How good is the Chord algorithm?*

Constantinos Daskalakis† Ilias Diakonikolas‡ Mihalis Yannakakis§

September 26, 2013

Abstract

The Chord algorithm is a popular, simple method for the succinct approximation of curves, which is widely used, under different names, in a variety of areas, such as, multiobjective and parametric optimization, computational geometry, and graphics. We analyze the performance of the Chord algorithm, as compared to the optimal approximation that achieves a desired accuracy with the minimum number of points. We prove sharp upper and lower bounds, both in the worst case and average case setting.

1 Introduction

Consider a typical design problem with more than one objectives (design criteria). For example, we may want to design a network that provides the maximum capacity with the minimum cost, or we may want to design a radiation therapy for a patient that maximizes the dose to the tumor and minimizes the dose to the healthy organs. In such *multiobjective* (or *multicriteria*) problems there is typically no solution that optimizes simultaneously all the objectives, but rather a set of so-called *Pareto optimal* solutions, i.e., solutions that are not dominated by any other solution in all the objectives. The trade-off between the different objectives is captured by the *trade-off* or *Pareto curve* (surface for three or more objectives), the set of values of the objective functions for all the Pareto optimal solutions.

Multiobjective problems are prevalent in many fields, e.g., engineering, economics, management, healthcare, etc. There is extensive research in this area published in different fields; see [Ehr, EG, FGE, Mit] for some books and surveys. In a multiobjective problem, we would ideally like to compute the Pareto curve and present it to the decision maker to select the solution that strikes the 'right' balance between the objectives according to his/her preferences (and different users may prefer different points on the curve). The problem is that the Pareto curve has typically an enormous number of points, or even an infinite number for continuous problems (with no closed form description), and thus we cannot compute all of them. We can only compute a limited number of solutions

*An extended abstract of this work appeared in the *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pp. 978-991.

†CSAIL, MIT. Email: costis@csail.mit.edu. Work done while the author was a postdoctoral researcher at Microsoft Research New England.

‡School of Informatics, University of Edinburgh. Email: ilias.d@ed.ac.uk. Research partially supported by a Simons Postdoctoral Fellowship at UC Berkeley. Most of this work was done while the author was at Columbia University, supported by NSF grants CCF-04-30946, CCF-07-28736, and by an Alexander S. Onassis Foundation Fellowship.

§Department of Computer Science, Columbia University. Email: mihalis@cs.columbia.edu. Research supported by NSF grants CCF-07-28736, CCF-10-17955.

(points), and of course we want the computed points to provide a good approximation to the Pareto curve so that the decision maker can get a good sense of the range of possibilities in the design space.

We measure the quality of the approximation provided by a computed solution set using a (multiplicative) approximation ratio, as in the case of approximation algorithms for single-objective problems. Assume (as usual in approximation algorithms) that the objective functions take positive values. A set of solutions S is an *ϵ-Pareto set* if the members of S approximately dominate within $1+\epsilon$ every other solution, i.e., for every solution s there is a solution $s' \in S$ such that s' is within a factor $1+\epsilon$ or better of s in all the objectives [PY1]. Often, after computing a finite set S of solutions and their corresponding points in objective space (i.e., their vectors of objective values), we 'connect the dots', taking in effect also the convex combinations of the solution points. In problems where the solution points form a convex set (examples include multiobjective flows, Linear Programming, Convex Programming), this convexification is justified and provides a much better approximation of the Pareto curve than the original set S of individual points. A set S of solutions is called an *ϵ-convex Pareto set* if the convex hull of the solution points corresponding to S approximately dominates within $1+\epsilon$ all the solution points [DY2]. Even for applications with nonconvex solution sets, sometimes solutions that are dominated by convex combinations of other solutions are considered inferior, and one is interested only in solutions that are not thus dominated, i.e., in solutions whose objective values are on the (undominated) boundary of the convex hull of all solution points, the so-called *convex Pareto set*. Note that every instance of a multiobjective problem has a unique Pareto set and a unique convex Pareto set, but in general it has many different ϵ-Pareto sets and ϵ-convex Pareto sets, and furthermore these can have drastically different sizes. It is known that for every multiobjective problem with a fixed number d of polynomially computable objective functions, there exist an ϵ-Pareto set (and also ϵ-convex Pareto set) of polynomial size, in particular of size $O((\frac{m}{\epsilon})^{d-1})$, where m is the bit complexity of the objective functions (i.e., the functions take values in the range $[2^{-m}, 2^m]$) [PY1]. Whether such approximate sets can be constructed in polynomial time is another matter: necessary and sufficient conditions for polynomial time constructibility of ϵ-Pareto and ϵ-convex Pareto sets are given respectively in [PY1, DY2].

The most common approach to the generation of Pareto points (called weighted-sum method) is to give nonnegative weights w_i to the different objective functions f_i (assume for simplicity that they are all minimization objectives) and then optimize the linear combining function $\sum_i w_i f_i$; this approach assumes availability of a subroutine Comb that optimizes such linear combinations of the objectives. For any set of nonnegative weights, the optimal solution is clearly in the Pareto set, actually in the convex Pareto set. In fact, the convex Pareto set is precisely the set of optimal solutions for *all* possible such weighted linear combinations of the objectives. Of course, we cannot try all possible weights; we must select carefully a finite set of weights, so that the resulting set of solutions provides a good approximation, i.e., is an ϵ-convex Pareto set for a desired small ϵ. It is shown in [DY2] that a necessary and sufficient condition for the polynomial time constructibility of an ϵ-convex Pareto set is the availability of a polynomial time Comb routine for the approximate optimization of nonnegative linear combinations.

In a typical multiobjective problem, the Comb routine is a nontrivial piece of software, each call takes a substantial amount of time, thus we want to make the best use of the calls to achieve as good a representation of the solution space as possible. Ideally, we would like to achieve the smallest possible approximation error ϵ with the fewest number of calls to the Comb routine. That is, given $\epsilon > 0$, compute an ϵ-convex Pareto set for the instance at hand using *as few Comb calls as possible*. (In [DY2] we study a different cost metric; we explain the difference at the end of this section, and we note

that both metrics are relevant for different aspects of decision making in multiobjective problems.)

We measure the performance of an algorithm by the ratio of its cost, i.e., the number of calls that it makes, to the minimum possible number required for the instance at hand (as is usual in approximation algorithms). Let $\mathrm{OPT}_\epsilon(I)$ be the number of points in the smallest ϵ-convex Pareto set for an instance I. Clearly, every algorithm that computes an ϵ-convex Pareto set needs to make at least $\mathrm{OPT}_\epsilon(I)$ calls. The *performance (competitive) ratio* of an algorithm $\mathcal{A}$ that computes an ϵ-convex Pareto set using $\mathcal{A}(I)$ calls for each instance I is $\sup_I \frac{\mathcal{A}(I)}{\mathrm{OPT}_\epsilon(I)}$. An important point that should be stressed here is that, as in the case of online algorithms, the algorithm does not have complete information about the input, i.e., the (convex) Pareto curve is *not* given explicitly, but can only be accessed indirectly through calls to the Comb routine; in fact, the whole purpose of the algorithm is to obtain an approximate knowledge of the curve.

In this paper we investigate the performance of the *Chord* algorithm, a simple, natural greedy algorithm for the approximate construction of the Pareto set. The algorithm and variants of it have been used often, under various names, for multiobjective problems [AN, CCS, CHSB, FBR, RF, Ro, YG] as well as several other types of applications involving the approximation of curves, which we will describe later on. We focus on the bi-objective case; although the algorithm can be defined (and has been used) for more objectives, most of the literature concerns the bi-objective case, which is already rich enough, and covers also most of the common uses of the algorithm.

We now briefly describe the algorithm. Let f_1, f_2 be the two objectives (say minimization objectives for concreteness), and let P be the (unknown) convex Pareto curve. First optimize f_1, and f_2 separately (i.e., call Comb for the weight tuples $(1,0)$ and $(0,1)$) to compute the leftmost and rightmost points a, b of the curve P. The segment (a,b) is a first approximation to P; its quality is determined by a point $q \in P$ that is least well covered by the segment. It is easy to see that this worst point q is a point of the Pareto curve P that minimizes the linear combination $f_2 + \lambda_{ab} f_1$, where λ_{ab} is the absolute value of the slope of (a,b), i.e., it is a point of P with a supporting line parallel to the 'chord' (a,b). Compute such a worst point q; if the error is at most ϵ, then terminate, otherwise add q to the set S to form an approximate set $\{a,q,b\}$ and recurse on the two intervals (a,q) and (q,b). In Section 2 we give a more detailed formal description (for example, in some cases one can determine from previous information that the maximum possible error in an interval is upper bounded by ϵ and do not need to call Comb).

The algorithm is quite natural, it has been often reinvented and is commonly used for a number of other purposes. As pointed out in [Ro], an early application was by Archimedes who used it to approximate a parabola for area estimation [Ar]. In the area of *parametric optimization*, the algorithm is known as the "Eisner-Severance" method after [ES]. Note that parametric optimization is closely related to bi-objective optimization. For example, in the parametric shortest path problem, each edge e has cost $c_e + \lambda d_e$ that depends on a parameter λ. The length of the shortest path is a piecewise linear function of λ whose pieces correspond to the vertices of the convex Pareto curve for the bi-objective shortest path problem with cost vectors c, d on the edges. A call to the Comb routine for the bi-objective problem corresponds to solving the parametric problem for a particular value of the parameter.

The Chord algorithm is also useful for the approximation of convex functions, and for the approximation and smoothening of convex and even non-convex curves. In the case of functions, an appropriate measure of distance between the function f and the approximation is the vertical distance, while for curves a natural measure is the Hausdorff distance; note that for a given curve and approximating segment ab, the same point q of the curve with supporting line parallel to ab maximizes

also the above distances. In the context of smoothening and compressing curves and polygonal lines for graphics and related applications, the Chord algorithm is known as the Ramer-Douglas-Peucker algorithm, after [Ra, DP] who independently proposed it.

Previous work has analyzed the Chord algorithm (and variants) for achieving an ϵ-approximation of a function or curve with respect to vertical and Hausdorff distance, and proved bounds on the cost of the algorithm as a function of ϵ: for all convex curves of length L, (under some technical conditions on the derivatives) the algorithm uses at most $O(\sqrt{L/\epsilon})$ calls to construct an ϵ-approximation , and there are curves (for example, a portion of a circle) that require $\Omega(\sqrt{L/\epsilon})$ calls [Ro, YG].

Note however that these results do not tell us what the performance ratio is, because for many instances, the optimal cost OPT_ϵ may be much smaller than $\sqrt{L/\epsilon}$, perhaps even a constant. For example, if P is a convex polygonal line with few vertices, then the Chord algorithm will perform very well for $\epsilon = 0$; in fact, as shown by [ES] in the context of parametric optimization, if there are N breakpoints, then the algorithm will compute the exact curve after $2N - 1$ calls. (The problem is of course that in most bi-objective and parametric problems, the number N of vertices is huge, or even infinite for continuous problems, and thus we have to approximate.)

In this paper we provide sharp upper and lower bounds on the performance (competitive) ratio of the Chord algorithm, both in the worst case and in the average case setting. Consider a bi-objective problem where the objective functions take values in $[2^{-m}, 2^m]$. We prove that the worst-case performance ratio of the Chord algorithm for computing an ϵ-convex Pareto set is $\Theta(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)})$. The upper bound implies in particular that for problems with polynomially computable objective functions and a polynomial-time (exact or approximate) Comb routine, the Chord algorithm runs in polynomial time in the input size and $1/\epsilon$. We show furthermore that there is no algorithm with constant performance ratio. In particular, every algorithm (even randomized) has performance ratio at least $\Omega(\log m + \log\log(1/\epsilon))$.

Similar results hold for the approximation of convex curves with respect to the Hausdorff distance. That is, the performance ratio of the Chord algorithm for approximating a convex curve of length L within Hausdorff distance ϵ, is $\Theta(\frac{\log(L/\epsilon)}{\log\log(L/\epsilon)})$. Furthermore, every algorithm has worst-case performance ratio at least $\Omega(\log\log(L/\epsilon))$.

We also analyze the expected performance of the Chord algorithm for some natural probability distributions. Given that the algorithm is used in practice in various contexts with good performance, and since worst-case instances are often pathological and extreme, it is interesting to analyze the average case performance of the algorithm. Indeed, we show that the performance on the average is exponentially better. Note that Chord is a simple natural greedy algorithm, and is not tuned to any particular distribution. We consider instances generated by a class of product distributions that are "approximately" uniform and prove that the expected performance ratio of the Chord algorithm is $\Theta(\log m + \log\log(1/\epsilon))$ (upper and lower bound). Again similar results hold for the Hausdorff distance.

Related Work. There is extensive work on multiobjective optimization, as well as on approximation of curves in various contexts. We have discussed already the main related references. The problem addressed by the Chord algorithm fits within the general framework of determining the shape by probing [CY]. Most of the work in this area concerns the exact reconstruction, and the analytical works on approximation (e.g., [LB, Ro, YG]) compute only the worst-case cost of the algorithm in terms of ϵ (showing bounds of the form $O(\sqrt{L/\epsilon})$). There does not seem to be any prior work comparing the cost of the algorithm to the optimal cost for the instance at hand, i.e., the approximation ratio, which is the usual way of measuring the performance of approximation algorithms.

The closest work in multiobjective optimization is our prior work [DY2] on the approximation of convex Pareto curves using a different cost metric. Both of the metrics are important and reflect different aspects of the use of the approximation in the decision making process. Consider a problem, say with two objectives, suppose we make several calls, say N, to the Comb routine, compute a number of solution points, connect them and present the resulting curve to the decision maker to visualize the range of possibilities, i.e., get an idea of the true convex Pareto curve. (The process may not end there, e.g., the decision maker may narrow the range of interest, followed by computation of a better approximation for the narrower range, and so forth). In this scenario, we want to achieve as small an error ϵ as possible, using as small a number N of calls as we can, ideally, as close as possible to the minimum number $\mathrm{OPT}_\epsilon(I)$ that is absolutely needed for the instance. In this setting, the cost of the algorithm is measured by the number of calls (i.e., the computational effort); this is the cost metric that we study in this paper, and the performance ratio is as usual the ratio of the cost to the optimum cost. Consider now a scenario where the decision maker does not just inspect visually the curve, but will look more closely at a set of solutions to select one; for instance a physician in the radiotherapy example will consider carefully a small number of possible treatments in detail to decide which one to follow. Since human time is much more limited than computational time (and more valuable, even small constant factors matter a lot), the primary metric in this scenario is the number n of selected solutions that is presented to the decision maker for closer investigation (we want n to be as close as possible to $\mathrm{OPT}_\epsilon(I)$), while the computational time, i.e., the number N of calls, is less important and can be much larger (as long as it is feasible of course). This second cost metric (the size n of the selected set) is studied in [DY2] for the convex Pareto curve (and in [VY, DY] in the nonconvex case). Among other results, it is shown there that for all bi-objective problems with an exact Comb routine and a continuous convex space, an optimal ϵ-convex Pareto set (i.e., one with $\mathrm{OPT}_\epsilon(I)$ solutions) can be computed in polynomial time using $O(m/\epsilon)$ calls to Comb in general, (though more efficient algorithms are obtained for specific important problems such as bi-objective LP). For discrete problems, a 2-approximation to the minimum size can be obtained in polynomial time, and the factor 2 is inherent. As remarked above, both cost metrics are important for different stages of the decision making. Recall also that, as noted earlier, the Chord algorithm runs in polynomial time, and furthermore, one can show that its solution set can be post-processed to select a subset that is a ϵ-convex Pareto set of size at most $2\mathrm{OPT}_\epsilon(I)$.

Structure of the paper. The rest of the paper is organized as follows: Section 2 describes the model and states our main results, Section 3 concerns the worst-case analysis, and Section 4 the average-case analysis. Section 5 concludes the paper and suggests the most relevant future research directions.

2 Model and Statement of Results

This section is structured as follows: After giving basic notation, in Section 2.1 we describe the relevant definitions and framework from multiobjective optimization. In Section 2.2 we provide a formal description of the Chord algorithm in tandem with an intuitive explanation. Finally, in Section 2.3 we state our results on the performance of the algorithm as well as our general lower bounds.

Notation. We start by introducing some notation used throughout the paper. For $n, i, j \in \mathbb{Z}_+$, we will denote $[n] := \{1, 2, \ldots, n\}$ and $[i, j] := \{i, i+1, \ldots, j\}$. For $p, q, r \in \mathbb{R}^2$, we denote by pq the line segment with endpoints p and q, (pq) denotes its length, $\triangle(pqr)$ is the triangle defined by p, q, r and $\angle(pqr)$ is the internal angle of $\triangle(pqr)$ formed by pq and qr.

We will use x and y as the two coordinates of the plane. If p is a point on the plane, we use $x(p)$ and $y(p)$ to denote its coordinates; that is, $p = \big(x(p), y(p)\big)$. We will typically use the symbol λ to denote the (absolute value of the) slope of a line, unless otherwise specified. Sometimes we will add an appropriate subscript, i.e., we will use λ_{pq} to denote the slope of the line defined by p and q. For a Lebesgue measurable set $A \subseteq \mathbb{R}^2$ we will denote its area by $S(A)$.

2.1 Definitions and Background

We describe the general framework of a bi-objective problem Π to which our results are applicable. A bi-objective optimization problem has a set of valid instances, and every instance has an associated set of feasible solutions, usually called the solution or decision space. There are two objective functions, each of which maps every instance–solution pair to a real number. The problem specifies for each objective whether it is to be maximized or minimized.

Consider the plane whose coordinates correspond to the two objectives. Every solution is mapped to a point on this plane. We denote the objective functions by x and y and we use $\mathcal{I}$ to denote the objective space (i.e., the set of 2-vectors of objective values of the feasible solutions for the given instance). As usual in approximation, we assume that the objective functions are polynomial time computable and take values in $[2^{-m}, 2^m]$, i.e., $\mathcal{I} \subseteq [2^{-m}, 2^m]^2$, where m is polynomially bounded in the size of the input. Note that this framework covers all discrete combinatorial optimization problems of interest (e.g., shortest paths, spanning tree, matching, etc), but also contains many continuous problems (e.g., linear and convex programs, etc). Throughout this paper we assume, for the sake of concreteness, that both objective functions x and y are to be minimized. All our results hold also for the case of maximization or mixed objectives.

Let $p, q \in \mathbb{R}^2_+$. We say that p *dominates* q if $p \leq q$ (coordinate-wise). We say that p *ϵ-covers* q ($\epsilon \geq 0$) if $p \leq (1+\epsilon)q$. Let $A \subseteq \mathbb{R}^2_+$. The *Pareto set* of A, denoted by $\mathrm{P}(A)$, is the subset of undominated points in A (i.e., $p \in \mathrm{P}(A)$ iff $p \in A$ and no other point in A dominates p). The convex Pareto set of a A, denoted by $\mathrm{CP}(A)$, is the minimum subset of A whose convex combinations dominate (every point in) A. We also use the term *lower envelope* of A to denote the Pareto set of its convex hull, i.e., $\mathrm{LE}(A) = \mathrm{P}(\mathcal{CH}(A))$. In particular, if A is convex its lower envelope is identified with its Pareto set. If A is finite, its lower envelope is a convex polygonal chain with vertices the points of $\mathrm{CP}(A)$. Note that, for any set A, the lower envelope $\mathrm{LE}(A)$ is a convex and monotone decreasing planar curve. For $p, q \in \mathrm{LE}(A)$ we will denote by $\mathrm{LE}(pq)$ the subset of $\mathrm{LE}(A)$ with endpoints p, q.

