n points, P, we can calculate the LSRL between all pairs of points in $O(n^2)$ time.*

Proof. For every interval, I we will calculate a constant number of statistics. Namely, the regressors of the LSRL for the interval, the sum of the squared errors, the number of points in the interval, the variances of the x and y coordinates, $\sigma^2_{I_x}$ and $\sigma^2_{I_y}$, and means of the x and y coordinates of the points, μ_{I_x} and μ_{I_y}, and the covariance of the x and y coordinates of the points, $cov(x, y)$. These are values that are easily initialized in constant time for every interval containing two adjacent points. We can also define a procedure merge(a,b) that will calculate all of these statistics for the interval defined by the union of two adjacent intervals a and b.

The merge procedure is straightforward for some of the required statistics, and requires more careful calculation for others. The mean of the new interval is simple a weighted average of the means of the two subintervals. The number of points in the new interval is easy to calculate. To merge to variances, one can use the identity noted in Chan et al. [2], namely for two point sets A and B, if we let $\delta = \mu_B - \mu_A$ then

$$\sigma^2_{A\cup B} = \sigma^2_A + \sigma^2_B + \delta^2 \frac{|A||B|}{|A \cup B|}.$$

A similar calculation applies for covariance. Finally, one must find the sum of squared errors. For linear regression:

$$1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \mu_y)^2} = R^2 = \frac{cov(x, y)}{\sigma_x \sigma_y}$$

We can calculate the right hand side of this equation from the statistics we have already calculated. The denominator of the fraction on the right hand side is proportional to the variance of the y-coordinates. Thus, we can calculate the sum of the squared residuals, as desired.

It is simple to calculate all of these statistics for an interval of two adjacent points. We begin by doing so for every pair of adjacent points. We can now employ a simple dynamic programming scheme. We progressively fill in each diagonal of a DP table, using the recurrence: $V(i, j) = merge(V(i, j - 1), V(j - 1, j))$ This procedure takes $O(n^2)$ time, as there are $O(n^2)$ intervals, and the above procedure clearly takes constant time to complete for each interval.

□

2.2. **Finding the Optimal Function.** We now have everything we need to solve the problem. We know, for a given interval, which line to choose, and we can calculate these lines. We know that the optimal function has breakpoints at a subset of the input points. We will say, then, that the input points represent a set of candidate points, C. We will now construct the graph $G(C, E)$ where $E = \{(c_i, c_j)|i < j\}$, and the weight of each edge $(c_i, c_j) = V(i, j)$. We now use the Bellman-Ford algorithm to find the shortest path of length k in this graph from c_0 to c_n. To do this, instead of doing $O(n)$ iterations of relaxing every edge, we will do exactly k. Since potential paths grow by 1 in each iteration, the length of the paths after k iterations will be exactly k. To see why this must be optimal, consider a function of at most k pieces with lower error than that returned by this algorithm. We know that there is a function with the same error as this function whose

breakpoints are a subset of C. We also can see that the pieces of this function would create a path from c_0 to c_n, and that the length of this path would be equal to the error of the function. This contradicts the fact that Bellman-Ford returns the shortest path of length k. Thus, the function returned here is the optimal k-piece discontinuous, piecewise linear function.

2.3. **Running Time.** We showed that we can construct the graph $G(V, E)$ in $O(n^2)$ time, carefully finding all edge weights and the associated optimal lines by dynamic programming. We now have a graph of with a vertex set of size n, and an edge set of size n^2. We relax every edge in each of the k iterations of the modified Bellman-Ford algorithm described above. Thus, the entire algorithm will run in $O(n^2k)$ time.

3. An Exact Algorithm

We will now discuss an exact algorithm for constructing the k-piece, continuous piecewise linear function which is optimal in the L_2-norm. We will consider the case of finding the optimal function over a particular coverage of a point set, and then extrapolate these results to find the optimum function over all coverages.