An *ϵ-convex Pareto set* of A (henceforth ϵ-CP) is a subset $\mathrm{CP}_\epsilon(A)$ of A whose convex combinations ϵ-cover (every point in) A. Note that such a set need not contain points dominated by convex combinations of other points, as they are redundant. If a set contains no redundant points, we call it non-redundant. Let $S = \{s_i\}_{i=1}^k \subseteq \mathbb{R}^2_+$, $x(s_i) < x(s_{i+1})$ and $y(s_i) > y(s_{i+1})$, be a non-redundant set. By definition, S is an ϵ-CP for A if and only if the polygonal chain $\mathrm{LE}(S) = \langle s_1, \ldots, s_k \rangle$ ϵ-covers $\mathrm{CP}(A)$.

We define the *ratio distance* from a point p to a point q as $\mathcal{RD}(p,q) = \max\{x(q)/x(p) - 1, y(q)/y(p) - 1, 0\}$. (Note the asymmetry in the definition.) Intuitively, it is the minimum value of $\epsilon \geq 0$ such that q ϵ-covers p. We also define the ratio distance between sets of points. If $S_1, S_2 \subseteq \mathbb{R}^2_+$, then $\mathcal{RD}(S_1, S_2) = \sup_{q_1 \in S_1} \inf_{q_2 \in S_2} \mathcal{RD}(q_1, q_2)$. As a corollary of this definition, the set $S \subseteq A$ is an ϵ-CP for A if and only if $\mathcal{RD}(\mathrm{LE}(A), \mathrm{LE}(S)) = \mathcal{RD}(\mathrm{CP}(A), \mathrm{LE}(S)) \leq \epsilon$.

The above definitions apply to any set $A \subseteq \mathbb{R}^2_+$. Let Π be a bi-objective optimization problem in the aforementioned framework. For an instance of Π, the set A corresponds to the objective space $\mathcal{I}$ (for the given instance). We do not assume that the objective space $\mathcal{I}$ is convex; it may well be discrete

or a continuous non-convex set. It should be stressed that the objective space is not given explicitly, but rather implicitly through the instance. In particular, we access the objective space $\mathcal{I}$ of Π via an oracle Comb that (exactly or approximately) minimizes non-negative linear combinations $y + \lambda x$ of the objectives. That is, the oracle takes as input a parameter $\lambda \in \mathbb{R}_+$ and outputs a point $q \in \mathcal{I}$ (i.e., a feasible point) that (exactly or approximately) minimizes the combined objective function $h_\lambda(x, y) = y + \lambda x$. We use the convention that, for $\lambda = +\infty$, the oracle minimizes the x objective.

Formally, for $\lambda \in \mathbb{R}_+$, we denote by $\mathrm{Comb}(\lambda)$ the problem of optimizing the combined objective $h_\lambda(x, y)$ over $\mathcal{I}$. Let $\delta \in \mathbb{R}_+$ be an accuracy parameter. Then, for $\lambda \in \mathbb{R}_+$, we will denote by $\mathrm{Comb}_\delta(\lambda)$ a routine that returns a point $q \in \mathcal{I}$ that optimizes h_λ up to a factor of $(1 + \delta)$, i.e., $h_\lambda(q) \leq (1 + \delta) \cdot \min\{h_\lambda(p) \mid p \in \mathcal{I}\}$. In other words, the Comb_δ routine is an "approximate optimization oracle" for the objective space $\mathcal{I}$. We say that the Comb problem has a polynomial time approximation scheme (PTAS), if for any instance of Π and any $\delta > 0$ there exists a routine $\mathrm{Comb}_\delta(\lambda)$ (as specified above) that runs in time polynomial in the size of the instance. As shown in [DY2], for any bi-objective problem in the aforementioned framework, there is a PTAS for constructing an ϵ-convex Pareto set if and only if there is a PTAS for the Comb problem.

We now provide a geometric characterization of $\mathrm{Comb}_\delta(\lambda)$ that will be crucial throughout this paper. Consider the point $q \in \mathcal{I}$ returned by $\mathrm{Comb}_\delta(\lambda)$ and the corresponding line $\ell(q, \lambda)$ through q with slope $-\lambda$, i.e., $\ell(q, \lambda) = \{(x, y) \in \mathbb{R}^2 \mid h_\lambda(x, y) = h_\lambda(q)\}$. Then there exists no solution point (i.e., no point in $\mathcal{I}$) below the line $(1 + \delta)^{-1} \cdot \ell(q, \lambda) \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{R}^2 \mid h_\lambda(x, y) = h_\lambda(q)/(1 + \delta)\}$. Geometrically, this means that we sweep a line of absolute slope λ, until it touches (exactly or approximately) the undominated boundary (lower envelope) of the objective space $\mathcal{I}$. For $\delta = 0$, the routine returns a point on the lower envelope $\mathrm{LE}(\mathcal{I})$, while for $\delta > 0$ it returns a (potentially) dominated point of $\mathcal{I}$ "close" to the boundary (where the notion of "closeness" is quantitatively defined by the aforementioned condition). See Figure 1 for an illustration.

If q is the (feasible) point in $\mathcal{I}$ returned by $\mathrm{Comb}_\delta(\lambda)$, then we write $q = \mathrm{Comb}_\delta(\lambda)$. We assume that either $\delta = 0$ (i.e., we have an exact routine), or we have a PTAS. For $\delta = 0$, i.e., when the optimization is exact, we omit the subscript and denote the Comb routine simply by $\mathrm{Comb}(\lambda)$.

We will denote by $\mathrm{OPT}_\epsilon(\mathcal{I})$ the size of an optimum ϵ-convex Pareto set for the given instance, i.e., an ϵ-convex Pareto set with the minimum number of points. Note that, obviously every algorithm that constructs an ϵ-CP, must certainly make at the very least $\mathrm{OPT}_\epsilon(\mathcal{I})$ calls to Comb, just to get $\mathrm{OPT}_\epsilon(\mathcal{I})$ points – which are needed at a minimum to form an ϵ-CP; this holds even if the algorithm somehow manages to always be lucky and call Comb with the right values of λ that identify the points of an optimal ϵ-CP. Having obtained the points of an optimal ϵ-CP, another $\mathrm{OPT}_\epsilon(\mathcal{I})$ many calls to Comb with the slopes of the edges of the polygonal line defined by the points, suffice to verify that the points form an ϵ-CP. Hence, the "offline" optimum number of calls is at most $2 \cdot \mathrm{OPT}_\epsilon(\mathcal{I})$.

Let $\mathrm{CHORD}_\epsilon(\mathcal{I})$ be the number of Comb calls required by the Chord algorithm on instance $\mathcal{I}$. The *worst-case performance ratio* of the algorithm is defined to be $\sup_{\mathcal{I}} \frac{\mathrm{CHORD}_\epsilon(\mathcal{I})}{\mathrm{OPT}_\epsilon(\mathcal{I})}$. If the inputs are drawn from some probability distribution $\mathcal{D}$, then we will use the *expected performance ratio* $\mathbb{E}_{\mathcal{I} \sim \mathcal{D}}\left[\frac{\mathrm{CHORD}_\epsilon(\mathcal{I})}{\mathrm{OPT}_\epsilon(\mathcal{I})}\right]$ as a measure (note that we shall omit the subscript "$\mathcal{I} \sim \mathcal{D}$" when the underlying distribution over instances will be clear from the context).

While the main focus of this paper is on approximation of multiobjective optimization problems, our analysis also applies (with minor modifications) to related settings (in which the the Chord algorithm has been used). Consider for example the following classical *curve simplification* problem: Given a convex curve C of length (at most L) on the plane, find the minimum number of points on C

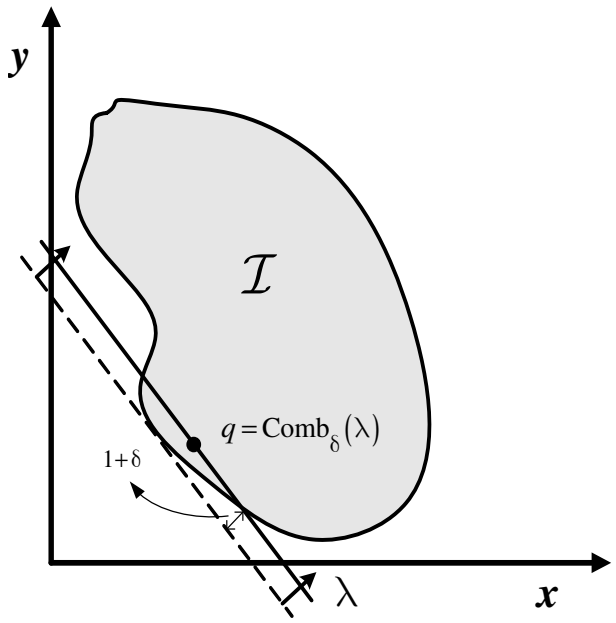

Figure 1: A geometric interpretation of the $\mathrm{Comb}_\delta(\lambda)$ routine. The shaded region represents the objective space $\mathcal{I}$. The routine guarantees that there exist no solution points strictly below the dotted line.

so that the corresponding convex polygonal chain C_ϵ approximates the curve C within distance error ϵ. A popular distance measure in this setting is the *Hausdorff distance* of C_ϵ from C, i.e., the maximum euclidean distance of a point in the actual curve from the approximating curve. Note that the Hausdorff distance is invariant under translation, while our ratio distance is invariant under scaling.

We remark that our upper and lower bounds for the performance of the Chord algorithm wrt the ratio distance apply with minor modifications for the Hausdorff distance. This can be seen as follows: A curve of length L located anywhere on the plane can be scaled down by L and translated to the unit square $[1, 2]$. By definition, approximating the original curve within Hausdorff distance ϵ is equivalent to approximating the new curve within Hausdorff distance error ϵ/L. A simple calculation shows that for a convex curve in the unit square $[1, 2]$ the two metrics (Hausdorff and ratio distance) are within a constant factor of each other. Hence, the upper and lower bounds on the approximation wrt ratio distance give immediately corresponding bounds wrt Hausdorff.

We also define the horizontal distance. We use this distance as an intermediate tool for our lower bound construction in Section 3.1. The *horizontal distance* from a point p to a point q is defined $\Delta x(p, q) = \max\{x(q) - x(p), 0\}$. The horizontal distance from p to a line ℓ (that is not horizontal) is $\Delta x(p, \ell) = \Delta x(p, p_\ell)$, where p_ℓ is the y-projection of p on ℓ (i.e., the point that a horizontal line from p intersects ℓ).

Remark 2.1. All the upper bounds of this paper on the performance of the Chord algorithm hold under the assumption that we have a PTAS for the Comb problem. On the other hand, our lower bounds apply even for the special case that an exact routine is available. For the clarity of the exposition, we describe the Chord algorithm and prove our upper bounds for the case of an exact Comb routine. We then describe the simple modifications in the algorithm and analysis for the case of an approximate routine.

2.2 The Chord Algorithm

We have set the stage to formally describe the algorithm. Let Π be a bi-objective problem with an efficient exact Comb routine. Given $\epsilon > 0$ and an instance $\mathcal{I}$ of Π (implicitly via Comb), we would like to construct an ϵ-CP for $\mathcal{I}$ using as few calls to Comb as possible. As mentioned in the introduction, a popular algorithm for this purpose is the Chord algorithm that is the main object of study in this paper. In Table 1 below we describe the algorithm in detailed pseudo-code. The pseudo-code corresponds exactly to the description of the algorithm in the introduction.

The (recursively defined) routine Chord is called from the main algorithm and returns a set of points $Q \subseteq \mathcal{I}$ that is an ϵ-CP for $\mathcal{I}$. The recursive description of the algorithm is quite natural and will be useful in the analysis.

Chord Algorithm (*Input:* $\mathcal{I}$, ϵ)

$a = \mathrm{Comb}(+\infty)$;

$b = \mathrm{Comb}(0)$;

$c = (x(a), y(b))$;

Return $Q =$ Chord $(\{a, b, c\}, \epsilon)$.

Routine Chord (*Input:* $\{l, r, s\}, \epsilon$)

If $\mathcal{RD}(s, lr) \leq \epsilon$ **return** $\{l, r\}$;

$\lambda_{lr} =$ absolute slope of lr; $q = \mathrm{Comb}(\lambda_{lr})$;

If $\mathcal{RD}(q, lr) \leq \epsilon$ **return** $\{l, r\}$;

$\ell(q) :=$ line parallel to lr through q;

$s_l = ls \cap \ell(q)$; $s_r = rs \cap \ell(q)$;

$Q_l =$ Chord$(\{l, q, s_l\}, \epsilon)$; $Q_r =$ Chord$(\{q, r, s_r\}, \epsilon)$;

Return $Q_l \cup Q_r$.

Table 1: Pseudo-code for Chord algorithm.

Let us proceed to explain the notation used in the pseudo-code in tandem with some intuitive understanding of the algorithm. First, the feasible points $a, b \in \mathcal{I}$ minimize the x-objective and y-objective respectively. (Note that these points may be dominated, i.e., are not necessarily the extreme points of $\mathrm{CP}(\mathcal{I})$; however, this does not affect our analysis.) By monotonicity and convexity, the lower envelope is contained in the right triangle $\triangle(acb)$, i.e., $\mathrm{LE}(\mathcal{I}) \subseteq \triangle(acb)$. (Note that the point c is not a feasible point, but is solely defined for the purpose of "sandwiching" the lower envelope.)

The Chord routine takes as input (i) The desired error tolerance ϵ, and (ii) The ordered 3-set of points $\{l, r, s\}$. In every recursive call of the Chord routine, the points l (left) and r (right) are (feasible) points of the lower envelope, i.e., $l, r \in \mathcal{I} \cap \mathrm{LE}(\mathcal{I})$. Moreover, the point s is a (not necessarily feasible) point and the following conditions are satisfied:

- The point s lies to the right of l, to the left of r and below the line segment lr. In particular, this implies that the triangle $\triangle(lsr)$ is either right or obtuse, i.e., $\angle(lsr) \in [\pi/2, \pi)$.
- The subset of the lower envelope (convex curve) with endpoints l and r is contained in $\triangle(lsr)$, i.e., $\mathrm{LE}(lr) \subseteq \triangle(lsr)$.

See Figure 2 for an illustration. The red curve represents the lower envelope between the points l and r, i.e., the unknown curve we want to approximate. (Note that the feasible points l, r are the results of previous recursive calls.) The point $q = \mathrm{Comb}(\lambda_{lr})$ is the feasible point in $\mathrm{LE}(\mathcal{I})$ computed in the current iteration (recursive call). We remark that this point is at maximum ratio distance from the "chord" lr – among all points of $\mathrm{LE}(lr)$. The Chord routine will recurse on the triangles $\triangle(ls_lq)$ and $\triangle(qs_rr)$. Note that, by construction, the line s_ls_r is parallel to lr.

During the execution of the algorithm, we "learn" the objective space in an "online" fashion. After a number of iterations, we have obtained information that imposes an "upper" and a "lower" approximation to $\mathrm{LE}(\mathcal{I})$. In particular, the computed solution points define a polygonal chain that is an upper approximation to $\mathrm{LE}(\mathcal{I})$ and the supporting lines at these points define a lower approximation. As the number of iterations increases, these bounds become more and more refined, hence we obtain a better approximation to the curve.

Consider for example Figure 2. Before the current iteration of the Chord routine, the only information available to the algorithm is that the lower envelope (between points l and r) lies in $\triangle(lsr)$, i.e., lr is an upper approximation and the polygonal chain $\langle l, s, r\rangle$ is a lower approximation. Given the information the algorithm has at this stage, the potential error of this initial approximation is the ratio distance $\mathcal{RD}(s, lr)$. (If this distance is at most ϵ, then the segment lr ϵ-covers the subset of the lower envelope between l, r and the routine terminates. Otherwise, the point q is computed and the approximation is refined, if necessary, as explained above.) After the current iteration, the upper approximation is refined to be $\langle l, q, r\rangle$ and the lower approximation is $\langle l, s_l, s_r, r\rangle$. The potential error given the available information now is $\max\{\mathcal{RD}(s_l, lq), \mathcal{RD}(s_r, qr)\} < \mathcal{RD}(s, lr)$. By applying these arguments recursively, we get that the Chord algorithm always terminates and, upon termination, it outputs an ϵ-CP for $\mathcal{I}$ (see Lemma 3.10 for a rigorous proof).

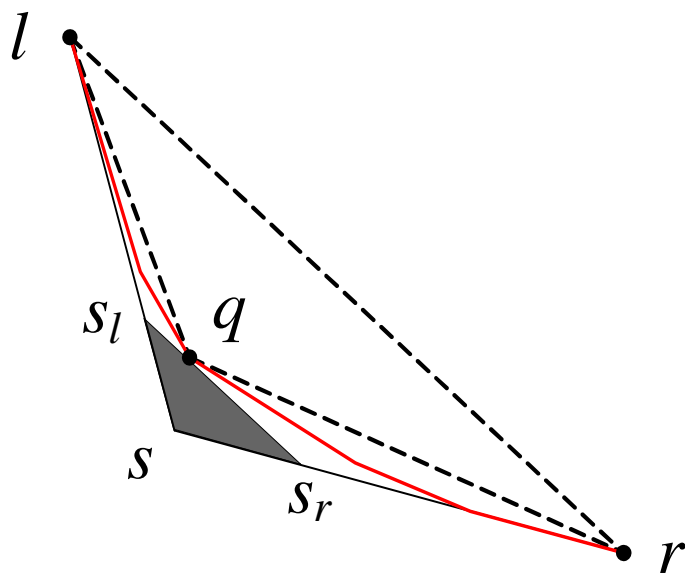

Figure 2: Illustration of one iteration of the Chord routine.

Consider the recursion tree built by the Chord algorithm. Every node of the tree corresponds to a triangle $\triangle(lsr)$ (input to the Chord routine at the corresponding recursive call). In the analysis, we shall use the following convention: There is no node in the recursion tree if, at the corresponding step, the routine terminates without calling Comb (i.e., if $\mathcal{RD}(s, lr) \leq \epsilon$ in the aforementioned description).

The pseudo-code of Table 1 is specialized for the ratio distance, but one may use other metrics based on the application. In the context of convex curve simplification, our upper and lower bounds

for the Chord algorithm also apply for the Hausdorff distance (i.e. the maximum euclidean distance of a point in the actual curve from the approximate curve).

2.3 Our Results

We are now ready to state our main results. Our first main result is an analysis of the Chord algorithm on worst-case instances that is tight up to constant factors. In particular, for the ratio distance we prove

Theorem 2.2. *The worst-case performance ratio of the Chord algorithm (wrt the ratio distance) is* $\Theta\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$.

The lower bound on the performance of the Chord algorithm is proved in Section 3.1. In Section 3.1.2, we also prove a general lower bound of $\Omega(\log m + \log\log(1/\epsilon))$ on the performance ratio of *any* algorithm in the Comb based model, as well as lower bounds for approximating with respect to the horizontal (resp. vertical) distance.

In Section 3.2 we give a proof of the upper bound. In Section 3.2.1 we start by presenting the slightly weaker upper bound of $O(m+\log(1/\epsilon))$; this result has the advantage that its proof is simple and intuitive. The proof of the asymptotically tight upper bound requires a more careful analysis and is presented in Section 3.2.2.

Remark 2.3. It turns out that the Hausdorff distance behaves very similarly to the ratio distance. In particular, by essentially identical proofs, it follows that the performance ratio of the Chord algorithm for approximating a convex curve of length L within Hausdorff distance ϵ, is $\Theta(\frac{\log(L/\epsilon)}{\log\log(L/\epsilon)})$. Furthermore, every algorithm has worst-case performance ratio at least $\Omega(\log\log(L/\epsilon))$.

In the process, we also analyze the Chord algorithm with respect to the horizontal distance metric (or by symmetry the vertical distance). We show that in this setting the performance ratio of the algorithm is unbounded. In fact, we can get a strong lower bound in this case: *Any* algorithm with oracle access to Comb has an unbounded performance ratio (Theorem 3.7), even on instances that lie in the unit square and the desired accuracy is a constant.

Our second main result is an asymptotically tight analysis of the Chord algorithm in an average case setting (wrt the ratio distance). Our random instances are drawn from standard distributions that have been widely used for the average case analysis of geometric algorithms in a variety of settings. In particular, we consider (i) a Poisson Point Process on the plane and (ii) n points drawn from an "un-concentrated" product distribution. We now formally define these distributions.

Definition 2.4. A (spatial, homogeneous) *Poisson Point Process (PPP)* of intensity ν on a bounded subset $\mathcal{S} \subseteq \mathbb{R}^2$ is a collection of random variables $\{N(A) \mid A \subseteq \mathcal{S} \text{ is Lebesgue measurable}\}$ (representing the number of points occurring in every subset of $\mathcal{S}$), such that: (i) for any Lebesgue measurable A, $N(A)$ is a Poisson random variable with parameter $\nu \cdot S(A)$; (ii) for any collection of *disjoint* subsets $A_1, \ldots, A_k$ the random variables $\{N(A_i), i \in [k]\}$ are mutually independent.

Definition 2.5. Let A be a bounded Lebesgue-measurable subset of $\mathbb{R}^2$, and let $\mathcal{D}$ be a distribution over A. The distribution $\mathcal{D}$ is called γ-*balanced*, $\gamma \in [0,1)$, if for all Lebesgue measurable subsets $A' \subseteq A$, $\mathcal{D}(A') \in \left[(1-\gamma)\cdot\mathcal{U}(A'), \frac{\mathcal{U}(A')}{(1-\gamma)}\right]$, where $\mathcal{U}$ is the uniform distribution over A.

We assume that γ is an absolute constant and we omit the dependence on γ in the performance ratio below. We prove:

Theorem 2.6. *For the aforementioned classes of random instances, the expected performance ratio of the Chord algorithm (wrt to the ratio distance) is* $\Theta\big(\log m + \log\log(1/\epsilon)\big)$.

The upper bound proof is given in Section 4.1 and the lower bound one in Section 4.2. We first present detailed proofs for the case of PPP and then present the (more involved) case of product distributions. (We note that similar results apply also for approximation under the Hausdorff distance.)

3 Worst–Case Analysis

3.1 Lower Bounds

In Section 3.1.1 we prove a tight lower bound for the Chord algorithm for the ratio distance metric. In Section 3.1.2 we show our general lower bounds, both for the ratio distance and the horizontal distance.

3.1.1 Lower Bound for Chord Algorithm

Our main result in this section is a proof of the lower bound statement in Theorem 2.2. In fact, we show a stronger statement that also rules out the possibility of constant factor bi-criteria approximations, i.e., it applies even if the Chord algorithm is allowed (ratio distance) error $\Omega(\epsilon)$ and we compare it against the optimal ϵ-CP set.

Theorem 3.1. *Let* $\mu \geq 1$ *be an absolute constant. Let* $\epsilon > 0$ *be smaller than a sufficiently small constant and* $m > 0$ *be large enough. There exists an instance* $\mathcal{I}_{LB} = \mathcal{I}_{LB}(\epsilon, m, \mu)$ *such that* $\mathrm{OPT}_\epsilon(\mathcal{I}_{LB}) \leq 3$ *and* $\mathrm{CHORD}_{\mu\cdot\epsilon}(\mathcal{I}_{LB}) = \Omega\Big((1/\mu)\cdot\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\Big)$.

Proof. The lower bound applies even if an exact Comb routine is available, hence we restrict ourselves to this case. Before we proceed with the formal proof, we give an explanation of our construction for the case $\mu = 1$ and $m = 1$. The rough intuition is that the algorithm can perform poorly when the input instance is "skewed", i.e., we have a triangle $\triangle(abc)$ where $(ac) >> (bc)$. For such instances one can force the algorithm to select many "redundant" points (hence, perform many calls to Comb) to obtain a certificate it has found an ϵ-CP set, even when few points (calls) suffice.

For the special case under consideration, the "hard" instance has endpoints $a = (1, 2)$ and $b = (1 + 2\epsilon, 1)$, where ϵ is sufficiently small (to be specified later). Initially, the only available information to the algorithm is that the convex Pareto set for the given instance lies in the right triangle $\triangle(acb)$, where $c = (1, 1)$. Observe that, for an instance with these endpoints, the initial error $\mathcal{RD}(c, ab)$ is roughly equal to 2ϵ and one intermediate point q^* together with the rightmost point of the curve always suffice to form an ϵ-CP set, i.e., the optimal size is no more than 2. Our construction will define a sequence of points $\{q_1, \ldots, q_j\}$ (ordered in increasing x-coordinate) which will form the Convex Pareto set for the corresponding instance and that force the Chord algorithm to select *all* the q_i's (in order of increasing i), until it finds $q^* = q_j$. That is, the Chord algorithm will monotonically converge to the optimal point by visiting all the vertices of the instance in order (in increasing x-coordinate).

Let $\lambda_{ab} = 1/(2\epsilon)$ be the slope of ab. The algorithm starts by calling $\mathrm{Comb}(\lambda_{ab})$ to find a solution point at maximum ratio distance from the chord ab. Our construction guarantees that $q_1 = \mathrm{Comb}(\lambda_{ab})$. That is, if $\ell(q_1)$ is the line parallel to ab through q_1, $\ell(q_1)$ supports the objective space. The point q_1 is selected on the line segment ac so that $(q_1c) = (ac)/k$, i.e., it is obtained by subdividing (the length of) ac geometrically with ratio k – for an appropriate k (to be specified next).

Consider the point $q_1^* = \ell(q_1) \cap bc$. The error of the approximation $\{a, q_1, b\}$ equals $\mathcal{RD}(q_1^*, q_1b)$ (note that the error to the left of q_1 is 0). If $k \leq 2$, we are already done, since $x(q_1^*) \geq 1 + \epsilon$, which

implies $\mathcal{RD}(q_1^*, q_1 b) \leq \epsilon$. On the other hand, if $k \geq 1/\epsilon$, we are also done since $y(q_1) \leq 1 + \epsilon$, hence $\mathcal{RD}(q_1^*, q_1 b) \leq \epsilon$. If $\omega(1) \leq k \leq o(1/\epsilon)$, it is not hard to show that

$$\mathcal{RD}(q_1^*, q_1 b) \approx \Delta x(q_1^*, q_1 b) = (q_1^* b) = 2\epsilon \cdot (1 - 1/k)$$

Hence, after the first call to Comb, the error has decreased by an additive of $2\epsilon/k << \epsilon$ and the algorithm will recurse on the triangle $\triangle(q_1 q_1^* b)$.

Note that $\lambda_{q_1 b} = \lambda_{ab}/k = (2\epsilon)^{-1}/k$. The algorithm proceeds by calling $\mathrm{Comb}(\lambda_{q_1 b})$ and this call will return the point q_2. Let $\ell(q_2)$ be the (supporting) line parallel to $q_1 b$ through q_2. Similarly, $\ell(q_2)$ supports the objective space. The point q_2 is selected by repeating our "geometric subdivision trick." Recall that there are no feasible points below the line $q_1 q_1^*$. Let q_2' be the projection of q_2 on ac. We select q_2 on the segment $q_1 q_1^*$ so that $(q_2' c) = (q_1 c)/k$. The error of the approximation $\{a, q_1, q_2, b\}$ equals $\mathcal{RD}(q_2^*, q_2 b)$, where $q_2^* = \ell(q_2) \cap bc$. If $(q_2' c) = (ac)/k^2 >> \epsilon$, we have that

$$\mathcal{RD}(q_2^*, q_2 b) \approx \Delta x(q_2^*, q_2 b) = (q_2^* b) \approx 2\epsilon \cdot (1 - 2/k),$$

i.e., after the second step of the algorithm, the error has decreased by another additive $2\epsilon/k$ and the algorithm will recurse on the triangle $\triangle(q_2 q_2^* b)$.

We can repeat this process iteratively, where (roughly) in step i we select q_i on the line $q_{i-1} q_{i-1}^*$, so that the length of the projection satisfies $(q_i' c) = (q_{i-1}' c)/k$. This iterative process can continue as long as $(q_i' c) >> \epsilon$. Also note that the number j of iterations cannot be more than $\approx k/2$ because $x(q_i) \approx 1 + i \cdot (2\epsilon/k)$ and $x(q^*) \leq 1 + \epsilon$. Since, $(q_i' c) = 1/k^i$ it turns out that the optimal choice of parameters is $2j \approx k \approx \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}$.

We stress that the actual construction is more elaborate than the one presented in the intuitive explanation above. Also, to show a bi-criterion lower bound, we need to add one more point q_{j+1} so that the Chord algorithm selects $\{q_1, \ldots, q_j\}$ until it forms a $(\mu \cdot \epsilon)$-CP, while the point q_{j+1} (along with the rightmost point b) suffice to form an ϵ-CP.

The formal proof comes in two steps. We first analyze the Chord algorithm with respect to the horizontal distance metric. We show that the performance ratio of the algorithm is unbounded in this setting (this statement also holds for the vertical distance by symmetry). In particular, for every $k \in \mathbb{N}$, there exists an instance $\mathcal{I}_G$ (lying entirely in the unit square) so that the Chord algorithm (applied for additive error $1/2$) has performance ratio k . We then show that, for an appropriate setting of the parameters in $\mathcal{I}_G$, we obtain the instance $\mathcal{I}_{LB}$ that yields the desired lower bound with respect to the ratio distance.

Step 1: The instance $\mathcal{I}_G(H, L, k, j)$ lies in the triangle $\triangle(abc)$, where $a = (1, 1 + H)$, $b = (1 + L, 1)$ and $c = (1, 1)$. The points a and b are (the extreme) vertices of the convex Pareto set. We introduce two additional parameters. The first one, $k \in \mathbb{N}$, is the ratio used in the construction to geometrically subdivide the length of the line ac in every iteration. The second one, $j \in \mathbb{N}$ with $j \in [1, k-1]$, is the number of iterations and equals the number of vertices in the instance.

We define a set of points $Q = \{q_i\}_{i=0}^{j+2}$ ordered in increasing x–coordinate and decreasing y–coordinate. Our instance will be the convex polygonal line with vertices the points of Q. We set $q_0 = a$ and $q_{j+2} = b$. The set of points $\{q_1, \ldots, q_{j+1}\}$ is defined recursively as follows:

1. The point q_1 has $x(q_1) = x(a)$ and $y(q_1) = y(c) + \big(y(a) - y(c)\big)/k$.
2. For $i \in [2, j+1]$ the point q_i is defined as follows: Let $\ell(q_{i-1})$ denote the line parallel to $q_{i-2} b$ through q_{i-1}. The point q_i is the point of this line with $y(q_i) = y(c) + \big(y(q_{i-1}) - y(c)\big)/(k+i-1)$.

The algorithm is applied on this instance with desired horizontal distance error

$$\epsilon_L(L,k,j) \stackrel{\text{def}}{=} L \cdot \frac{k-1}{k+j-1}.$$

Also denote

$$\epsilon'_L(L,k,j) \stackrel{\text{def}}{=} L \cdot \frac{k-1}{k} \cdot \frac{j}{k+j-1} = (j/k) \cdot \epsilon_L.$$

Note that $\epsilon'_L < \epsilon_L$.

See Figures 3 and 4 for a graphic illustration of the worst-case instances for the Chord algorithm. We would like to stress that the figures are not drawn to scale. In particular, in the figures below we have $H = L$, while the actual lower bound for the ratio distance applies for $H >> L$; in particular, for $H = 2^m$ and $L = O(\epsilon)$.

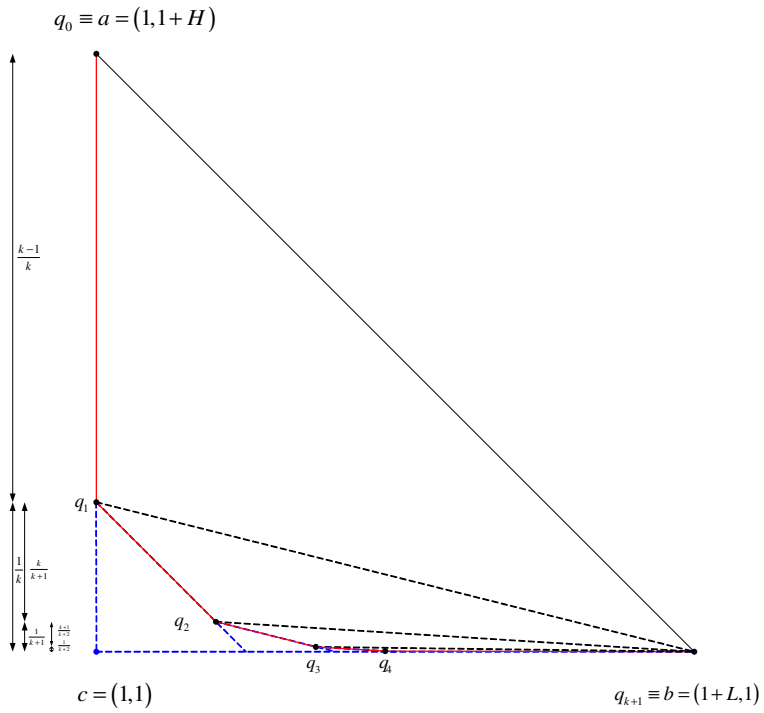

Figure 3: Illustration of the lower bound. The figure depicts the case $j = k = 4$.

We show the following:

Lemma 3.2. *The Chord algorithm applied to the instance $\mathcal{I}_G$ and error bound ϵ_L wrt horizontal distance selects the sequence of points $\langle q_1, q_2, \ldots, q_j \rangle$, while the set $\{a, q_{j+1}, b\}$ attains error $\epsilon'_L < \epsilon_L$.*

Proof. For $i \in [j-1]$, let q_i^* be the intersection of the line $\ell(q_i)$ – the line parallel to $q_{i-1}b$ through q_i– with bc. The error of $\{a, q_{j+1}, b\}$ is exactly $\Delta x(q_1, aq_{j+1})$. Observe that $\Delta x(q_1, aq_{j+1}) < \Delta x(q_1, aq_j^*)$. By a simple geometric argument we obtain $\Delta x(q_1, aq_j^*) = \epsilon'_L$, which yields the second statement. For the first statement, we show inductively that the recursion tree built by the algorithm for $\mathcal{I}_G$ is a path of length $j-1$ and at depth $i-1$, for $i \in [j]$, the Chord subroutine selects point q_i. The proof amounts

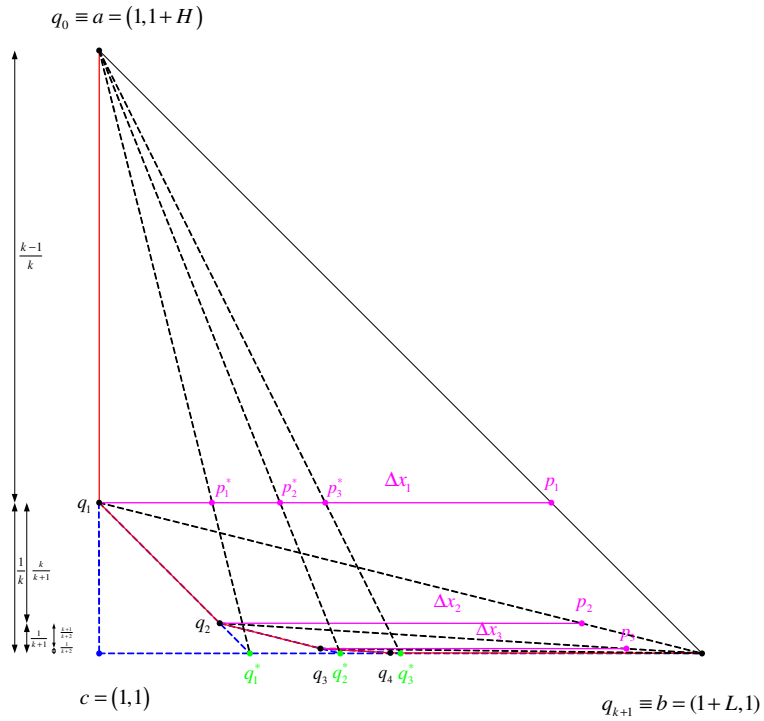

Figure 4: Illustration of definitions for counterexample in Figure 3.

to noting that the error of the approximation $\{q_1, \ldots, q_i\}$ is $\Delta x(q_i^*, q_i b)$, which is $> \epsilon_L$ for $i < j$ and $= \epsilon_L$ for $i = j$.

To provide the details we need some notation. For $i \in [j]$, we denote $\Delta x_i \stackrel{\text{def}}{=} \Delta x(q_i, q_{i-1}b)$. Let p_i be the y-projection of q_i on $q_{i-1}b$, so that $\Delta x_i = (q_i p_i)$. Recall that q_i^* denotes the intersection of the line $\ell(q_i)$ – the line parallel to $q_{i-1}b$ through q_i–with bc. If p_i^* is the y-projection of q_1 on aq_i^*, we have $\Delta x(q_1, aq_i^*) = (q_1 p_i^*)$. (See Figure 4 for an illustration of these definitions.) We start with the following claim:

Claim 3.3. *For all $i \in [j]$, we have $\Delta x_i = L \cdot (k-1)/(k+i-1)$.*

Proof. By induction on i. For the induction basis ($i = 1$), we observe that the triangles $\triangle(aq_1p_1)$ and $\triangle(acb)$ are similar, hence

$$\frac{\Delta x_1}{(bc)} = \frac{(aq_1)}{(ac)}$$

which yields

$$\Delta x_1 = (bc) \cdot (1 - 1/k) = L \cdot (k-1)/k$$

as desired.

Suppose that the claim is true for $i \in [j-1]$. We will prove it for $i+1$. We similarly exploit the similarity of the triangles $\triangle(q_i q_i^* b)$ and $\triangle(q_i q_{i+1} p_{i+1})$, from which we get

$$\frac{\Delta x_{i+1}}{(q_i^* b)} = \frac{(q_i q_{i+1})}{(q_i q_i^*)} = \frac{y(q_i) - y(q_{i+1})}{y(q_i)} \tag{1}$$

where the second equality follows from the collinearity of q_i, q_{i+1}, q_i^*. Observe that the third term in (1) is equal to $(k+i-1)/(k+i)$ by construction. Now note that, because of the parallelogram $(q_i p_i b q_i^*)$, we have $(q_i^* b) = \Delta x_i$. Hence, (1) and the induction hypothesis imply

$$\Delta x_{i+1} = \Delta x_i \cdot \frac{k+i-1}{k+i} = L \cdot \frac{k-1}{k+i-1} \cdot \frac{k+i-1}{k+i} = L \cdot \frac{k-1}{k+i}$$

which completes the proof of the claim. ■

Since $(q_i^* b) = \Delta x_i$ (as noted in the proof of Claim 3.3), it follows that for all $i \in [j-1]$ we have

$$x(q_i^*) = 1 + L \cdot \frac{i}{k+i-1}. \tag{2}$$

We will start by showing that the set $\{a, q_{j+1}, b\}$ is an ϵ'_L-convex Pareto under the horizontal distance. First, note that the error to the right of q_{j+1}, i.e., the distance of the lower envelope from $q_{j+1}b$ is in fact zero (since $q_{j+1}b$ is the rightmost edge of the lower envelope). It suffices to bound from above the error to its left. Since aq_{j+1} has absolute slope larger than ab, the unique point (of the lower envelope) at maximum distance from aq_{j+1} is q_1. Thus, we have that $\Delta x(q_1, aq_{j+1}) < \Delta x(q_1, aq_j^*)$. From the similarity of the triangles $\triangle(caq_j^*)$ and $\triangle(q_1 a p_j^*)$ we get

$$\Delta x(q_1, aq_j^*) = (1-1/k) \cdot (cq_j^*) = (1-1/k) \cdot (x(q_j^*) - x(c)) = L \cdot (1-1/k) \cdot \frac{j}{k+j-1} = \epsilon'_L.$$

Hence, $\Delta x(q_1, aq_{j+1}) < \epsilon'_L$ as desired.