3.1. **A Convex Program.** As described above, consider the case where we are asked to find the optimal k piece function over a particular coverage of the sequence P of n points. Note that according to our earlier definition, two functions of identical coverage need not have identical sets of breakpoints. Specifically, each breakpoint $b_j \in B$ lies between two consecutive points p_i and p_{i+1}. The coverage of any two function f and g are identical as long as any two breakpoints b_j^f and b_j^g lie between the same two points. Thus, we can define a particular coverage by a set of points $T = \{t_1, t_2, \dots t_{k-1}\}$ where $T \subset P$ and $t_j = (p_q, p_{q+1})$. The goal of minimizing error for a particular coverage can then be written as the following convex program:

$$\text{minimize} \sum_{n=1}^{k} \sum_{p_i \in P_n} (y_i - m_n x_i + c_n)^2$$

$$\text{Subject to: } x_q \leq \frac{c_{n-1} - c_n}{m_n - m_{n-1}} \leq x_{q+1} \qquad \text{For all } t_j \in T$$

3.2. **Two Piece Case.** First, we consider the case where, given a sequence of points, P, we wish to cover the points with a continuous chain of two linear pieces. This is just an instance of the convex program defined above. Let us consider the case where we restrict the sole breakpoint for this function to lie in the vertical strip between x_i and $x_i + 1$ for some $0 < i < n$. We will then repeat the process for all points, and choose the minimum covering over all points. We can define such a coverage be defining the set I to have one element that describes the pair of adjacent points used to partition P. Since there is only one such strip in this case, we will refer to the bounds of this strip as x_L and x_R for simplicity.

3.3. **The Algorithm.** First we define the procedure for the case where $k = 2$.

(1) Choose a pair of consecutive points p_i, p_{i+1} from P. Let $T = \{(p_i, p_{i+1})\}$.
(2) Construct the Least-Squares Regression Line (LSRL) over both P_1 and P_2. If these lines intersect between x_i and x_{i+1}, we have found the optimal two piece function for this coverage.
(3) Otherwise, since the objective function is convex, and has a a global minimum lying outside the feasible region, we can see that the optimal solution lies on the boundary of the feasible region. This implies that one of the constraints of our convex program has become tight. Try making each of these constraints tight (one at a time) and solve for the optimal solution in each case. Making a constraint tight leaves the following constrained optimization problem:

$$\text{minimize } z = \sum_{p_i \in P_1} (y_i - m_1 x_i + c_1)^2 + \sum_{p_i \in d_2} (y_i - m_2 x_i + c_2)^2$$

$$\text{Subject to: } x_q = \frac{c_1 - c_2}{m_2 - m_1}$$

We can differentiate with respect to all four variables. Setting these derivatives equal to zero gives a system of four equations and four unknowns as follows:

$$\frac{\partial z}{\partial m_1} = -2 \sum_{p_i \in P_1} (m_1)(y_i - m_1 x_i + c_1) = 0$$

$$\frac{\partial z}{\partial m_2} = -2 \sum_{p_i \in P_2} (m_2)(y_i - m_2 x_i + c_2) = 0$$

$$\frac{\partial z}{\partial c_1} = 2 \sum_{p_i \in P_1} (y_i - m_1 x_i + c_1) = 0$$

$$x_q = \frac{c_1 - c_2}{m_2 - m_1}$$

Which we solve analytically. Try making both constraints tight in this manner and retain the solution which gives a lower value of the objective function.

(4) repeat this for all coverages of P

Now, we will use this as a subroutine of a dynamic programming solution to find the optimal solution. We will proceed by finding the optimal function for a particular coverage of P and then trying all possible coverages. Let $\Delta_2[i, j]$ with $i \leq j$ represent the error of the of the function $f^*_{i,j}$ which minimizes $\Delta_2(f, P_{i,j})$ where $P_{i,j} = \bigcup_{m=i}^{j} P_i$.

We will consider two functions $f^*_{i,j}$ and $f^*_{j+1,m}$ to have formed a **valid** solution if the intersection of the functions occurs in the interval $[x_j, x_{j+1}]$. The solution that results from

merging these two functions can then be written as

$$f^*_{i,m}(x) = \begin{cases} f^*_{i,j}(x) & x \leq b_j \\ f^*_{j+i,m}(x) & x > b_i \end{cases}$$

If these two functions do not form a valid solution, then their intersection lies outside the interval $[x_j, x_{j+1}]$. If their intersection lies to the left of this interval, then we can show that the left side of the inequality in the convex program representation of the problem becomes tight in the optimal solution. Likewise, if it lies to the right, the right side of the inequality will become tight. Thus, for every partition of a given coverage into two locally optimal functions, we can note which constant will become tight if these functions do not form a valid solution. We will note any constraint which must be tight in the optimal solution as the algorithm runs, and we can then make all these constraints tight if no valid solution is found.