We now proceed to analyze the behavior of the Chord algorithm. We will show that the algorithm selects the points $q_1, q_2, \ldots, q_j$ (in this order) till it terminates. Formally, we consider the recursion tree built by the algorithm for the instance $\mathcal{I}_G$ and prove that it is a path of length $j-1$. In particular, for all $i \in [j]$, at depth $i-1$, the Chord subroutine selects point q_i.

We prove the aforementioned statement by induction on the depth d of the tree. Recall that the Chord algorithm initially finds the extreme points a and b. For $d = 0$ (first recursive call), the algorithm selects a point of the lower envelope with maximum horizontal distance from ab. By construction, all the points (of the lower envelope) in the line segment $q_1 q_2$ have the same (maximum) distance from ab (since $q_1 q_2$ is parallel to ab). Hence, any of those points may be potentially selected. Since Comb is a black-box oracle, we may assume that indeed q_1 is selected[†]. The maximum error after q_1 is selected equals $\Delta x(q_1^*, q_1 b) = x(b) - x(q_1^*) = L \cdot (1-1/k) > \epsilon_L$. Hence, the algorithm will not terminate after it has selected q_1.

For the inductive step, we assume that the recursion tree is a path up to depth $d \in [j-2]$ and the algorithm selected the points $\{q_1, q_2, \ldots, q_{d+1}\}$ up to this depth. We analyze the algorithm at depth $d+1$. At depth $d+1$ the information available to the algorithm is that (i) the error to the left of q_{d+1} is 0 and (ii) the error to its right is $\Delta x(q_{d+1}^*, q_{d+1} b) = L \cdot (k-1)/(k+d) > \epsilon_L$ (since $d \leq j-2$). Hence, the algorithm does not terminate and it calls Comb to find a point between q_{d+1} and b at maximum distance from $q_{d+1}b$. By construction, the points of maximum distance are those belonging to the line $q_{d+2}q_{d+3}$ (which is parallel to $q_{d+1}b$); similarly, we can assume q_{d+2} is selected. At this point of the

[†]This simplifying assumption is only used for the sake of the exposition. We can slightly perturb the instance so that the absolute slope of $q_1 q_2$ is "slightly" smaller than λ_{ab}, so that the effect on the actual distances is negligible. By doing so, the point q_1 will be the "unique minimizer" for $\mathrm{Comb}(\lambda_{ab})$.

execution the algorithm has the information that the error to the left of q_{d+2} is 0 and the error to its right is $\Delta x(q^*_{d+2}, q_{d+2}b) = L \cdot (k-1)/(k+d+1) > \epsilon_L$, unless $d = j-2$. This completes the induction and the proof of Lemma 3.2. ■

Step 2: The instance $\mathcal{I}_{LB}$ is obtained from $\mathcal{I}_G(H, L, k, j)$ by appropriately setting the four relevant parameters. In particular,

1. Fix $H^* := 2^m - 1$, $L^* := (\mu+1)\cdot\epsilon$, $j^* := \Theta\big((1/\mu)\cdot\frac{\log(H^*/\epsilon)}{\log\log(H^*/\epsilon)}\big)$ and $k^* := \mu \cdot j^* + 1$.
2. Set $\mathcal{I}_{LB}(\epsilon, m, \mu) := \mathcal{I}_G(H^*, L^*, k^*, j^*)$.
3. Also, define ${\epsilon_L}^* := \epsilon_L(L^*, k^*, j^*)$ and ${\epsilon'_L}^* := \epsilon'_L(L^*, k^*, j^*)$.

Observe that, under this choice of parameters, we have ${\epsilon_L}^* \geq \mu\cdot\epsilon$ and ${\epsilon'_L}^* < \epsilon$. Our main lemma for this step is the following:

Lemma 3.4. *The Chord algorithm applied to $\mathcal{I}_{LB}$ and error bound ${\epsilon_L}^*$ wrt ratio distance selects (a superset of) the points $\{q_1, \ldots, q_{j^*/8}\}$, while the set $\{a, q_{j^*+1}, b\}$ forms an ${\epsilon'_L}^*$-convex Pareto set.*

Proof. The main idea of the proof is that for the particular instance under consideration, the horizontal distance metric is a very good approximation to the ratio distance. As a consequence, one can show that the behavior of the Chord algorithm in both metrics is similar. The reason we "lose" a constant factor in the number of points (i.e., the Chord algorithm selects $j^*/8$ points under the ratio distance as opposed to j^* under the horizontal distance) is due to the error term in the approximation between the metrics. (The factor of 8 is not important; any constant factor bigger than 1 would suffice.)

We now proceed with the details. Consider the set $Q = \{q_i\}_{i=0}^{j^*+2}$ defining the instance $\mathcal{I}_{LB}$. We will need the following lemma that quantifies the closeness of the two metrics in our setting.

Lemma 3.5. *Let $a = (1, 1+H)$, $b = (1+L, 1)$ and $c = (1,1)$. Consider a point s_1 in $\triangle(abc)$ and let λ be the absolute slope of s_1b. If c' is the x-projection of s_1 on bc, then for any point s_2 in $\triangle(s_1c'b)$ we have*

$$\mathcal{RD}(s_2, s_1b) < \Delta x(s_2, s_1b) \leq \mathcal{RD}(s_2, s_1b) + L^2 + L/\lambda. \tag{3}$$

Proof. The proof, though elementary, requires some careful calculations. Let $c' = (1+\delta, 1)$ be the x-projection of s_1 on the segment bc, that is $x(s_1) = 1+\delta$. Clearly, $0 \leq \delta \leq L$. If λ is the absolute slope of s_1b, we have that

$$y(s_1) = 1 + \lambda\cdot(L-\delta).$$

Fix a point $s_2 = (1+\delta_x, 1+\delta_y) \in \triangle(s_1c'b)$. We want to show that $\Delta x(s_2, s_1b)$, the horizontal distance of s_2 from s_1b, is a good approximation to the corresponding ratio distance $\mathcal{RD}(s_2, s_1b)$ when λ is large. (See Figure 5 for an illustration.)

We first calculate the horizontal distance $\Delta x(s_2, s_1b)$. Observe that

$$\Delta x(s_2, s_1b) = (s_2q) = (pq) - (ps_2)$$

where p is the y-projection of s_2 on s_1c', i.e., $x(p) = x(c')$ and $y(p) = y(s_2)$. It is clear that

$$(ps_2) = x(s_2) - x(p) = x(s_2) - x(c') = \delta_x - \delta.$$

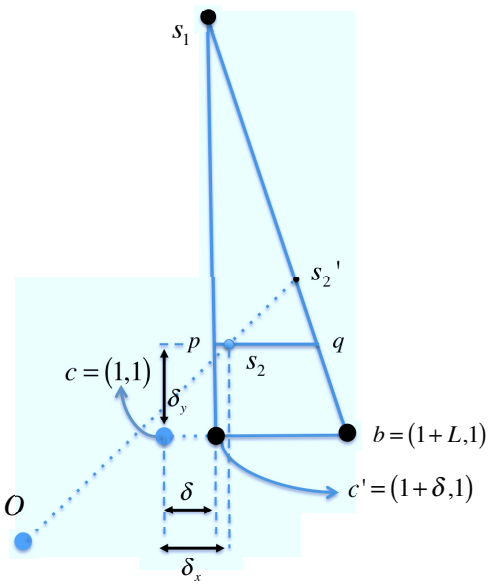

Figure 5: Illustration of the relation between the horizontal and the ratio distance.

From the similarity of the triangles $\triangle(s_1pq)$ and $\triangle(s_1c'b)$ it follows

$$\frac{(pq)}{(c'b)} = \frac{(s_1p)}{(s_1c')}.$$

Since $(c'b) = L - \delta$, $(s_1c') = \lambda \cdot (L - \delta)$ and $(s_1p) = (s_1c') - (pc') = (s_1c') - \delta_y$ we get

$$(pq) = (L - \delta) \cdot \Big(1 - \frac{\delta_y}{\lambda \cdot (L - \delta)}\Big).$$

Therefore,

$$\Delta x(s_2, s_1b) = (L - \delta) \cdot \Big(1 - \frac{\delta_y}{\lambda \cdot (L - \delta)}\Big) - (\delta_x - \delta) = L - \delta_x - \frac{\delta_y}{\lambda}.$$

The ratio distance $\mathbf{r} \stackrel{\text{def}}{=} \mathcal{RD}(s_2, s_1b)$ by definition is such that $s_2' = (1 + \mathbf{r}) \cdot s_2 \in s_1b$. We thus get

$$(1 + \mathbf{r}) \cdot y(s_2) + \lambda(1 + \mathbf{r}) \cdot x(s_2) = y(b) + \lambda x(b)$$

or equivalently

$$\mathbf{r} = \frac{\lambda\big(x(b) - x(s_2)\big) - \big(y(s_2) - y(b)\big)}{y(s_2) + \lambda x(s_2)}.$$

Substitution yields that

$$\mathbf{r} = \frac{\lambda(L - \delta_x) - \delta_y}{1 + \delta_y + \lambda(1 + \delta_x)} = \frac{L - \delta_x - \frac{\delta_y}{\lambda}}{1 + \delta_x + \frac{1+\delta_y}{\lambda}}.$$

Observe that the numerator of the above fraction equals $\Delta x(s_2, s_1 b)$ and the denominator is bigger than 1. Hence, $\mathcal{RD}(s_2, s_1 b) < \Delta x(s_2, s_1 b)$. For the other inequality we can write:

$$\begin{aligned}
\Delta x(s_2, s_1 b) - \mathcal{RD}(s_2, s_1 b) &= \left(L - \delta_x - \frac{\delta_y}{\lambda}\right) \cdot \left(1 - \frac{1}{1 + \delta_x + \frac{1+\delta_y}{\lambda}}\right) \\
&= \left(L - \delta_x - \frac{\delta_y}{\lambda}\right) \cdot \left(\frac{\delta_x + \frac{1}{\lambda} + \frac{\delta_y}{\lambda}}{1 + \delta_x + \frac{1}{\lambda} + \frac{\delta_y}{\lambda}}\right) \qquad (4) \\
&\leq L \cdot (L + 1/\lambda) = L^2 + L/\lambda \qquad (5)
\end{aligned}$$

as desired. To obtain (5), we bound each term in (4) separately. The first term is clearly at most L. As for the second term, first note that the denominator is at least 1. To bound the numerator from above, we claim that $\delta_x + \frac{\delta_y}{\lambda} \leq L$. To see this we use our assumption that s_2 lies in $\triangle(s_1 c' b)$. In particular, this implies that s_2 lies below (or on) the line segment $s_1 b$, i.e.,

$$y(s_2) + \lambda x(s_2) \leq y(b) + \lambda x(b)$$

which gives

$$\delta_x + \frac{\delta_y}{\lambda} \leq \delta \leq L$$

as desired. This completes the proof of Lemma 3.5. ■

We may now proceed with the proof of Lemma 3.4. By Lemma 3.2, the set $\{a, q_{j^*+1}, b\}$ attains horizontal distance error at most ${\epsilon'_L}^*$. By the first inequality of (3), the ratio distance error of $\{a, q_{j^*+1}, b\}$ is at most as big, hence the second statement of Lemma 3.4 follows.

By Lemma 3.2, the Chord algorithm under the horizontal distance metric selects all the points $\{q_1, \ldots, q_{j^*}\}$ in order until it guarantees an ϵ^*_L-approximation. In paricular, after the algorithm has selected the subset $\{q_1, \ldots, q_i\}$, for $i \in [j^*]$, the horizontal approximation error is

$$\Delta x(q^*_i, q_i b) = L^* \cdot (k^* - 1)/(k^* + i - 1). \qquad (6)$$

We remark that the error term $E(L, \lambda) = L^2 + L/\lambda$ in the RHS of (3) leads to the "constant factor loss", i.e., the fact that the Chord algorithm under the ratio distance picks $j^*/8$ (as opposed to j^*) points. (Also note that, since the ratio distance is a lower bound for the horizontal distance, the Chord algorithm under the former metric will select at most j^* points.)

Suppose we invoke the Chord algorithm with desired error of 0, i.e., we want to reconstruct the lower envelope exactly. Then the algorithm will select the points q_i in order of increasing i. It is also clear that the error of the approximation decreases monotonically with the number of calls to Comb. Hence, to complete the proof, It suffices to show that after the algorithm has selected $\{q_1, \ldots, q_{j^*/8}\}$, the ratio distance error will be bigger than ϵ^*_L. To do this, we appeal to Lemma 3.5.

The ratio distance error of the set $\{q_1, \ldots, q_{j^*/8}\}$ is $\mathcal{RD}(q^*_{j^*/8}, q_{j^*/8} b)$. An application of (the second inequality of) (3) for $s_1 = q_{j^*/8}$, $s_2 = q^*_{j^*/8}$ gives

$$\mathcal{RD}(q^*_{j^*/8}, q_{j^*/8} b) \geq \Delta x(q^*_{j^*/8}, q_{j^*/8} b) - E(L^*, \lambda_{q_{j^*/8} b}).$$

For the first term of the RHS, it follows from (6) by substitution that

$$\Delta x(q^*_{j^*/8}, q_{j^*/8} b) = L^* \cdot \frac{\mu}{\mu + 1/8}$$

and we similarly get $\epsilon_L^* = L^* \cdot \frac{\mu}{\mu+1}$. Hence,

$$\Delta x(q^*_{j^*/8}, q_{j^*/8}b) = \epsilon_L^* + \Gamma,$$

where $\Gamma = L^* \cdot \Omega(1/\mu)$.

Consider the error term $E(L^*, \lambda_{q_{j^*/8}b}) = (L^*)^2 + L^*/\lambda_{q_{j^*/8}b}$. We will show that $E(L^*, \lambda_{q_{j^*/8}b}) < \Gamma$ which concludes the proof. The first summand $(L^*)^2 = (\mu+1)^2 \cdot \epsilon^2$ is negligible compared to L^*/μ, as long as $\epsilon = o(1/\mu^2)$. To bound the second summand from above we need a lower bound on the slope $\lambda_{q_{j^*/8}b}$.

Recall (Equation (2) in Lemma 3.3) that $x(q_i^*) = 1 + L^* \cdot i/(k^* + i - 1)$. It is also not hard to verify that

$$\left|x(q_i) - x(q^*_{i-1})\right| = O\left(L^*/(k^*)^i\right).$$

Also recall that

$$y(q_i) = 1 + \frac{H^*}{\prod_{j=1}^{i}(k^* + j - 1)} > 1 + \frac{H^*}{(2k^*)^i}.$$

Hence,

$$\lambda_{q_i b} = \frac{y(q_i) - y(b)}{x(b) - x(q_i)} > \left(\frac{H^*}{L^*}\right) \cdot (2k^*)^{-i}.$$

It is straightforward to check that for the chosen values of the parameters, $\lambda_{q_{j^*/8}b} >> \mu$, hence the second summand will also be significantly smaller than L^*/μ. In conclusion, $\mathcal{RD}(q^*_{j^*/8-1}, q_{j^*/8-1}b) > \epsilon_L^*$ and Lemma 3.4 follows. ■

This also completes the proof of Theorem 3.1. ■

3.1.2 General Lower Bounds

We can show an information–theoretic lower bound against any algorithm that uses Comb as a black box with respect to the ratio distance metric. Even though the bound we obtain is exponentially weaker than that attained by the Chord algorithm, it rules out the possibility of a constant factor approximation in this model. In particular, we show:

Theorem 3.6. *Any algorithm (even randomized) with oracle access to a* Comb *routine, has performance ratio* $\Omega\left(\log m + \log\log(1/\epsilon)\right)$ *with respect to the ratio distance.*

Proof. Let $\mathcal{A}$ be a general algorithm with oracle access to Comb. The algorithm is given the desired error ϵ, and it wants to compute an ϵ-CP set. To do this, it queries the Comb routine on a sequence of (absolute) slopes $\{\lambda_i\}_{i=1}^k$ and terminates when it has obtained a "certificate" that the set of points $\cup_{i=1}^k \mathrm{Comb}(\lambda_i)$ forms an ϵ-CP set.

The queries to Comb can be adaptive, i.e., the i-th query λ_i of the algorithm $\mathcal{A}$ can depend on the information (about the input instance) the algorithm has obtained from the previous queries $\lambda_1, \ldots, \lambda_{i-1}$. On the other hand, the adversary also specifies the input instance adaptively, based on the queries made by the algorithm.

We will define a family of instances $\mathcal{Q}$ and an error $\epsilon > 0$ with the following properties:

(1) Each instance $I \in \mathcal{Q}$ has $\mathrm{OPT}_\epsilon(I) \leq 3$.

(2) In order for an algorithm $\mathcal{A}$ to have a certificate it found an ϵ-CP set for an (adversarially chosen) instance $I \in \mathcal{Q}$, it needs to make at least $\Omega\left(\log m + \log\log(1/\epsilon)\right)$ calls to Comb.

Our construction uses the lower bound example for the Chord algorithm from Section 3.1.1 essentially as a black box. Consider the instance $\mathcal{I}_{LB}(\epsilon, m, \mu := 1)$ (note that for $\mu = 1$ we have that $\epsilon_L^* = \epsilon$ and $\epsilon'^*_L < \epsilon$) and let $Q = \{q_i\}_{i=0}^{j+1}$ be the corresponding set of points, where $q_0 = a$ and $q_{j+1} = b$ and $j = j^*/8 = \Omega\left(\frac{m+\log(1/\epsilon)}{\log m + \log\log(1/\epsilon)}\right)$. Our family of input instances $\mathcal{Q}$ consists of all "prefixes" of Q, i.e., $\mathcal{Q} = \{I_i\}_{i=1}^{j}$, where $I_i = \{a \equiv q_0, q_1, \ldots, q_i, q_{j+1} \equiv b\}$, $i \in [j]$.

By the properties of Q, each I_i defines a convex monotone decreasing polygonal curve, i.e., all its points are vertices of the corresponding convex Pareto. Moreover, the set $\{a, q_i, b\}$ is an ϵ_i-CP set for I_i, where $\epsilon_i = \mathcal{RD}(q_1, aq_i)$. Note that $\epsilon_i < \epsilon_{i+1}$, $i \in [j]$, and that

$$\epsilon_{j+1} < \Delta x(q_1, aq_{j+1}) < \Delta x(q_1, aq_j^*) \leq \epsilon'^*_L < \epsilon$$

where the first inequality holds from (3), and the rest follow by construction. This proves property (1) above.

We now proceed to prove (2). We claim that, given an arbitrary (unknown) instance $I \in \mathcal{Q}$, in order for an algorithm $\mathcal{A}$ to have a certificate it discovered an ϵ-CP set, $\mathcal{A}$ must uniquely identify the instance I. In turn, this task requires $\Omega(\log|\mathcal{Q}|) = \Omega(\log j)$ Comb calls.

Indeed, consider an unknown instance $I \in \mathcal{Q}$ and let q_i be the rightmost point of I (excluding b) the algorithm $\mathcal{A}$ has discovered up to the current points of its execution. At this point, the information available to the algorithm is that $I \in \{I_\ell\}_{\ell \geq i}$. Hence, the error the algorithm can guarantee for its current approximation is

$$\mathcal{RD}(q_{i+1}, q_i b) \geq \Delta x(q_{i+1}, q_i b) - E(L^*, \lambda_{q_i b}) > \epsilon_L^* \geq \epsilon.$$

The first inequality above follows from Lemma 3.5. Also note that the term in the RHS is minimized for $i = j$ and (by the same analysis as in Lemma 3.4) the minimum value is bigger than ϵ_L^*.

It remains to show that an adversary can force any algorithm to make at least $\Omega(\log|\mathcal{Q}|)$ queries to Comb until it has identified an unknown instance of $\mathcal{Q}$. Clearly, identifying an unknown instance $I \in \mathcal{Q}$, i.e., finding the index ℓ such that $I = I_\ell$, is equivalent to identifying q_ℓ – the rightmost point of I to the left of b. First, we can assume the algorithm is given the extreme points a, b beforehand. A general algorithm $\mathcal{A}$ is allowed to query the Comb routine for any slope $\lambda \in [0, +\infty)$. Suppose that $I = I_\ell$. Then, for $\lambda \in \Lambda_i := [\lambda_{q_{i-1}q_i}, \lambda_{q_i q_{i+1}})$, the Comb routine returns: (i) q_i if $\ell \geq i$, (ii) q_ℓ if $\ell = i - 1$, and (iii) b if $\ell < i - 1$.[†]. That is, the information obtained from a query $\lambda \in \Lambda_i$ is whether $\ell \geq i$, $\ell = i - 1$ or $\ell < i - 1$. So, for our class of instances, a general deterministic algorithm $\mathcal{A}$ is equivalent to a ternary decision tree with the corresponding structure. The tree has $|\mathcal{Q}|$ many leaf nodes and there are at most 2^{d-1} leaves at depth d. Every internal node of the tree corresponds to a query to the Comb routine, hence the depth measures the worst-case number of queries made by the algorithm. It is straightforward that any such tree has depth $\Omega(\log|\mathcal{Q}|)$ and the theorem follows. (If we allow randomization, the algorithm is a randomized decision tree. The expected depth of any such tree remains $\Omega(\log|\mathcal{Q}|)$ which yields the theorem for randomized algorithms as well.) ■

We next show that, for the horizontal distance metric (or vertical distance by symmetry), any algorithm in our model has an unbounded performance ratio, even for instances that lie in the unit square and for desired error $1/2$.

[†]Strictly speaking, if $\ell \geq i$ and $\lambda = \lambda_{q_{i-1}q_i}$, the Comb routine can return any point of the edge $q_{i-1}q_i$. However, if $\mathrm{Comb}(\lambda_{q_{i-1}q_i})$ returns q_i it can only help the algorithm (for our class of instances). Hence, we can make this assumption for the purposes of our lower bound.

Theorem 3.7. *Any algorithm with oracle access to a* Comb *routine has an unbounded performance ratio with respect to the horizontal distance, even on instances that lie in the unit square and for approximation error* $1/2$*.*

Proof. Let $\mathcal{A}$ be an algorithm that, given the desired error ϵ, computes an ϵ-approximation with respect to the horizontal distance. Fix $k \in \mathbb{Z}_+$. We show that an adaptive adversary can force the algorithm to make $\Omega(k)$ queries to Comb even when $O(1)$ queries suffice. That is, the performance ratio of the algorithm is $\Omega(k)$; since k can be arbitrary the theorem follows.

Our family of (adversarially constructed) instances all lie in the unit square and are obtained by a modification of the lower bound instance $\mathcal{I}_{\mathcal{G}}(H := 1, L := 1, 2k, j := k)$ for Chord under the horizontal distance (see Step 1 in Section 3.1.1). Since the horizontal distance is invariant under translation, we can shift our instances so that the points $a = (0, 1)$ and $b = (1, 0)$ are the leftmost and rightmost points of the convex Pareto set. After this shift, the point c is identified with the origin.

Consider the initial triangle $\triangle(acb)$, where $a = (0, 1)$, $b = (1, 0)$ and $c = (0, 0)$, and let $\lambda_1 > 0$ be the first query made by the algorithm. Given the value of λ_1, the adversary adds a vertex q_1 to the instance so that $q_1 = \mathrm{Comb}(\lambda_1)$. The strategy of the adversary is the following: The point q_1 belongs to ac, i.e., $x(q_1) = 0$. If $\lambda_1 \geq \lambda_{ab} = 1$, then the point q_1 has $y(q_1) = 1/(2k)$. Otherwise, $y(q_1) = \lambda_1/(2k)$. Let $\ell(q_1)$ be the line with slope λ_1 through q_1 and q_1^* be its intersection with cb. The current information available to the algorithm is that the point q_1 is a feasible point and that there are no points below the line $\ell(q_1)$. Hence, the horizontal distance error it can certify is $\Delta x(q_1^*, q_1 b) = (q_1^* b) = 1 - (cq_1^*)$. On the other hand, if the true convex Pareto set was $\mathrm{CP}_1 = \{a, q_1, q_1^*, b\}$, the set $\{a, q_1^*, b\}$ would attain error $\Delta x(q_1, aq_1^*) < (cq_1^*)$. Note that $(cq_1^*) = y(q_1)/\lambda_1 \leq 1/(2k)$.

After its first query, the algorithm knows that the error to the left of q_1 is 0, hence it needs to focus on the triangle $\triangle(q_1 q_1^* b)$. Therefore, it is no loss of generality to assume that $\lambda_2 < \lambda_{q_1 q_1^*} = \lambda_1$. Given λ_2, the adversary adds a vertex q_2 to the instance so that $q_2 = \mathrm{Comb}(\lambda_2)$. Its strategy is to place q_2 on $q_1 q_1^*$ and to set $y(q_2) = \lambda_{q_1 b}/(2k)$ if $\lambda_2 \geq \lambda_{q_1 b}$ and $y(q_2) = \lambda_2/(2k)$ otherwise. Let $\ell(q_2)$ be the line with slope λ_2 through q_2 and q_2^* be its intersection with cb. The information available to the algorithm after its second query is that the point q_2 is a feasible point and that there are no points below the line $\ell(q_2)$. Hence, the horizontal distance error it can certify is $\Delta x(q_2^*, q_2 b) = (q_2^* b) = 1 - (cq_2^*)$. On the other hand, if the true convex Pareto set was $\mathrm{CP}_2 = \{a, q_1, q_2, q_2^*, b\}$, the set $\{a, q_2^*, b\}$ would attain error $\Delta x(q_1, aq_2^*) < (cq_2^*)$. Similarly, $(cq_2^*) = (cq_1^*) + (q_1^* q_2^*) \leq 1/(2k) + y(q_2)/\lambda_2 \leq 2/(2k)$. The adversary argument continues by induction on i. The induction hypothesis is the following: After its i-th query, the algorithm has computed a set of points $\{q_1, \ldots, q_i\}$, $q_j = \mathrm{Comb}(\lambda_j)$, $j \in [i]$, where λ_j is the j-th query and $\lambda_j > \lambda_{j+1}$. (The q_j's are vertices of the convex Pareto set of the corresponding instance ordered left to right). Let $\ell(q_i)$ be the line with slope λ_i through q_i and q_i^* be its intersection with cb. Then $(cq_i^*) = x(q_i^*) \leq i/(2k)$.

We now prove the induction step. We start by noting that the error to the left of q_i is zero, so the algorithm needs to focus on the triangle $\triangle(q_i q_i^* b)$; this implies that (without loss of generality) $\lambda_{i+1} < \lambda_{q_i q_i^*} = \lambda_i$. Given λ_{i+1}, the adversary adds a vertex q_{i+1} such that $q_{i+1} = \mathrm{Comb}(\lambda_{i+1})$. Similarly, the strategy of the adversary is to place q_{i+1} on $q_i q_i^*$ and to set $y(q_{i+1}) = \lambda_{q_i b}/(2k)$ if $\lambda_{i+1} \geq \lambda_{q_i b}$ and $y(q_{i+1}) = \lambda_{i+1}/(2k)$ otherwise. Let $\ell(q_{i+1})$ be the line with slope λ_{i+1} through q_{i+1} and q_i^* be its intersection with cb. The information available to the algorithm after its $(i+1)$-th query is that the point q_{i+1} is a feasible point and that there are no points below the line $\ell(q_{i+1})$. Hence, the horizontal distance error it can certify is $\Delta x(q_{i+1}^*, q_{i+1} b) = (q_{i+1}^* b) = 1 - (cq_{i+1}^*)$. On the other hand, if the true convex Pareto set was $\mathrm{CP}_{i+1} = \{a, q_1, q_2, \ldots, q_{i+1}, q_{i+1}^*, b\}$, the set $\{a, q_{i+1}^*, b\}$ would attain

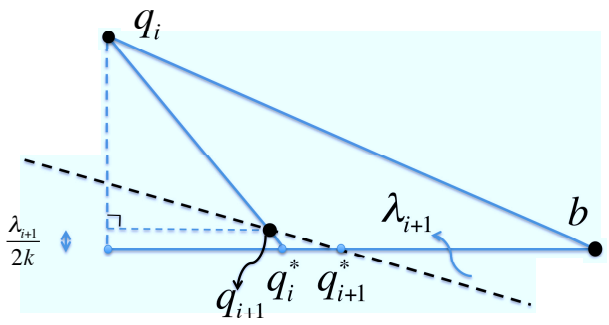

Figure 6: Illustration of general lower bound for horizontal distance.

error $\Delta x(q_1, aq^*_{i+1}) < (cq^*_{i+1})$. Now note that

$$(cq^*_{i+1}) = (cq^*_i) + (q^*_i q^*_{i+1}) \leq i/(2k) + y(q_{i+1})/\lambda_{i+1} \leq (i+1)/(2k)$$

where the second inequality uses the inductive hypothesis and the definition of q_{i+1}. (See Figure 6 for an illustration; the figure depicts the case that $\lambda_{i+1} < \lambda_{q_i b}$.) This completes the induction.

The overall adversary argument is obtained from the above construction for $i = k-1$. That is, the adversarially constructed instance is the set $\mathrm{CP}_{k-1} = \{a, q_1, q_2, \ldots, q_{k-1}, q^*_{k-1}, b\}$, where $q_i = \mathrm{Comb}(\lambda_i)$ (recall that λ_i is the i-th query). The set $\{a, q^*_{k-1}, b\}$ has error

$$\Delta x(q_1, aq^*_{k-1}) < (cq^*_{k-1}) \leq 1/2 - 1/(2k) < 1/2,$$

while the algorithm can certify error

$$\Delta x(q^*_{k-1}, q_{k-1}b) = (q^*_{k-1}b) = 1 - (cq^*_{k-1}) = 1/2 + 1/(2k) > 1/2.$$

Thus, after $k-1$ steps, the error of the algorithm remains more than $1/2$, while 3 queries suffice to attain error $< 1/2$. This completes the proof of Theorem 3.7. ■

3.2 Upper Bound

In this section we establish the upper bound statement of Theorem 2.2, i.e., we show that the performance ratio of the Chord algorithm for the ratio distance is $O\big(\frac{m+\log(1/\epsilon)}{\log m + \log\log(1/\epsilon)}\big)$. For the sake of the exposition, we start by showing the slightly weaker upper bound of $O(m + \log(1/\epsilon))$. The proof of the asymptotically tight upper bound is more involved and builds on the understanding obtained from the simpler argument presented first.

3.2.1 An $O(m + \log(1/\epsilon))$ Upper Bound

Before we proceed with the argument, some comments are in order. Perhaps the most natural approach to prove an upper bound would be to argue that the error of the approximation constructed by the

Chord algorithm decreases substantially (say by a constant factor) in every iteration (subdivision) or after an appropriately defined "epoch" of a few iterations. This, if true, would yield the desired result – since the initial error cannot be more than $2^{O(m)}$. Unfortunately, such an approach badly fails, as implied by the construction of Theorem 3.1. Recall that, in the simplest setting of that construction, the initial error is 2ϵ and decreases by an additive $2\epsilon/k$ in every iteration, where $k = \Omega\left(\log(1/\epsilon)/\log\log(1/\epsilon)\right)$. Hence, the error decreases by a sub-constant factor in every iteration. In fact, we note that such an argument cannot hold for *any* algorithm in our setting (i.e., given oracle access to Comb), as follows from our general lower bound (Theorem 3.6)[†].

Our approach is somewhat indirect. We prove that the *area* between the (implicitly constructed) "upper" and "lower" approximation decreases by a constant factor in every iteration of the algorithm (Lemma 3.11). This statement can be viewed as a "potential function type argument." To obtain an upper bound on the performance ratio, one additionally needs to relate the area to the ratio distance. Indeed, we show that, when the area (between the upper approximation and the lower approximation) has become "small enough" (roughly at most $\epsilon^2/2^{2m}$), the error of the approximation (ratio distance of the lower approximation from the upper approximation) is at most ϵ (Lemma 3.13). We combine the above with a simple charging argument (Lemma 3.12) to get the desired performance guarantee. Formally, we prove:

Theorem 3.8. *Let T_1 be the triangle at the root of the Chord algorithm's recursion tree, let $S(T_1)$ be its area, and denote $\alpha \stackrel{def}{=} \min\{x(q), y(q) \mid q \in T_1\}$. The algorithm finds an ϵ-CP set after $O\left(\log\left(S(T_1)/S_0\right)\right) \cdot \mathrm{OPT}_\epsilon$ calls to Comb, where $S_0 \stackrel{def}{=} \epsilon^2 \cdot \alpha^2$.*

The claimed upper bound on the performance ratio follows from the previous theorem, since $T_1 \subseteq [2^{-m}, 2^m]^2$, which implies $S(T_1) \leq 2^{2m}$ and $\alpha \geq 2^{-m}$.

To prove Theorem 3.8 we will need a few lemmas. We start by proving correctness, i.e., we show that, upon termination, the algorithm computes an ϵ-CP set for $\mathcal{I}$. This statement may be quite intuitive, but it requires a proof. The following claim formally states some basic properties of the algorithm:

Claim 3.9. *Let $T_i = \triangle(a_i b_i c_i)$ be the triangle processed by the Chord algorithm at some recursive step. Then the following conditions are satisfied: (i) $a_i, b_i \in \mathcal{I} \cap \mathrm{LE}(\mathcal{I})$; in particular, $a_i = \mathrm{Comb}(\lambda_{a_i c_i})$ and $b_i = \mathrm{Comb}(\lambda_{b_i c_i})$, (ii) $x(a_i) \leq x(c_i) \leq x(b_i)$, $y(a_i) \geq y(c_i) \geq y(b_i)$ and c_i lies below the line $a_i b_i$, and (iii) $\mathrm{LE}(a_i b_i) \subseteq T_i$.*

That is, whenever the Chord routine is called with parameters $\{l, r, s\}$ (see Table 1), the points l and r are feasible solutions of the lower envelope and the segment of the lower envelope between them is entirely contained in the triangle $\triangle(lsr)$.

Proof. Consider the node (corresponding to) T_i in the recursion tree built by the algorithm. We will prove the claim by induction on the depth of the node. The base case corresponds to either an empty tree or a single node (root). The Chord routine is initially called for the triangle $T_1 = \triangle(abc)$. Conditions (i) and (ii) are thus clearly satisfied. It follows from the definition of the Comb routine that

[†]In [RF] the authors – in essentially the same model as ours – propose a variant of the Chord algorithm (that appropriately subdivides the current triangle into three sub-problems). They claim (Lemma 3 in [RF]) that the error reduces by a factor of 2 in every such subdivision. However, their proof is incorrect. In fact, our counterexample from Section 3.1 implies the same lower bound for their proposed heuristic as for the Chord algorithm.

there exist no solution points strictly to the left of a and strictly below b. Hence, by monotonicity and convexity, we have $\mathrm{LE}(\mathcal{I}) = \mathrm{LE}(ab) \subseteq \triangle(acb)$, i.e., condition (iii) is also satisfied. This establishes the base case.

For the induction step, suppose the claim holds true for every node up to depth $d \geq 0$. We will prove it for any node of depth $d+1$. Indeed, let T be a depth $d+1$ node and let $T_i = \triangle(a_i b_i c_i)$ be T's parent node in the recursion tree. By the induction hypothesis, we have that (i) $a_i, b_i \in \mathcal{I} \cap \mathrm{LE}(\mathcal{I})$; $a_i = \mathrm{Comb}(\lambda_{a_i c_i})$ and $b_i = \mathrm{Comb}(\lambda_{b_i c_i})$ (ii) $x(a_i) \leq x(c_i) \leq x(b_i)$, $y(a_i) \geq y(c_i) \geq y(b_i)$ and c_i lies below the line $a_i b_i$ and (iii) $\mathrm{LE}(a_i b_i) \subseteq T_i$. We want to show the analogous properties for T.

We can assume without loss of generality that T is T_i's left child (the other case being symmetric). Then, it follows by construction (see Table 1) that $T = \triangle(a_i q_i a_i')$ where $q_i = \mathrm{Comb}(\lambda_{a_i b_i})$. We claim that $q_i \in T_i$. Indeed, note that $\lambda_{a_i c_i} > \lambda_{a_i b_i} > \lambda_{b_i c_i}$ (as follows from property (ii) of the induction hypothesis). By monotonicity and convexity of the lower envelope, combined with property (iii) of the induction hypothesis, the claim follows. Now note that $a_i' \in a_i c_i$ and $a_i' q_i \parallel a_i b_i$. Hence, property (i) of the inductive step is satisfied. Since $a_i c_i$ has non-positive slope (as follows from (ii) of the inductive hypothesis), a_i lies to the left and above a_i' ; similarly, since $a_i b_i$ has negative slope, a_i' lies to the left and above q_i. We also have that $\lambda_{a_i c_i} = \lambda_{a_i a_i'} \geq \lambda_{a_i q_i} \geq \lambda_{a_i' q_i} = \lambda_{a_i b_i}$, where the first inequality follows from the fact that $q_i \in T_i$. Property (ii) follows from the aforementioned. By definition of the Chord routine, we have $a_i' q_i \parallel a_i b_i$ and there are no solution points below $a_i' q_i$. By convexity, we get that $\mathrm{LE}(a_i q_i)$ lies below $a_i q_i$. Hence, property (iii) also follows. This proves the induction and the claim. ■

By exploiting the above claim, we can prove correctness:

Lemma 3.10. *The set of points Q computed by the Chord algorithm is an ϵ-CP set.*

Proof. Let $Q = \{a =: q_0, q_1, q_2, \ldots, q_r, q_{r+1} := b\}$ be the set of feasible points in $\mathrm{LE}(\mathcal{I})$ output by the algorithm, where the points of Q are ordered in increasing order of their x-coordinate (decreasing order of their y-coordinate). Note that all the q_i's are in convex position. We have that $\mathcal{RD}(\mathrm{LE}(\mathcal{I}), \langle q_0, \ldots, q_{r+1} \rangle) = \max_{i=0}^{r} \mathcal{RD}(\mathrm{LE}(q_i q_{i+1}), q_i q_{i+1})$. So, it suffices to show that, for all i, $\mathcal{RD}(\mathrm{LE}(q_i q_{i+1}), q_i q_{i+1}) \leq \epsilon$.