For the case where $k > 2$, then, we run the following procedure for every coverage of P:

function OPT(k,P)
  Δ_2 = a k by k dimensional matrix
  for $i = 1 \to n - 1$ **do**
    $\Delta_2[i, i+1]$ = The optimal two piece covering of $P_i \cup P_{i+1}$
  end for
  for $j = 2 \to k - 1$ **do**
    $i = 1$
    while $i + j \leq k$ **do**
      for $m = i \to i + j$ **do**
        if $f^*_{i,m}$ and $f^*_{m+1,j}$ form a valid soltuion **then**
          $\Delta_2[i, i+j] = \Delta_2[i, m] + \Delta_2[m+1, j]$
          Break
        end if
        Otherwise, note which constraint must be tight in the optimal solution
      end for
      if $\Delta_2[i, i+j]$ =null **then**
        Tighten all constraints which were found to be tight in the optimal solution.
        Solve the resulting constrained optimization problem
      end if
      $i++$
    end while
  end for
  return $\Delta_2[1, k]$ and its associated function
end function

Claim 3. *Let f^*_i be the line which minimizes over $P_i \subseteq P$. The function f consisting of$f_1 = f^*_1$ and $f_2 = f^*_2$ is a global minimum of the objective function, and is therefore optimal if it represents a feasible solution.*

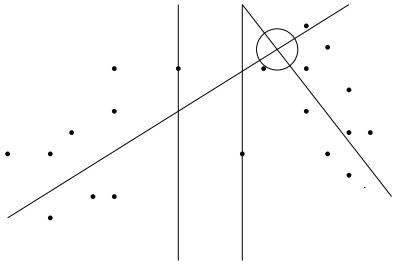

FIGURE 3. Begin by constructing the optimal regression line covering each side of the partition

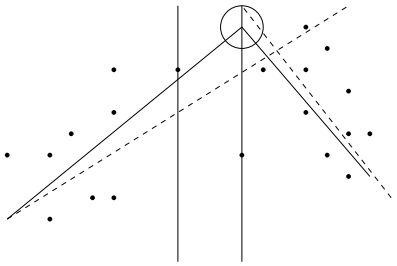

FIGURE 4. If this intersection point lies outside the strip defining a given partition, the breakpoint of the optimal function, f^*, lies on the boundary of this strip

Proof. The argument here is very similar to claim 1. Having any other function producing lower error would imply that the error of this function in at least one of the pieces was less than the error of the LSRL for the points in that interval, which is impossible. Since this solution thus represents a global minimum for the objective function, it must be optimal if it is feasible. □

Claim 4. *If the function f described in claim 3 is infeasible, then for the optimal two piece function f', either $B = \{x_L\}$ or $B = \{x_R\}$.*

Proof. Note that the objective function is convex. Thus, connecting this minimum point to any other point on the objective has a positive slope. Now assume that the optimal feasible solution lies somewhere in the interior of the feasible region. We have established that the objective function has a global minimum. Furthermore, the solution producing this error is unique. Thus, any other point on the function must have a greater value than this minimum. Since a line between any two points on the function lies wholly above the function itself, the line connecting the global minimum to this point has positive slope. By assumption, the global minimum lies outside the feasible region, and the point we have selected lies within it. Since the line connecting these points is continuous, this line must cross the boundary of the feasible region at some point. The point where this line crosses the feasible region is both feasible and gives a lower value of the objective function. Thus, the optimal solution lies on the boundary of the feasible region in this case. This implies that at least one constraint in our convex program is tight. Since there are only two, and they are mutually exclusive, we know exactly one constraint is tight. Thus, the optimal solution lies either on the line $y = x_L$ or $y = x_R$. □

Claim 5. *If two functions $f^*[1, j]$ and $f^*[j + 1, k]$ form a valid solution for a particular coverage $\mathcal{P}$ of P f^*, then this function is the optimal k piece covering of P*

Proof. This claim is similar to claims 1 and 3 and follows much of the same logic. We assume the functions $f^*[i,j]$ and $f^*[j+1,k]$ to be optimal. If there exists a function over the coverage with lower error than the sum of the errors of the two functions in question, it would imply that either the first j pieces of $\mathcal{P}$ or the last $k-j$ pieces of $\mathcal{P}$ are covered with lower error in this optimal solution. Thus, any valid solution must represent an optimal covering of P. □