Since the algorithm terminates with the set Q, it follows that, for all i, the Chord routine was called by the algorithm for the adjacent feasible points $\{q_i, q_{i+1}\}$ and returned without adding a new feasible point between them. Let c_i be the corresponding third point (argument to the Chord routine). Then, by Claim 3.9, we have that c_i is between q_i and q_{i+1} in both coordinates and below the segment $q_i q_{i+1}$ and moreover $\mathrm{LE}(q_i q_{i+1}) \subseteq \triangle(q_i q_{i+1} c_i)$. Since the Chord routine returns without adding a new point on input $(\{q_i, q_{i+1}, c_i\}, \epsilon)$, it follows that either $\mathcal{RD}(c_i, q_i q_{i+1}) \leq \epsilon$ or the point $q' = \mathrm{Comb}(\lambda_{q_i q_{i+1}})$ satisfies $\mathcal{RD}(q', q_i q_{i+1}) \leq \epsilon$. In the former case, since $\mathrm{LE}(q_i q_{i+1}) \subseteq \triangle(q_i q_{i+1} c_i)$, we obtain $\mathcal{RD}(\mathrm{LE}(q_i q_{i+1}), q_i q_{i+1}) \leq \mathcal{RD}(c_i, q_i q_{i+1}) \leq \epsilon$ as desired. In the latter case, we have $\mathcal{RD}(\mathrm{LE}(q_i q_{i+1}), q_i q_{i+1}) = \mathcal{RD}(q', q_i q_{i+1}) \leq \epsilon$. That is, we claim that q' is a point of $\mathrm{LE}(q_i q_{i+1})$ (by Claim 3.9) with maximum ratio distance from $q_i q_{i+1}$. Since the feasible points in $\mathrm{LE}(q_i q_{i+1})$ lie between $q_i q_{i+1}$ and its parallel line through q', the claim follows. This completes the proof of Lemma 3.10. ■

To bound from above the performance ratio we need a few more lemmas. Our first lemma quantifies the area shrinkage property. It is notable that this is a statement independent of ϵ.

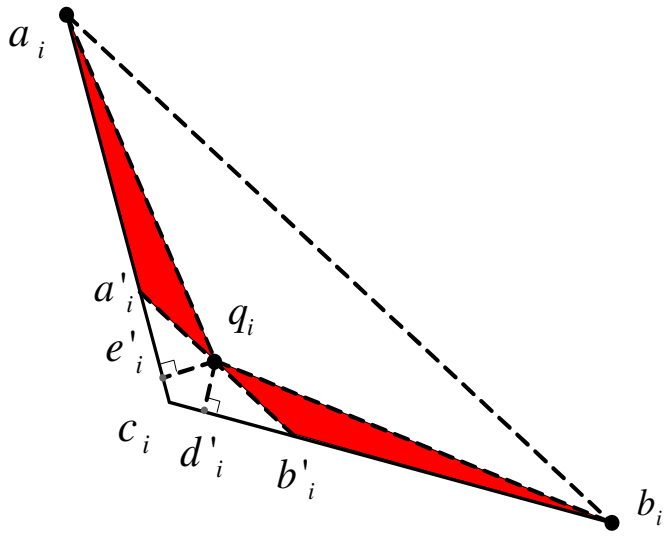

Figure 7: Illustration of the area shrinkage property of the Chord algorithm.

Lemma 3.11. *Let $T_i = \triangle(a_i b_i c_i)$ be the triangle processed by the Chord algorithm at some recursive step. Denote $q_i = \mathrm{Comb}(\lambda_{a_i b_i})$. Let $T_{i,l} = \triangle(a_i a_i' q_i)$, $T_{i,r} = \triangle(b_i b_i' q_i)$ be the triangles corresponding to the two new subproblems. Then, we have*

$$S(T_{i,l}) + S(T_{i,r}) \leq S(T_i)/4.$$

Proof. Let d_i', e_i' be the projections of q_i on $b_i c_i$ and $a_i c_i$ respectively; see Figure 7 for an illustration. Let

$$y \stackrel{\text{def}}{=} (a_i' c_i)/(a_i c_i) = (b_i' c_i)/(b_i c_i) \in [0, 1] \tag{7}$$

where the equality holds because the triangles T_i and $\triangle(a_i' b_i' c_i)$ are similar (recalling that $a_i' b_i' \parallel a_i b_i$). Hence, we get

$$S(\triangle(a_i' b_i' c_i)) = y^2 S(T_i). \tag{8}$$

We have

$$\begin{aligned} S(T_{i,l}) \;+\; S(T_{i,r}) &= \big((a_i a_i') \cdot (q_i e_i') + (b_i b_i') \cdot (q_i d_i')\big)/2 \\ &= (1-y) \cdot \big((a_i c_i) \cdot (q_i e_i') + (b_i c_i) \cdot (q_i d_i')\big)/2 \qquad (9) \\ &= (1-y) \cdot \big(S(\triangle(a_i c_i q_i)) + S(\triangle(b_i c_i q_i))\big) \\ &= (1-y) \cdot \big(S(T_{i,l}) + S(T_{i,r}) + S(\triangle(a_i' b_i' c_i))\big) \end{aligned}$$

where (9) follows from (7). By using (8) and expanding we obtain

$$S(T_{i,l}) + S(T_{i,r}) = y \cdot (1-y) \cdot S(T_i) \leq S(T_i)/4$$

as desired. ∎

Our second lemma gives a convenient lower bound on the value of the optimum.

Lemma 3.12. *Consider the recursion tree $\mathcal{T}$ built by the algorithm and let $\mathcal{L}'$ be the set of lowest internal nodes, i.e., the internal nodes whose children are leaves. Then $\mathrm{OPT}_\epsilon \geq |\mathcal{L}'|$.*

Proof. Recall that, by convention, there is no node in the tree if, for a triangle, the Chord routine terminates without calling Comb. Each lowest internal node of the tree corresponds to a triangle $T_i = \triangle(a_i b_i c_i)$ with the property that the ratio distance of the convex Pareto set from the line segment $a_i b_i$ is strictly greater than ϵ (as otherwise, the node would be a leaf). Each such triangle must contain a point $q \notin \{a_i, b_i\}$ of an optimal ϵ-CP set. Any two nodes of $\mathcal{L}'$ are not ancestor of each other, and therefore the corresponding triangles are disjoint (neighboring triangles can only intersect at an endpoint). Thus, each one of them must contain a distinct point of the optimal ϵ-CP set, and hence the lemma follows. ■

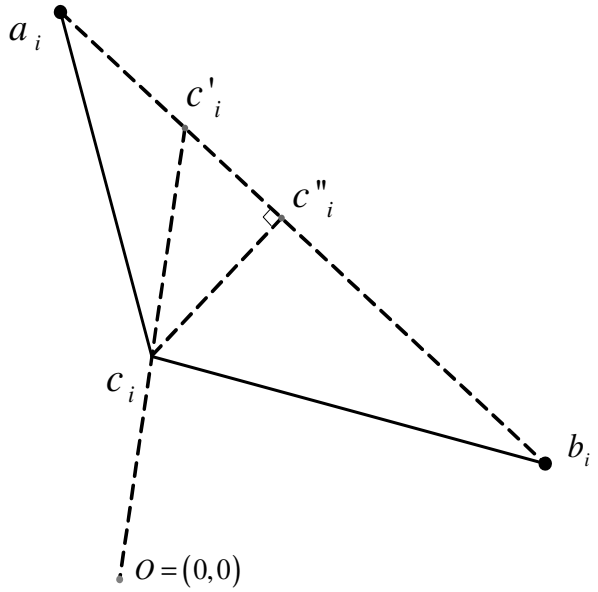

Figure 8: Illustration of the proof of Lemma 3.13.

Finally, we need a lemma that relates the ratio distance within a triangle to its area. We stress that the lemma applies only to triangles $T_i = \triangle(a_i b_i c_i)$ considered by the algorithm.

Lemma 3.13. *Consider a triangle $T_i = \triangle(a_i b_i c_i)$ considered in some iteration of the Chord algorithm such that $T_i \subseteq T_1$. Let $\alpha_i \stackrel{def}{=} \min\{x(c_i), y(c_i)\}$. If $S(T_i) \leq \epsilon^2 \cdot \alpha_i^2$, then $\mathcal{RD}(c_i, a_i b_i) \leq \epsilon$.*

Proof. The basic idea of the argument is that the worst-case for the area-error tradeoff is (essentially) when $c_i = (\alpha_i, \alpha_i)$, and the triangle T_i is right and isosceles (i.e., $(a_i c_i) = (c_i b_i)$). We now provide a formal proof. Let $T_i = \triangle(a_i b_i c_i)$ be a triangle considered by the algorithm. The points $a_i, b_i \in \mathrm{LE}(\mathcal{I})$ and we have $x(a_i) \leq x(c_i) \leq x(b_i)$, $y(a_i) \geq x(c_i) \geq x(b_i)$ and the point c_i lies below the line $a_i b_i$. Note the latter imply that $\angle(a_i c_i b_i) \geq \pi/2$. (See Figure 8.) We will relate the area $S(T_i)$ to the ratio distance $r \stackrel{\text{def}}{=} \mathcal{RD}(c_i, a_i b_i)$.

Consider the intersection c_i' of the lines $a_i b_i$ and $O c_i$, where O denotes the origin. Then we have that $c_i' = (1 + r) c_i$. From the definition of the ratio distance it follows that for any point $p \in a_i b_i$ it holds $\mathcal{RD}(c_i, p) \geq r$ (in fact, c_i' is the unique minimizer). Let c_i'' be the projection of c_i on $a_i b_i$. It follows that

$$\max\{y(c_i'')/y(c_i), x(c_i'')/x(c_i)\} \geq 1 + r. \tag{10}$$

For the area we have that $S(T_i) = (1/2)(a_ib_i)(c_ic_i'')$. Since T_i is either right or obtuse, the length of its largest base is at least twice the length of the corresponding height, i.e., $(a_ib_i) \geq 2(c_ic_i'')$. Hence, $S(T_i) \geq (c_ic_i'')^2$. By expanding and using (10), we get $S(T_i) \geq r^2 \cdot \alpha_i^2$ and the lemma follows. ■

At this point we have all the tools we need to complete the proof of Theorem 3.8.

Proof of Theorem 3.8. Lemma 3.10 gives correctness. To bound the performance ratio we proceed as follows: First, by Lemma 3.11, when a node of the tree is at depth $\lceil \log_4(S(T_1)/S_0) \rceil$, the corresponding triangle will have area at most S_0. Hence, by Lemma 3.13 (noting that $\min_i \alpha_i = \alpha$), it follows that the depth of the recursion tree $\mathcal{T}$ is $d = O(\log(S(T_1)/S_0))$. Every internal tree node is an ancestor of a node in $\mathcal{L}'$. The Chord algorithm makes one query for every node of the tree, hence $\mathrm{CHORD}_\epsilon \leq O(d) \cdot |\mathcal{L}'|$. Lemma 3.12 now implies that $\mathrm{CHORD}_\epsilon \leq O\left(\log\left(S(T_1)/S_0\right)\right) \cdot \mathrm{OPT}_\epsilon$, which concludes the proof. ■

3.2.2 Tight Upper Bound

In this subsection, we prove the asymptotically tight upper bound of $O\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$ on the worst-case performance ratio of the Chord algorithm.

The analysis is more subtle in this case and builds on the intuition obtained from the simple analysis of the previous subsection. The proof bounds in effect the length of paths in the recursion tree that consist of nodes with a single child. It shows that if we consider any ϵ-convex Pareto set S, the segment of the lower envelope between any two consecutive elements of S cannot contain too many points of the solution produced by the Chord algorithm.

Theorem 3.14. *The worst-case performance of the Chord algorithm (with respect to the ratio distance) is* $O\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$.

Proof. We begin by analyzing the case $\mathrm{OPT}_\epsilon = 3$ (i.e., the special case that one intermediate point suffices – and is required – for an ϵ-approximation) and then handle the general case. It turns out that this special case captures most of the difficulty in the analysis.

Let a (leftmost) and b (rightmost) be the extreme points of the convex Pareto curve as computed by the algorithm. We consider the case $\mathrm{OPT}_\epsilon = 3$, i.e., (i) the set $\{a, b\}$ is *not* an ϵ-CP and (ii) there exists a solution point q^* such that $\{a, q^*, b\}$ is an ϵ-CP.

Fix $k \in \mathbb{N}$ with $k = \Theta\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$. We will prove that, for an appropriate choice of the constant in the big-Theta, the Chord algorithm introduces at most k points in either of the intervals $[a, q^*]$, $[q^*, b]$. Suppose, for the sake of contradiction, that the Chord algorithm adds more points than that in the segment aq^* (the proof for q^*b being symmetric.)

We say that, in some iteration of the Chord algorithm, a triangle is *active*, if it contains the optimal point q^*. In each iteration, the Chord algorithm has an active triangle which contains the optimal point q^*. Outside that triangle, the algorithm has constructed an ϵ-approximation. We note that the Chord algorithm may in principle go back and forth between the two sides of q^*; i.e., in some iterations the line parallel to the chord touches the lower envelope to the left of q^* and in other iterations to the right.

Let $\triangle(abc)$ be the initial triangle. We focus our attention on the (not necessarily consecutive) iterations of the Chord algorithm that add points to the left of q^*. We index these iterations in increasing order with $i \in [1, k+1]$. Consider the "active" triangle $\triangle(a_ic_ib_i)$ generated in each such iteration, where a_i is the new point of the curve added in the iteration. We let $\triangle(a_0b_0c_0)$ be the initial triangle $\triangle(abc)$. It is clear that (i) a_{i+1} lies to the right of a_i, (ii) b_{i+1} lies to the left of (or is equal to) b_i and

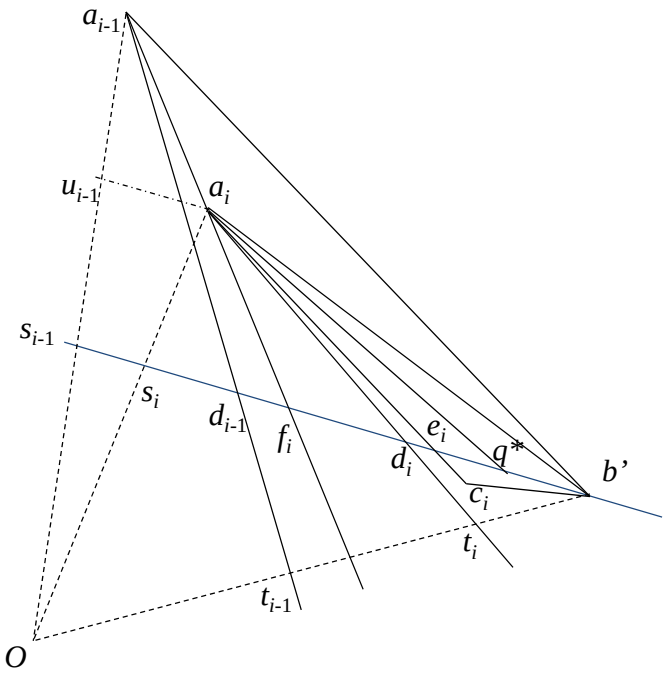

Figure 9: Illustration of the proof of Theorem 3.14.

(iii) $\triangle(a_{i+1}c_{i+1}b_{i+1}) \subseteq \triangle(a_ic_ib_i)$. Also, let us denote $b' := b_{k+1}$, that is b' is the b-vertex (i.e., the vertex that lies to the right of q^*) of the active triangle in the last iteration $k+1$. Note that b' could be equal to the point b (which would happen if the Chord algorithm introduces points only to the left of q^*, i.e., proceeds towards q^* monotonically), but in general this will not be the case.

Let e_i be the intersection point of the line a_ic_i with the line $b'q^*$; see Figure 9. Note that, to the left of q^* the convex Pareto curve has no points below the line $b'q^*$. All the points of the convex Pareto curve that are left of q^* and lie in the active triangle $\triangle(a_ic_ib_i)$ (these are the potentially not ϵ-covered points) are actually in the triangle $\triangle(a_ie_iq^*)$. Consider the line that goes through a_i and is parallel to $a_{i-1}b'$ and let d_i be its intersection with $b'q^*$. Note that the line a_ic_i is parallel to $a_{i-1}b_i$ (by construction of the algorithm, this is the line that added the point a_i and formed the active triangle $\triangle(a_ic_ib_i)$) and b_i lies to the right of (or is equal to) b', so the line a_id_i is to the left of (or is equal to) a_ic_i, hence d_i lies to the left of (or is equal to) e_i . Now, let f_i be the intersection point of $a_{i-1}a_i$ with the line $b'q^*$. Clearly, f_i lies to the left of d_i. Furthermore, f_i lies to the right of (or is equal to) d_{i-1}. The reason is that, below a_{i-1} the curve has no points (strictly) to the left of the line $a_{i-1}d_{i-1}$, so a_i is to the right of the line (or on the line).

If a point $p \in \mathbb{R}^2_+$ is above a line ℓ, in the sense that the segment connecting p to the origin O intersects ℓ, say at some point q, then we define the *excess ratio distance* of p from ℓ to be $(pq)/(Oq)$, i.e., the ratio distance of q from p. Let H_i be the excess ratio distance of a_i from the line $b'q^*$ for

$i \geq 0$. Let G_i be the excess ratio distance of b' from the line $a_i d_i$ for $i \geq 1$. Referring to Figure 9, $H_i = (a_i s_i)/(Os_i)$ and $G_i = (b't_i)/(Ot_i)$. Let $P_i = H_i/H_{i-1}$ and let $R_i = 1 - P_i$.