Claim 6. *If two functions $f^*[i,j]$ and $f^*[j+1,k]$ do not form a valid solution, the intersection of f_j^* and f_{j+1}^* must lie on the line $y = x_j$ if the intersection point lies left of the interval $[x_j, x_{j+1}]$, and must lie on $y = x_{j+1}$ if it lies to the right of this interval.*

Proof. This will follow from the same convexity argument made in claim 4. Consider the invalid solution created by merging the two given functions. If we remove the constraints the the intersection of these functions must occur in the interval $[x_j, x_{j+1}]$, we know we have an otherwise feasible solution, since both functions were feasible solutions to their subproblems. Now consider the case where these functions intersect to the left of the interval $[x_j, x_{j+1}]$. Clearly, adding the constraint the the intersection must occur to the left of the line $y = x_{j+1}$ does not change the feasibility of our invalid solution. However, the final constraint that the intersection must occur to the right of the line $y = x_j$ invalidates our solution. Thus, the only constraint which lies between our infeasible solution and the interior of the feasible region is this final constraint. We now make the convexity argument from claim 4. Choosing any point on the interior of the feasible region, we know that the line between this point and our infeasible solution must have positive slope, otherwise our infeasible solution would not be the union of two locally optimal functions. This line must intersect the feasible region, and we know that when it does, the only constraint which will be tight is that which requires the intersection point to lie right of line $y = x_j$ Thus, we again see that no point on the interior can be optimal, and in fact the optimal point must lie on this line. An analogous argument follows for when the intersection point violates the other side of the inequality in question. □

The correctness of the algorithm follows from the above two claims. If any valid solution is found by the algorithm, this solution must be the optimal function for a particular coverage of P. If no valid solution is found, it means that every intersection point has been shown to lie on a particular vertical line. We can thus turn each pair of inequalities into an equality (based on which side of the inequality was violated) and solve a constrained optimization problem analogous to that of the two piece case. Repeating this process for every coverage of P and selecting the function of lowest error will then give the optimal k piece, continuous, piecewise linear function over P.

3.4. **Analysis.** For a given partition of P, we must complete k^2 entries in a dynamic programming table. Each entry will take time linear in n to compute the objective function, and an additional $O(k^3)$ time to solve a constrained optimization problem of $O(k)$ variables. There are $O(n^{O(k)})$ coverages of a point set P, since any pair of adjacent points can be selected to define a strip with which to partition the point set. Thus, the running time of the algorithm is $O(n^{O(k)}(k^5 + n))$.

Theorem 1. *We can construct the optimal continuous piecewise linear function of at most k pieces in $O(n^{O(k)}(k^5 + n))$ time*

4. An Approximation Algorithm

We will now consider an algorithm to find a function whose error is at most factor $(1+\epsilon)$ of the optimal error. The algorithm will proceed as follows. First we will show how to bound the error contributed by any single point in terms of Δ_∞. Then, based on the bound, we will create a set of discrete grid marks G over the point set, P. We will then show that for any function f, f can be transformed so that every piece of f passes through at least two grid marks with out increasing the error by a factor more than $(1+\epsilon)$. We then develop a dynamic programming algorithm to find the lowest error, k-piece continuous piecewise linear function passing through points of G, which must be a $(1+\epsilon)$ approximation of the optimal function over P.

4.1. **The Algorithm.**

(1) We construct a set of grid marks G of size $O(n \log n)$. We then draw all lines through pairs of grid marks in G. We see that some function which has error at most a factor of $(1+\epsilon)$ of the optimal function consists of some subset of these lines.
(2) Let $T(j, p, q)$ represent the piecewise linear function of at most j pieces, with the last piece passing through grid marks p and q. Also, given $p, q, r, s \in G$, let us define $\phi(p, q, r, s)$ as the error of the line through through r and s over the points to the right of the intersection of this line with the line through p and q. We see that $T(i, p, q) = \min_{r,s \in V \subseteq G} T(j-1, r, s) + \phi(p, q, r, s) - \phi(r, s, p, q)$ which can be solved by dynamic programming.

4.2. **Discretization Scheme.**

Claim 7. *Let $\Delta_2(f^*, P)$ denote the error of the optimal k piece continuous function f^* over a point set P, while $\Delta_\infty(f^{**}, P)$ is the minimum be error of the analogous function over the L_∞-norm. Then $\Delta^2_\infty(f^{**}, P) \leq \Delta_2(f^*, P) \leq n\Delta^2_\infty(f^{**}, P)$.*

Proof. For the first inequality, we know that the function f^* has at least one point p_i such that $|f(x_i) - y_i| \geq \Delta_\infty(f^{**}, P)$. If this was not the case, then this solution would produce a value of the error function in the L_∞-norm less than Δ_∞, which is a contradiction, since $\Delta_\infty(f^{**}, P)$ is minimal. Likewise, consider the function f^{**}. If we measure this error in the L_2 norm, since the maximum distance between any point and f is Δ_∞, this solution would produce an error in the L_2-norm of at most $n\Delta_\infty(f^{**}, P)^2$. Since $\Delta_2(f^*, P)$ is minimal, it must be at least this low. □

4.2.1. *Drawing the Grid.* We now construct a vertical line V_i through each point in $p_i \in P$. Centered at each y_i we place grid points at

$$y_i \pm (1+\delta)^j \Delta^2_\infty \quad \forall 0 \leq j \leq m$$

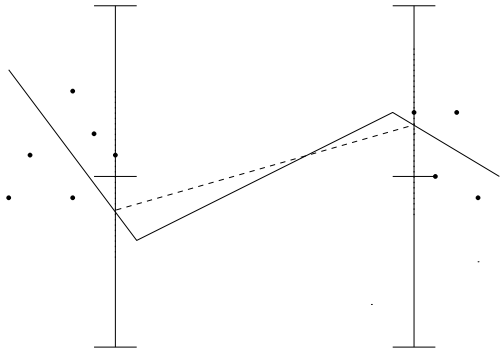

FIGURE 5. adjust lines covering zero points so that they will cover the closest two neighboring points

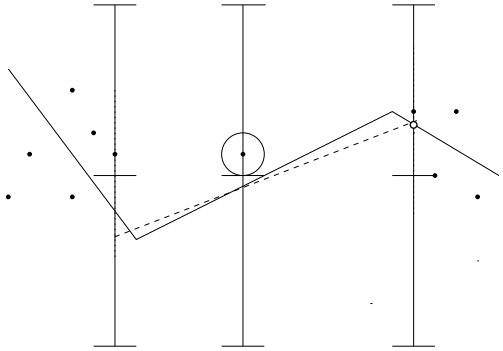

FIGURE 6. adjust lines covering one point so that the line passes through the closest interval

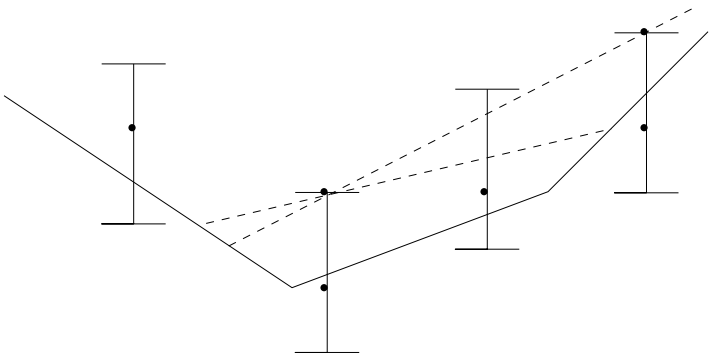

FIGURE 7. Translate each piece vertically until it intersects a grid mark. Then, rotate this piece until it again hits a grid mark. Note that no piece can leave an interval in which it lays under these conditions

where

$$y_i \pm (1+\delta)^{m-1}\Delta_\infty^2 \leq n\Delta_\infty^2 \leq y_i \pm (1+\delta)^m \Delta_\infty^2$$

4.3. **The Transformation Algorithm.** Assume we have a sequence of n points, P, and the set of corresponding grid marks as described above, G. Let a continuous piecewise function of at most k pieces, f, be given.

(1) If a given piece of f, f_j, does not cover any points in P, we will adjust f_j to be the line through the point covered by f_{j-1} which lies furthest to the right and the point covered by f_{j+1} which lies furthest left.