By definition, we have $H_i = P_i \cdot H_{i-1}$ for all i. Furthermore, H_0 is the excess ratio distance of $a = a_0$ from the line $b'q*$. Since the coordinates of the points are in $[2^{-m}, 2^m]$, it follows that $H_0 \leq 2^{2m}$. Thus, we get

$$H_k \leq 2^{2m} \cdot \prod_{i=1}^{k} P_i.$$

Claim 3.15. *$P_i = \frac{(a_i f_i)}{(a_{i-1} f_i)} = \frac{(f_i d_i)}{(f_i b')}$ and $R_i = \frac{(d_i b')}{(f_i b')}$.*

Proof. Let s_i be the intersection of the segment $a_i O$ with the line $b'q^*$ and s_{i-1} the intersection of $a_{i-1}O$ with $b'q^*$. By definition, $H_i = (a_i s_i)/(Os_i)$, and $H_{i-1} = (a_{i-1}s_{i-1})/(Os_{i-1})$. Let u_{i-1} be the intersection of $a_{i-1}O$ with the line from a_i parallel to $b'q^*$. From the similar triangles $\triangle(Os_{i-1}s_i)$ and $\triangle(Ou_{i-1}a_i)$, we have $H_i = (a_i s_i)/(Os_i) = (u_{i-1}s_{i-1})/(Os_{i-1})$. Therefore, $P_i = H_i/H_{i-1} = (u_{i-1}s_{i-1})/(a_{i-1}s_{i-1})$. From the similar triangles $\triangle(a_{i-1}s_{i-1}f_i)$ and $\triangle(a_{i-1}u_{i-1}a_i)$, the latter ratio is equal to $(a_i f_i)/(a_{i-1}f_i)$, yielding the first equality for P_i in the claim. The second equality follows from the similar triangles $\triangle(f_i a_i d_i)$ and $\triangle(f_i a_{i-1} b')$, since $a_i d_i$ is parallel to $a_{i-1}b'$. The second equality implies then the expression for $R_i = 1 - P_i$. ■

From the Claim we have, $(d_i b') = R_i \cdot (f_i b')$ and since d_{i-1} lies left of f_i, we have $(d_i b') \leq R_i \cdot (d_{i-1} b')$. Therefore,

$$(d_k b') \leq (d_1 b') \cdot \prod_{i=2}^{k} R_i.$$

Let t_i, t_{i-1} be the intersections of Ob' with the lines $a_i d_i$ and $a_{i-1}d_{i-1}$ respectively; see Figure 9. Clearly, $(b't_i)/(b't_{i-1}) \leq (b'd_i)/(b'd_{i-1})$, and hence $(b't_i) \leq R_i \cdot (b't_{i-1})$. Thus, $G_i = \frac{(b't_i)}{Ot_i} \leq \frac{R_i(b't_{i-1})}{(Ot_{i-1})} = R_i \cdot G_{i-1}$. Therefore,

$$G_k \leq G_1 \cdot \prod_{i=2}^{k} R_i.$$

Claim 3.16. *$H_k > \epsilon$ and $G_k > \epsilon$.*

Proof. Since the last iteration of the Chord algorithm adds a new point a_{k+1}, the segment $a_k b'$ does not ϵ-cover all the Pareto points left of q^* in the active triangle. These points are all in the triangle $\triangle(a_k d_k b')$. The ratio distance of any point in this triangle from $a_k b'$ is upper bounded by both $(a_k s_k)/(Os_k) = H_k$ and by $(b't_k)/(Ot_k) = G_k$. It follows that $H_k > \epsilon$ and $G_k > \epsilon$. ■

Thus, we get

$$\prod_{i=1}^{k} P_i > \epsilon/2^{2m} \tag{11}$$

and

$$G_1 \cdot \prod_{i=2}^{k} R_i > \epsilon \tag{12}$$

Claim 3.17. *$\prod_{i=2}^{k} R_i > 1/2$.*

Proof. If $G_1 \leq 2\epsilon$ then the claim follows from inequality (12). So suppose that $G_1 > 2\epsilon$. The point a_1 is at ratio distance at most ϵ from the line aq^* since $\{a, q^*, b\}$ is an ϵ-convex Pareto set. Therefore, q^* is at most excess ratio distance ϵ from the line a_1d_1, because a_1d_1 is parallel to ab' and b' is to the right of q^*. Since $G_1 > 2\epsilon$, it follows that $(d_1q^*) < (q^*b')$ and hence $(d_1b') < 2(q^*b')$. Therefore,

$$(d_kb') \leq (d_1b') \cdot \prod_{i=2}^{k} R_i < 2(q^*b') \cdot \prod_{i=2}^{k} R_i.$$

Since $(d_kb') > (q^*b')$ (as d_k is left of q^*), we conclude that

$$\prod_{i=2}^{k} R_i > 1/2.$$

■

Thus, we have a lower bound on the product of the P_i's from inequality (11) and on the product of the R_i's from Claim 3.17. It is easy to see (and is well-known) that for a fixed product of the P_i's, the product of the R_i's is maximized if all factors are equal. We include a proof for convenience.

Claim 3.18. *Let $0 < x_i < 1$ for $i = 1, \ldots, k$. The maximum of $\prod_i (1 - x_i)$ subject to $\prod_i x_i = c$ is achieved when all the x_i are equal.*

Proof. Suppose $k = 2$. Then $(1 - x_1)(1 - x_2) = 1 - (x_1 + x_2) + x_1x_2$ is maximized subject to $x_1x_2 = c$, when $x_1 + x_2$ is minimized, which happens when $x_1 = x_2$ by the arithmetic-geometric mean inequality ($x_1 + x_2 \geq 2\sqrt{x_1x_2}$ with equality iff $x_1 = x_2$. For general $k \geq 2$, if the x_i's maximize $\prod_i (1 - x_i)$ subject to $\prod_i x_i = c$ then we must have $x_i = x_j$ for all pairs $i \neq j$, because otherwise replacing x_i, x_j by their geometric mean will increase $\prod_i (1 - x_i)$. ■

Thus, for any value of $\prod_{i=2}^{k} P_i$, the product $\prod_{i=2}^{k} R_i$ is maximized when $P_i = 1/t$ for all $i = 2, \ldots, k$ and $R_i = 1 - 1/t$. Since $\prod_{i=2}^{k} R_i > 1/2$, we must have $k - 1 < t$ because $(1 - 1/t)^t < 1/e < 1/2$. Therefore, $\epsilon/2^{2m} < \prod_{i=2}^{k} P_i < 1/(k-1)^{k-1}$, hence $(k-1)^{k-1} < 2^{2m}/\epsilon$, which implies that $k = O\big(\frac{m+\log(1/\epsilon)}{\log m + \log\log(1/\epsilon)}\big)$.

We now proceed to analyze the general case, essentially by reducing it to the aforementioned special case. Suppose that the optimal solution has an arbitrary number of points, i.e., has the form $Q^* = \langle a, q_1, q_2, \ldots, q_r, b\rangle$. Charge the points computed by the Chord algorithm to the edges of the optimal solution as follows: if a point belongs to the portion of the lower envelope between the points q_{i-1} and q_i (where we let $a = q_0$ and $b = q_{r+1}$), then we charge the point to the edge $q_{i-1}q_i$; if the Chord algorithm generates a point q_i of the optimal solution then we can charge it to either one of the adjacent edges.

We claim that every edge of Q^* is charged with at most $2k + 1$ points of the Chord algorithm, where $k = O\big(\frac{m+\log(1/\epsilon)}{\log m + \log\log(1/\epsilon)}\big)$ is the same number as in the above analysis for the $\mathrm{OPT}_\epsilon = 3$ case. To see this, consider any edge $q_{i-1}q_i$ of Q^*. Let a_0 be the first point generated by the Chord algorithm that is charged to this edge, i.e., a_0 is the first point that lies between q_{i-1} and q_i. We claim that the Chord algorithm will generate at most k more points in each of the two portions $LE(q_{i-1}a_0)$ and $LE(a_0q_i)$ of the lower envelope. The argument for the two portions is symmetric.

Consider the portion $\mathrm{LE}(a_0q_i)$. The proof that the Chord algorithm will introduce at most k points in this portion is identical to the proof we gave above for the $\mathrm{OPT}_\epsilon = 3$ case, with a_0 in place of a

and q_i in place of q^*. The only fact about the assumption $\mathrm{OPT}_\epsilon = 3$ that was used there was that the edge aq^* ϵ-covers the portion of the lower envelope between a and q^*. It is certainly true here that the segment a_0q_i ϵ-covers the portion $\mathrm{LE}(a_0q_i)$, since the edge $q_{i-1}q_i$ ϵ-covers $\mathrm{LE}(q_{i-1}q_i) \supseteq \mathrm{LE}(a_0q_i)$. Hence, by the same arguments, the Chord algorithm will generate at most k points between a_0 and q_i. Thus, the algorithm will generate no more than $(2k+1)(r+1)$ points overall, and hence its performance ratio is $O\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$. This completes the proof. ■

Remark 3.19. We briefly sketch the differences in the algorithm and its analysis for the case of an approximate Comb routine. First, in this case, the description of the Chord algorithm (Table 1) has to be slightly modified; this is needed to guarantee that the set of computed points is indeed an ϵ-CP. In particular, in the Chord routine, we need to check whether $\mathcal{RD}(q, lr) \leq \epsilon'$ for an appropriate $\epsilon' < \epsilon$. In particular, we choose ϵ' such that $(1+\epsilon')(1+\delta) \leq (1+\epsilon)$, where δ is the accuracy of the approximate Comb routine, i.e., the routine Comb_δ. Consider the case that the Comb_δ routine always returns feasible points that belong to a $(1+\delta)$ scaled version of the lower envelope. The same analysis as in the current section establishes that the Chord algorithm performs at most $O\big(\frac{m+\log(1/\epsilon')}{\log m+\log\log(1/\epsilon')}\big)\mathrm{OPT}_{\epsilon'}$ calls to Comb_δ in this setting. If ϵ' is "close" to ϵ (say, $\epsilon' \geq \epsilon/2$) the first term is clearly $O\big(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\big)$. Hence, to prove the desired upper bound, it suffices to show that $\mathrm{OPT}_{\epsilon'} = O(\mathrm{OPT}_\epsilon)$. (It is clear that $\mathrm{OPT}_{\epsilon'} \geq \mathrm{OPT}_\epsilon$, but in principle it may be the case that $\mathrm{OPT}_{\epsilon'}$ is arbitrarily larger.) This is provided to us by a planar geometric lemma from [DY2] (Lemma 5.1) which states that if $(1+\epsilon') \geq \sqrt{1+\epsilon}$ then $\mathrm{OPT}_{\epsilon'} \leq 3\mathrm{OPT}_\epsilon$. Selecting $\epsilon' = \delta = \sqrt{1+\epsilon} - 1 \geq \epsilon/2$ suffices for the above and completes our sketched description.

4 Average Case Analysis

In Section 4.1 we present our average case upper bounds and in Section 4.2 we give the corresponding lower bound.

4.1 Upper Bounds

In Section 4.1.1 we start by proving our upper bound for random instances drawn from a Poisson Point Process (PPP). The analysis for the case of unconcentrated product distributions is somewhat more involved and is given in Section 4.1.2.

Overview of the Proofs. The analysis for both cases has the same overall structure, however each case has its difficulties. We start by giving a high-level overview of the arguments. For the sake of simplicity, in the following intuitive explanation, let n denote: (i) the expected number of points in the instance for a PPP and (ii) the actual number of points for a product distribution.

Similarly to the simple proof of Section 3.2.1 for worst-case instances, to analyze our distributional instances we resort to an indirect measure of progress, namely the area of the triangles maintained in the algorithm's recursion tree. We think that this feature of our analysis is quite interesting and indicates that this measure is quite robust.

In a little more detail, we first show (see Lemma 4.5 for the case of PPP) that every subdivision performed by the algorithm decreases the area between the upper and lower approximations by a significant amount (roughly at an exponential rate) with high probability. It then follows that at depth $\log\log n$ of the recursion tree, each "surviving triangle" contains an expected number of at most $\log\log n$ points with high probability. We use this fact, together with a charging argument in the same spirit as in the worst-case, to argue that the expected performance ratio is $\log\log n$ in this case.

To analyze the expected performance ratio in the complementary event, we break it into a "good" event, under which the ratio is $\log n$ with high probability, and a "bad" event, where it is potentially unbounded (in the Poisson case) or at most n (for the case of product distributions). The potential unboundedness of the performance ratio in the Poisson case creates complications in bounding the expected ratio of the algorithm over the entire space. We overcome this difficulty by bounding the upper tail of the Poisson distribution (see Claim 4.1).

In the case of product distributions, the worst case bound of n on the competitive ratio is sufficient to conclude the proof, but the technical challenges present themselves in a different form. Here, the "contents" of a triangle being processed by the algorithm depend on the information coming from the previous recursive calls making the analysis more involved. We overcome this by understanding the nature of the information provided from the conditioning.

On the choice of parameters. A simple but crucial observation concerns the interesting range for the parameters of the distributions. Suppose that we run the Chord algorithm with desired error $\epsilon > 0$ on some random instance that lies entirely in the set $[2^{-m}, 2^m]^2$. Then, it is no loss of generality to assume that the number of random points in the instance (expected number for the PPP case) is upper bounded by some fixed polynomial in 2^m and $1/\epsilon$. If this is not the case, it is easy to show that the Chord algorithm makes at most a constant number of Comb calls in expectation.

4.1.1 Poisson Point Process

For the analysis of the PPP case we will make crucial use of the following technical claim:

Claim 4.1. *Let X be a $\mathrm{Poisson}(\nu)$ random variable, with $\nu \geq 1$, and let $\mathcal{E}$ be some event. Then*

$$\mathbb{E}[X \mid \mathcal{E}] \Pr[\mathcal{E}] \leq \max\left\{\frac{1}{\nu}, O(\nu^3) \Pr[\mathcal{E}]\right\}.$$

Proof. Let k^* be such that $\Pr[X \geq k^* + 1] < \Pr[\mathcal{E}] \leq \Pr[X \geq k^*]$. Clearly,

$$\mathbb{E}[X \mid \mathcal{E}] \Pr[\mathcal{E}] \leq \sum_{i=k^*}^{+\infty} i \cdot \Pr[X = i] = \sum_{i=k^*}^{+\infty} i \cdot \frac{e^{-\nu}\nu^i}{i!} = \nu \sum_{i=k^*-1}^{+\infty} \frac{e^{-\nu}\nu^i}{i!} = \nu \Pr[X \geq k^* - 1].$$

We distinguish two cases. If $k^* - 1 \geq 2\nu^2$, then Chebyshev's inequality yields $\Pr[X \geq k^* - 1] \leq \frac{1}{\nu^2}$ which gives

$$\mathbb{E}[X \mid \mathcal{E}] \Pr[\mathcal{E}] \leq \frac{1}{\nu}.$$

If $k^* - 1 \leq 2\nu^2$, then

$$\Pr[X = k^*] \leq \frac{k^* + 1}{\nu} \Pr[X = k^* + 1] \leq O(\nu) \Pr[\mathcal{E}]$$

and

$$\Pr[X = k^* - 1] \leq \frac{(k^* + 1)^2}{\nu^2} \Pr[X = k^* + 1] \leq O(\nu^2) \Pr[\mathcal{E}].$$

Hence,

$$\begin{aligned} \mathbb{E}[X \mid \mathcal{E}] \Pr[\mathcal{E}] &\leq \nu \cdot \Pr[X \geq k^* - 1] \\ &\leq \nu \cdot \big(\Pr[\mathcal{E}] + \Pr[X = k^* - 1] + \Pr[X = k^*]\big) \\ &\leq O(\nu^3) \cdot \Pr[\mathcal{E}]. \end{aligned}$$

This concludes the proof of the claim. ∎

We start by pointing out that if the intensity of the PPP is very large, the Chord algorithm will terminate after a constant number of calls in expectation:

Proposition 4.2. *Let $T_1 = \triangle(abc)$ be at the root of the Chord algorithm's recursion tree and let ν denote the intensity of the PPP. Let $\alpha \stackrel{def}{=} \min\{x(c), y(c)\}$ and $S^* \stackrel{def}{=} (\epsilon^2\alpha^2/2) \cdot \min\{\lambda_{ab}, 1/\lambda_{ab}\}$. If $\nu \geq \nu_0 \stackrel{def}{=} 10S(T_1)/(S^*)^2$, then $\mathbb{E}[\mathrm{CHORD}_\epsilon(T_1)] = O(1)$.*

Proof. First note we can clearly assume that $y(a) > (1+\epsilon) \cdot y(b)$ and $x(b) > (1+\epsilon) \cdot x(a)$. Let $p_1 = (x(a), (1+\epsilon) \cdot y(b)) \in ac$ and $p_2 = ((1+\epsilon) \cdot x(a), y(b)) \in bc$. Let $T^* \subseteq \triangle(cp_1p_2)$ be the right triangle of maximum area whose hypotenuse is parallel to ab. (This is the shaded triangle in Figure 10.) We claim that $S(T^*) \geq S^*$. Indeed, it is clear that c is a vertex of T^* and that either p_1 or p_2 (or both) are vertices. Hence, one of the edges of T^* has length at least $\epsilon \cdot \alpha$. Since λ_{ab} is the slope of the hypotenuse, the other edge has length at least $\min\{\lambda_{ab}, 1/\lambda_{ab}\} \cdot (\epsilon\alpha)$.

If there is a feasible point in T^*, the Chord algorithm will find it by calling $\mathrm{Comb}(\lambda_{ab})$ and terminate (since such a point forms an ϵ-CP set). Let X^* be the number of random points that land in the triangle T^*. Note that X^* is a $\mathrm{Poisson}(\mu)$ random variable, with $\mu = \nu \cdot S(T^*)$. We can write

$$\mathbb{E}[\mathrm{CHORD}_\epsilon(T_1)] = \mathbb{E}[\mathrm{CHORD}_\epsilon(T_1) \mid X^* = 0] \Pr[X^* = 0] + \mathbb{E}[\mathrm{CHORD}_\epsilon(T_1) \mid X^* \geq 1] \Pr[X^* \geq 1].$$

Observe that the second term is bounded from above by a constant, hence it suffices to bound the first term. Recall that the number of calls performed by the Chord algorithm is at most twice the number Y_1 of feasible points in the root triangle T_1. Therefore, we can write

$$\mathbb{E}[\mathrm{CHORD}_\epsilon(T_1) \mid X^* = 0] \Pr[X^* = 0] \leq 2\mathbb{E}[Y_1 \mid X^* = 0] \Pr[X^* = 0].$$

Recall that Y_1 is a $\mathrm{Poisson}\,(\nu_1)$ random variable, where $\nu_1 = \nu \cdot S(T_1)$ and note that we can assume wlog that $\nu_1 \geq 1$ (since otherwise the expected number of feasible points in T_1 is at most 1 and we are done). Hence, by Claim 4.1 the RHS above is bounded by

$$2 \max\left\{\frac{1}{\nu_1}, O(\nu_1^3) \Pr[X^* = 0]\right\}.$$

Thus, to complete the proof it suffices to show that $\nu_1^3 \Pr[X^* = 0] = O(1)$. First note that this quantity equals $\nu^3 S^3(T_1) \exp(-\nu S(T^*))$ which is at most $\nu_0^3 S^3(T_1) \exp(-\nu_0 S^*)$ by monotonicity (which holds for our choice of ν_0). The latter expression is at most $O(\beta^6 \exp(-10\beta))$, where $\beta = S(T_1)/S^* > 1$, which is easily seen to be absolutely bounded. ■

The same proposition also applies for the case that we have an approximate Comb_δ routine (in this case, we replace ϵ by an appropriate $\epsilon' < \epsilon$ so that $(1+\delta)(1+\epsilon') \leq (1+\epsilon)$). We remark that a similar proposition can be shown for the Hausdorff distance; this does not hold for the case of the horizontal/vertical distance, which is why the average case bounds of this section do not apply for the latter metrics.

Note that by assumption $T_1 \subseteq [2^{-m}, 2^m]^2$ which implies that $S(T_1) \leq 2^{2m-1}$ and $\alpha \geq 2^{-m}$. We also have that $\epsilon/2^{2m} \leq \lambda_{ab} \leq 2^{2m}/\epsilon$ (since otherwise the set $\{a, b\}$ is an ϵ-CP) which gives $S^* \geq \epsilon^3 2^{-4m-1}$. Therefore, $\nu_0 = \mathrm{poly}(2^m/\epsilon)$.

The main result of this section is the following theorem, which combined with Proposition 4.2, yields the desired upper bound of $O(\log m + \log\log(1/\epsilon))$ on the expected performance ratio.