(2) If f_j covers exactly one point, p_i, and passes between no other grid marks, rotate the line about the point $(x_i, f_j(x_i))$ until one of its associated breakpoints b_{j-1} or b_j lies above or beneath the leftmost point covered by either f_{j-1} or f_{j+1}.
(3) Now, for each piece, f_i of f, translate f_i vertically until it intersects a grid mark. Then, rotate f_i around this grid mark until it intersects another grid mark. If any piece rotates past the end of an adjacent piece, we can remove the adjacent piece from the function. We are now left with a new function, f'.

Claim 8. *For any piece f_j of f which intersects a grid mark $g_i \in G$, rotating f_i about g_i until it intersects another grid mark $g_q \in G$ can not result in f_j becoming vertical*

Proof. The case where the size of the point set P is 1 results in a trivial function f which minimizes error, so without loss of generality, assume that there are multiple points in P. Now, consider the case where the line f_j covers multiple points. For each point p_i covered by f_j, we are guaranteed that f_j passes between two grid marks on the line V_i. Thus, rotating f_j about some g_i, we know that f_j passes through some interval not on the line V_i that will bound the rotation of f_j and prevent it from becoming vertical. In the rare case where f_j covers exactly one point, if g_i lies outside the interval covered by f_j, then the point that is covered by f_j provides us with the same guarantee as before. If g_i is above or below the one point covered by f_j, then we know that f_j passes between at least one other pair of grid marks which again bounds the rotation of f_j. Thus, no line will become vertical during rotation. □

Claim 9. *After the Transformation Algorithm is performed on some function f, we are left with a new function f' such that every piece of f' passes through two elements of G and $\frac{1}{(1+\epsilon)}\Delta_2(f, P) \leq \Delta_2(f', P) \leq (1+\epsilon)\Delta_2(f, P)$*

Proof. First, we consider translating some line f_i vertically upwards until it intersects a grid mark, and then rotating it around this grid mark until it intersects yet another grid mark. Consider an adjacent line f_j. For any point p, the coverage of p can only change if f_i and f_j pass between the same pair of grid marks on the vertical line V through p. To see this, consider the case where f_i and f_j pass through distinct pairs of grid marks on V. The coverage of p is said to change when the intersection point between f_i and f_j changes from occurring on one side of V to the other. Now consider the translation which causes the intersection point to lie directly on V. At this point, we know that both lines pass through the same pair of grid marks on V. Since this was not initially the case, and the translation was a continuous motion, this translation would require one of the lines to cross a grid mark. The same argument applies to rotation. If the rotation is causing the intersection point between f_i and f_j to move closer to V, then we again have a continuous motion, and can again see that both lines would lie in the same interval when this intersection point is forced to lie on V. We can then say again that the coverage of p will only change if these lines lay in the same interval on V initially. Finally, since f_j is not vertical after rotation, and the original function f was x-monotone by assumption, no point originally covered by f_{j+1} can become covered by f_{j-1} or vice versa. Thus, the only case where the coverage of

p changes is when both lines pass through the same interval on V.

□

After performing this algorithm on each piece, we are left with some new function f'. We can see that for any point, p_i, the line covering p_i in f' lies between the same two grid marks on V_i that the line covering p_i in f laid in. First, consider points for which the line covering that given point did not change. Since no line left an interval (to do so would require crossing a grid mark, which cannot happen by design), these points are still covered by some line passing through the same interval. Now consider points for which the coverage did change. We showed above that the new line covering this point passes through the same interval as the old line. Since this new line can never leave an interval through which it passes, we are guaranteed that the line covering one of these points passes through the same interval that it did before the transformation. We can now use our error bound for Δ_2 to see that this fact implies that our desired error bound holds.

We begin by partitioning P into

$$P^c = \{p_i \in P | \Delta_2(f, p_i) \leq \delta \Delta_\infty(P)^2\}$$

and

$$P^f = P \backslash P^c$$

For any $p_i \in P^c$ by construction, $|\Delta_2(f, p_i) - \Delta_2(g, p_i)| \leq \delta \Delta_\infty(p_i)^2$ which implies that

$$\sum_{p_i \in P^c} |\Delta_2(f, P) - \Delta_p(g, P)| \leq n\delta \Delta_\infty(P)^2$$

Consider $p_i \in P^f$

Since the lines lie in the same interval, there exists a j such that

$$y_i \pm (1+\delta)^{j-1}\Delta_\infty^2 \leq \Delta_2^2(f, P), \Delta_2^2(g, P) \leq y_i \pm (1+\delta)^j \Delta_\infty^2$$

Thus, $\Delta_2(f, p_i) \leq (1+\delta)\Delta_2(g, p_i)$. Summing Over all i, we see that

$$\Delta_2(f, P) \leq (1+\delta)\Delta_2(g, P^f)$$

In total, then,

$$\begin{aligned}
\Delta_2(f, P) &= \Delta_2(f, P^c) + \Delta_2(f, P^f) \\
&\leq n\delta\Delta_\infty(P)^2 + (1+\delta)\Delta_2(g, P^f) \\
&\leq n\delta\Delta_\infty(P)^2 + (1+\delta)\Delta_2(g, P) \\
&\leq n\delta\Delta_2(g, P) + (1+\delta)\Delta_2(g, P) \\
&\leq \Delta_2(g, P)(1 + (n+1)\delta
\end{aligned}$$

Choosing $\delta = \frac{\epsilon}{n+1}$ gives that

$$\Delta_2(f, P) \leq (1+\epsilon)\Delta_2(g, P)$$

as desired. To get the inverse relationship, we can perform the same proof, but switch the positions of f and g.

We now apply Lemma 4.4 to our function f and f' and see that we have the desired error

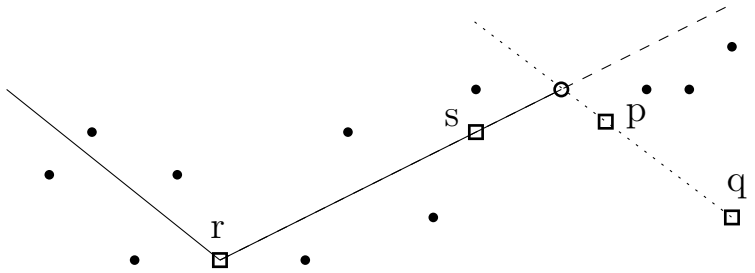

FIGURE 8. To get the error of the function formed by adding the line through p and q to the optimal function ending in a piece through r and s we take $T(i-1,r,s)$, subtract the error from the top dashed line, and add the error from the dotted line below it. The error contributed by the dashed line is $\phi(r,s,p,q)$. The error contributed by the dotted line is $\phi(p,q,r,s)$.

bound described in claim 9, with the additional feature that every piece of f' is guaranteed to pass through two elements of G.

4.4. **Dynamic Programming.** We are now ready to find our approximation of the optimal function. We wish to find the k piece function of minimum error such that each of the k pieces passes through a pair of grid marks. Using the notation from above, we let $T(j,p,q)$ be the function of minimal error consisting of j pieces whose last piece passes through the grid marks $p,q \in G$. Consider the last piece in this function, which is fixed, and passes through grid marks p and q. It is obvious that the first $j-1$ pieces of this function belong to the function which is the minimal $j-1$ piece covering whose last piece passes through some pair other pair of grid marks r and s. If this were not the case, then changing the first $j-1$ pieces to be one of these functions would reduce the overall error. Thus, we must simply minimize over all pairs of grid marks r and s the error of a function whose first $j-1$ pieces are the optimal $j-1$ piece function ending in a line through r and s, and whose final piece is the line through p and q. To get the error of such a function, however, we must do some cleanup. First, we must add the error contributed by the new ith piece which passes through p and q. Also, since $T(i-1,r,s)$ includes the error between these points and the line through r and s, we must subtract this error so that the error from these points is not counted twice. Thus, we add $\phi(p,q,r,s)$, the error contributed by points which are now covered by the new line, and subtract $\phi(r,s,p,q)$, the error between these same points and the line through r and s which formerly covered them. Finally, we will only consider cases where there intersection of the line through r and s and the line through p and q occurs to the right of the final breakpoint of $T(j-1,r,s)$. We will call a line and a piecewise continuous function **compatible** in this case, and define the set $V_{i,p,q}$

to be the set of pairs of points (r, s) such that $T(i-1, r, s)$ is compatible with the line through p and q. This produces the following recurrence relation:

$$T(i, j, p, q) = \min_{r,s \in V_{i,p,q}} T(j-1, r, s) + \phi(p, q, r, s) - \phi(r, s, p, q)$$

as given in step (2) of the algorithm above. Note that we can sum these error since we are only considering compatible solutions, which implies that the addition of the ith piece will change the error only of points previously covered by the $(i-1)$th piece of a function. Solving for $T(i, p, q)$ for each pair of points from $i = 1$ to k means that for a given value of i, solutions to all necessary subproblems will already exist when solving for the set of functions of at most i pieces (all subproblems of $i-1$ pieces will already have been solved). After solving all subproblems, the function of k pieces which gives the lowest error is a $(1+\epsilon)$ approximation of the optimal k piece function.

4.5. **Running Time.** The above approximation algorithm will run in $O((\frac{1}{\epsilon} n \log(n))^4 k)$. We can see that each vertical line in our grid passes through $O(\frac{1}{\epsilon} \log n)$ grid marks. Thus, G consists of $S = O(\frac{1}{\epsilon} n \log n)$ grid marks. There are then $O(S^2)$ lines passing through pairs of the points. It then takes $O(n^2)$ time to compute the error over every interval for each of these lines. Finally, we run a dynamic programming algorithm which fills in a table of size $O(S^2 k)$. We must consider each of the $O(S^2)$ lines to fill in each cell in the table. This gives an overall running time of $O(S^4 k)$.

Theorem 2. *We can construct a continuous piecewise linear function of at most k pieces covering P with error at most a factor of $(1+\epsilon)$ of the optimal such function over these points in $O((\frac{1}{\epsilon} n \log(n))^4 k)$ time.*

5. Conclusion

We have thus provided a method for approximating the optimal continuous k-piece piecewise linear function for a sequence of points, and an exact algorithm for finding the optimal discontinuous k-piece piecewise linear function. There are two main directions for future research. The first is to reduce the dependence of the exact algorithm on k. Namely, to come up with a more subtle argument than simply minimizing error over all partitions. Some dynamic programming argument could possibly be made which would eliminate this need and solve the problem in polynomial time.

The second area for future work is finding a coreset for the problem. A coreset refers to a systematically chosen subset of the input points with the property that such a subset always changes error by no more than some factor. The general idea would be to show that a coreset for the problem of finding such a regression line exists, and then perform the above methods on a this subset instead of the entire input set. Har-Peled shows how to find coresets for a class of problems similar to this, and it is likely that this result could be extended to include our problem [6]. It is likely, then, that a $(1+\epsilon)$ approximation could be found in $O(n + k^{O(1)})$ time.

Although the above techniques do not necessarily provide a solution with a practical

running time, they do provide a direction for future research into a problem with a previously unknown result. Through future insights, these methods could form the basis for efficient and accurate simplification of curves in the L_2-norm.

References

[1] Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

[2] Tony F Chan, Gene H Golub, and Randall J LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, pages 30–41. Springer, 1982.

[3] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[4] Jean H Gallier. *Curves and Surfaces in Geometric Modeling: Theory & Algorithms*. Morgan Kaufmann Pub, 2000.

[5] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 471–475. ACM, 2001.

[6] Sariel Har-Peled. Coresets for discrete integration and clustering. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pages 33–44. Springer, 2006.

[7] Hiroshi Imai and Masao Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of information processing*, 9(3):159–162, 1987.

[8] Piotr Indyk, Reut Levi, and Ronitt Rubinfeld. Approximating and testing k-histogram distributions in sub-linear time. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 15–22. ACM, 2012.

[9] HV Jagadish, Nick Koudas, S Muthukrishnan, Viswanath Poosala, Ken Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proceedings of the International Conference on Very Large Data Bases*, pages 275–286. INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS, 1998.

[10] RJ Oosterbaan, DP Sharma, KN Singh, and KVGK Rao. Crop production and soil salinity: evaluation of field data from india by segmented linear regression with breakpoint. In *Proc. Symp. Land Drainage for Salinity Control in Arid and Semi-Arid Regions III. Drainage Research Institute, Cairo*, pages 373–383, 1990.

[11] S. Sankararaman, P. K. Agarwal, T. Mølhave, and A. P. Boedihardjo. Computing Similarity between a Pair of Trajectories. *ArXiv e-prints*, March 2013.