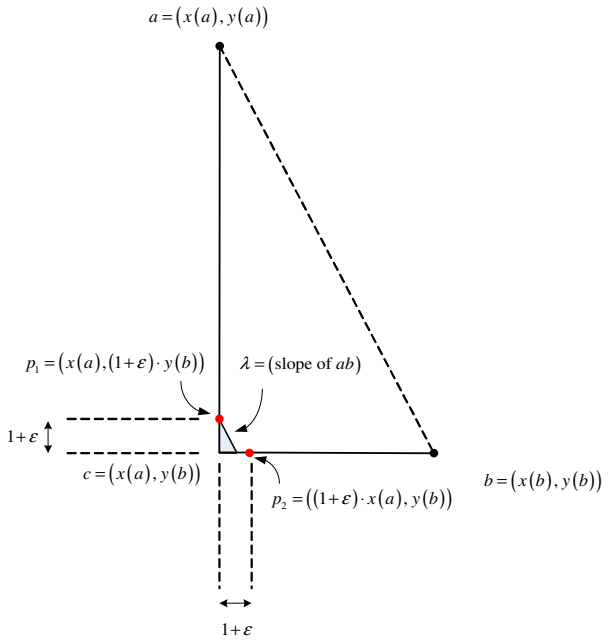

Figure 10: Illustration of Proposition 4.2.

Theorem 4.3. *Let T_1 be the triangle at the root of the Chord algorithm's recursion tree, and suppose that points are inserted into T_1 according to a Poisson Point Process with intensity ν. The expected performance ratio of the Chord algorithm on this instance is $O(\log\log(\nu S(T_1)))$.*

The proof of Theorem 4.3 will require a sequence of lemmas. Throughout this section, we will denote $S_1 \stackrel{\text{def}}{=} S(T_1)$. Recall that the number of queries performed by the Chord algorithm is bounded from above by twice the total number of points in the triangle T_1. Since the expected number of points in T_1 is $\nu \cdot S_1$, we have:

Proposition 4.4. *The expected performance ratio of the Chord algorithm is $O(\nu S_1)$.*

Hence, we will henceforth assume that νS_1 is bounded from below by a sufficiently large positive constant. (If this is not the case, the expected total number of points inside T_1 is $O(1)$ and the desired bound clearly holds.)

Our first main lemma in this section is an average case analogue of our Lemma 3.11: the lemma says that the area of the triangles maintained by the algorithm decreases *geometrically* (as opposed to linearly, as is the case for arbitrary inputs) at every recursive step (with high probability). Intuitively, this geometric decrease is what causes the performance ratio to drop by an exponential in expectation.

Lemma 4.5. *Let $T_i = \triangle(a_i b_i c_i)$ be the triangle processed by the Chord algorithm at some recursive step. Denote $q_i = \mathrm{Comb}(\lambda_{a_i b_i})$. Let $T_{i,l} = \triangle(a_i a_i' q_i)$ and $T_{i,r} = \triangle(b_i b_i' q_i)$. For all $c > 0$, with probability at least $1 - \frac{1}{(\ln(\nu S_1))^c}$ conditioning on the information available to the algorithm,*

$$S(T_{i,l}), S(T_{i,r}) \leq \sqrt{S(T_i)} \cdot \sqrt{\frac{c \cdot \ln\ln \nu S_1}{\nu}}.$$

Proof. It follows from the properties of the Chord algorithm (see e.g., Claim 3.9) that, before the Comb routine on input T_i is invoked, the following information is known to the algorithm, conditioning on the history (see Figure 11):

- There exist solution points at the locations q_j, for all $j \in [i-1]$.
- There is no point below the line $a_j c_j$, for all $j \in [i]$.
- There is no point below the line $c_j b_j$, for all $j \in [i]$.

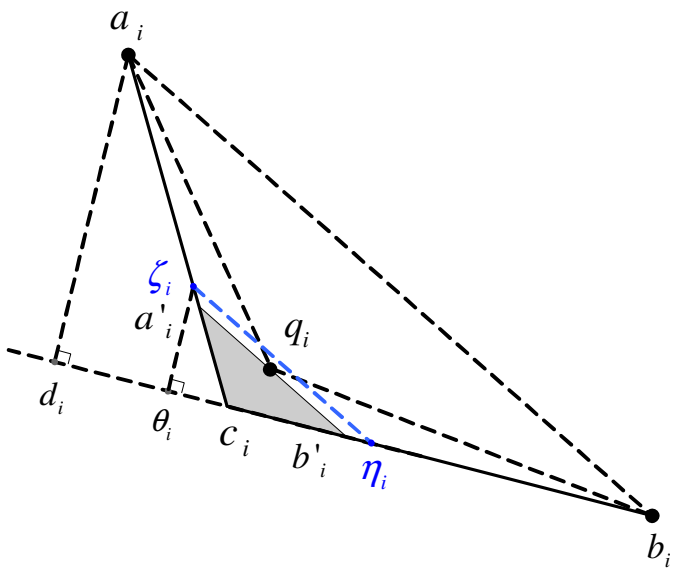

Figure 11: Illustration of the average case area shrinkage property of the algorithm.

By the definition of the Poisson Point Process, conditioning on the above information, the number of points in a region of area S inside T_i follows a Poisson distribution with parameter $\nu \cdot S$. Hence, letting $\zeta_i \eta_i$ being parallel to $a_i b_i$ so that the triangle $T^* \stackrel{\text{def}}{=} \triangle(c_i \zeta_i \eta_i)$ has area $S^* \stackrel{\text{def}}{=} \frac{c \cdot \ln\ln(\nu S_1)}{\nu}$, it follows that, with probability at least $1 - \frac{1}{(\ln(\nu S_1))^c}$, $T^* \neq \emptyset$ (i.e., the triangle contains a feasible point). Hence, with probability at least $1 - \frac{1}{(\ln(\nu S_1))^c}$ the point q_i is contained in T^*.

We bound from above the area of $T_{i,l}$ by the area of $T'_{i,l} = \triangle(a_i c_i \eta_i)$ (clearly, $T_{i,l} \subseteq T'_{i,l}$) and similarly the area of $T_{i,r}$ by the area of $T'_{i,r} = \triangle(\zeta_i c_i b_i)$. From the similarity of the triangles T_i and T^* we get

$$\frac{(\zeta_i \eta_i)}{(a_i b_i)} = \frac{(c_i \eta_i)}{(b_i c_i)} = \frac{(\zeta_i \theta_i)}{(a_i d_i)}.$$

Hence,

$$\frac{S^*}{S(T_i)} = \frac{\frac{1}{2}(c_i \eta_i)(\zeta_i \theta_i)}{\frac{1}{2}(b_i c_i)(a_i d_i)} = \left(\frac{(c_i \eta_i)}{(b_i c_i)}\right)^2,$$

which gives $(c_i \eta_i) = (b_i c_i)\sqrt{\frac{S^*}{S(T_i)}}$. Therefore,

$$\begin{aligned} S(T'_{i,l}) &= \frac{1}{2}(a_i d_i)(c_i \eta_i) = \frac{1}{2}(a_i d_i)(b_i c_i)\sqrt{\frac{S^*}{S(T_i)}} \\ &= \sqrt{S(T_i) \cdot S^*} = \sqrt{S(T_i)} \cdot \sqrt{\frac{c \cdot \ln\ln(\nu S_1)}{\nu}}. \end{aligned}$$

Finally,

$$S(T'_{i,r}) = \frac{1}{2}(\zeta_i\theta_i)(b_ic_i) = \frac{1}{2}\frac{(c_i\eta_i)}{(b_ic_i)}(a_id_i)(b_ic_i) = S(T'_{i,l}).$$

This concludes the proof of Lemma 4.5. ■

Let us choose $c \in \left(\frac{1}{\ln\ln(\nu S_1)}, \frac{\nu S_1}{\ln\ln(\nu S_1)}\right)$, and let T be a triangle maintained by the algorithm at depth d of the recursion. It follows from Lemma 4.5 that, with probability at least $1 - \frac{d}{(\ln(\nu S_1))^c}$,

$$S(T) \leq S_1^{2^{-d}} \cdot \left(\frac{c \cdot \ln\ln(\nu S_1)}{\nu}\right)^{1-2^{-d}},$$

where to bound the probability of the above event we have taken a union bound over the events on the path of the recursion tree connecting T to the root of the recursion.

Now consider the top $d^* := \lceil \log_2 \ln(\nu S_1) \rceil$ levels of the recursion tree of the algorithm. Using Lemma 4.5 and a union bound it follows that, with overall probability at least $1 - \frac{2\cdot\ln(\nu S_1)}{(\ln(\nu S_1))^c}$, the area of *every* triangle at depth (exactly) d^* of the recursion tree is bounded from above by

$$S^{**} := (e \cdot c \cdot \ln\ln(\nu S_1))/\nu,$$

where we used our assumption on the range of c.

Let $\mathcal{A}$ be the event that all the nodes (triangles) maintained by the algorithm at depth d^* of the recursion tree (if any) have area at most S^{**}. In the aforementioned, we argued that the probability of the event $\mathcal{A}$ is at least $1 - \frac{2}{(\ln(\nu S_1))^{c-1}}$. We now show the following:

Lemma 4.6. *Conditioning on $\mathcal{A}$, the expected performance ratio of the Chord algorithm is $O(\log\log(\nu S_1))$.*

Proof. Let $\mathcal{T}$ be the recursion tree of the algorithm and $\mathcal{T}_{d^*}$ be obtained from $\mathcal{T}$ by pruning it at level d^*. Let $\mathcal{V}_{d^*}$ be the set of nodes of $\mathcal{T}_{d^*}$ and let $\mathcal{L}_{d^*}$ be the subset of nodes in $\mathcal{V}_{d^*}$ that lie at depth d^* from the root. Clearly $\mathcal{L}_{d^*}$ is a subset of the leaves of $\mathcal{T}_{d^*}$.

For a triangle (node) T maintained by the algorithm at depth d^* of the recursion, that is $T \in \mathcal{L}_{d^*}$, we let the random variable X_T denote the number of points inside T. Also, denote by $\mathcal{L}'_{d^*}$ the set of lowest internal nodes of the tree $\mathcal{T}_{d^*}$. By (a straightforward analogue of) Lemma 3.12, we have $\mathrm{OPT}_\epsilon \geq |\mathcal{L}'_{d^*}|$. Also, since $\mathcal{T}_{d^*}$ is a depth d^* binary tree, it holds $|\mathcal{V}_{d^*}| \leq 2d^* \cdot |\mathcal{L}'_{d^*}|$.

We condition on the information $\mathcal{F}$ available to the algorithm in the first d^* levels of its recursion-tree (without the information obtained from processing – i.e., calling Comb for – any triangle at depth d^*). By assumption, $\mathcal{F}$ satisfies the event $\mathcal{A}$. Conditioning on the information $\mathcal{F}$, for all $T \in \mathcal{L}_{d^*}$, X_T follows a Poisson distribution with parameter $\nu \cdot S(T)$. So, given that the event $\mathcal{A}$ holds, we have $\mathbb{E}[X_T \,|\, \mathcal{F}] \leq \nu \cdot S^{**}$.

Note that the Chord algorithm makes a query to Comb for every node in the tree $\mathcal{T}_{d^*}$. Also recall that the number of queries performed by the algorithm on a triangle T containing a total number of X_T points is at most $2X_T$. Hence, the expected total number of queries made can be bounded as follows:

$$\begin{aligned}
\mathbb{E}[\mathrm{CHORD}_\epsilon \,|\, \mathcal{F}] &\leq |\mathcal{V}_{d^*}| + 2 \cdot \sum_{T \in \mathcal{L}_{d^*}} \mathbb{E}[X_T \,|\, \mathcal{F}] \\
&\leq |\mathcal{V}_{d^*}| + 2\,|\mathcal{L}_{d^*}| \cdot \nu \cdot S^{**} \\
&\leq |\mathcal{V}_{d^*}| + 4|\mathcal{L}'_{d^*}| \cdot \nu \cdot S^{**} \\
&\leq |\mathcal{L}'_{d^*}| \cdot (2d^* + 4\nu \cdot S^{**}),
\end{aligned}$$

where the third inequality uses the fact that $|\mathcal{L}_{d^*}| \leq 2\,|\mathcal{L}'_{d^*}|$. So, conditioning on the information $\mathcal{F}$, the expected performance ratio of the algorithm is

$$\begin{aligned}\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\mathcal{F}\right] &\leq \mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{|\mathcal{L}'_{d^*}|}\,\middle|\,\mathcal{F}\right]\\ &\leq (2d^* + 4\nu\cdot S^{**})\\ &= O\left(\log\log(\nu S_1)\right).\end{aligned}$$

Integrating over all possible $\mathcal{F}$ in $\mathcal{A}$ concludes the proof of Lemma 4.6. ■

From Lemma 4.6 it follows that

$$\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\mathcal{A}\right] = O\left(\log\log(\nu S_1)\right),$$

and from the preceding discussion we have that $\Pr\left[\bar{\mathcal{A}}\right] \leq \frac{2}{(\ln(\nu S_1))^{c-1}}$. Hence, we have established the following.

Lemma 4.7. *For $c \in \left(\frac{1}{\ln\ln(\nu S_1)}, \frac{\nu S_1}{\ln\ln(\nu S_1)}\right)$, there exists an event $\mathcal{A}$, with $\Pr[\mathcal{A}] \geq 1 - \frac{2}{(\ln(\nu S_1))^{c-1}}$, such that the expected performance ratio of the algorithm conditioning on $\mathcal{A}$ is $O\left(\log\log(\nu S_1)\right)$.*

Let $\mathcal{B}$ be the event that all the triangles at the level $\lceil \log_2(\nu S_1)\rceil$ of the recursion tree of the algorithm (if any) have area at most $(e\cdot c'\cdot \ln \nu S_1)/\nu$. With the same technique, but using different parameters in the argument, we can also establish the following:

Lemma 4.8. *For $c' \in \left(\frac{1}{\ln(\nu S_1)}, \frac{\nu S_1}{\ln(\nu S_1)}\right)$, there exists an event $\mathcal{B}$, with $\Pr[\mathcal{B}] \geq 1 - \frac{2}{(\nu S_1)^{c'-1}}$, such that the expected performance ratio of the algorithm conditioning on $\mathcal{B}$ is $O\left(\log(\nu S_1)\right)$.*

We want to use Lemmas 4.7 and 4.8 together with Proposition 4.4 to deduce that the expected performance ratio of the algorithm is $O\left(\log\log(\nu S_1)\right)$. This may seem intuitive, but it is in fact not immediate. For technical purposes let us define the event $\mathcal{C} = \mathcal{B} \setminus \mathcal{A}$, where $\mathcal{A}$ is the event defined in the proof of Lemma 4.7. It is easy to see that conditioning on $\mathcal{C}$ the expected performance ratio of the algorithm can still be bounded by $O(\log(\nu S_1))$, since this expectation is affected only by whatever happens at level $\lceil \log_2(\nu S_1)\rceil$ of the recursion tree and below. On the other hand, using the fact that $\Pr[\mathcal{A}] \geq 1 - \frac{2}{(\ln(\nu S_1))^{c-1}}$, it follows that

$$\Pr[\mathcal{C}] \leq \frac{2}{(\ln(\nu S_1))^{c-1}}.$$

We bound the expectation of the performance ratio using the law of total probability as follows:

$$\begin{aligned}\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\right] &\leq \mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\mathcal{A}\right]\cdot\Pr[\mathcal{A}] + \mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\mathcal{C}\right]\cdot\Pr[\mathcal{C}]\\ &+ \mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\overline{\mathcal{A}\cup\mathcal{C}}\right]\cdot\Pr\left[\overline{\mathcal{A}\cup\mathcal{C}}\right]\\ &\leq O\left(\log\log(\nu S_1)\right)\cdot\left(1 - \frac{2}{(\ln(\nu S_1))^{c-1}}\right) + O\left(\log(\nu S_1)\right)\cdot\frac{2}{(\ln(\nu S_1))^{c-1}}\\ &+ \mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\overline{\mathcal{A}\cup\mathcal{C}}\right]\cdot\Pr\left[\overline{\mathcal{A}\cup\mathcal{C}}\right]. \qquad (13)\end{aligned}$$

where (13) follows from Lemmas 4.7 and 4.8. To conclude, we need to bound the last summand in the above expression. Note first that $\mathcal{B} \subseteq \mathcal{A} \cup \mathcal{C}$. Hence,

$$\Pr\left[\overline{\mathcal{A} \cup \mathcal{C}}\right] \leq \frac{2}{(\nu S_1)^{c'-1}}.$$

We again use the fact that the number of queries made by the Chord algorithm (hence, also the performance ratio) is bounded by twice the total number of points in the triangle at the root of the recursion tree. This number X follows a Poisson distribution with parameter $\nu \cdot S_1$. Hence, we have

$$\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\overline{\mathcal{A} \cup \mathcal{C}}\right] \cdot \Pr\left[\overline{\mathcal{A} \cup \mathcal{C}}\right] \leq 2 \cdot \mathbb{E}\left[X \,\middle|\, \overline{\mathcal{A} \cup \mathcal{C}}\right] \cdot \Pr\left[\overline{\mathcal{A} \cup \mathcal{C}}\right].$$

To bound the right hand side of the above inequality we use Claim 4.1 and obtain:

$$\begin{aligned}\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\,\middle|\,\overline{\mathcal{A} \cup \mathcal{C}}\right] \cdot \Pr\left[\overline{\mathcal{A} \cup \mathcal{C}}\right] &\leq \max\left\{\frac{1}{\nu S_1}, O((\nu S_1)^3) \cdot \Pr\left[\overline{\mathcal{A} \cup \mathcal{C}}\right]\right\} \\ &\leq \max\left\{\frac{1}{\nu S_1}, O((\nu S_1)^3)\frac{2}{(\nu S_1)^{c'-1}}\right\}.\end{aligned}$$

Choosing $c' = 4$, the above RHS becomes $O(1)$. Plugging this into (13) with $c = 2$ gives

$$\mathbb{E}\left[\frac{\mathrm{CHORD}_\epsilon}{\mathrm{OPT}_\epsilon}\right] = O\left(\log\log(\nu S_1)\right).$$

This concludes the proof of Theorem 4.3.

4.1.2 Product Distributions

We start by proving the analogue of Proposition 4.2.

Proposition 4.9. *Let $T_1 = \triangle(abc)$ be at the root of the Chord algorithm's recursion tree and suppose that n points are inserted into T_1 independently from a γ-balanced distribution, where $\gamma \in [0, 1)$. Let $\alpha \stackrel{def}{=} \min\{x(c), y(c)\}$, $S^* \stackrel{def}{=} (\epsilon^2\alpha^2/2) \cdot \min\{\lambda_{ab}, 1/\lambda_{ab}\}$ and $\beta \stackrel{def}{=} S(T_1)/S^*$. If $n \geq n_0 \stackrel{def}{=} 10\beta^2/(1-\gamma)^2$, then $\mathbb{E}[\mathrm{CHORD}_\epsilon(T_1)] = O(1)$.*

Proof. As in Proposition 4.2 we have that $S(T^*) \geq S^*$ (see Figure 10). The probability that a random point falls into T^* is at least $t^* = (1-\gamma)(1/\beta)$, hence the probability than none of the n points falls into T^* is at most $(1-t^*)^n$. As before, if there is a feasible point in T^*, the Chord algorithm will find it and terminate. In all cases, the algorithm terminates after at most $2n$ calls to Comb. Hence, $\mathbb{E}[\mathrm{CHORD}_\epsilon(T)] \leq 3 + 2n \cdot (1-t^*)^n$. The last summand is bounded by $2n \cdot e^{-t^* \cdot n}$ which is at most $2n_0 \cdot e^{-t^* \cdot n_0}$ for all $n \geq n_0$ by monotonicity. The latter quantity equals $O(r^2 e^{-10r})$, where $r = \beta/(1-\gamma) > 1$, which is easily seen to be $O(1)$. ■

Recalling that $S(T_1) \leq 2^{2m-1}$ and $S^* \geq \epsilon^3 2^{-4m-1}$ we deduce that $n_0 = \mathrm{poly}(2^m/\epsilon)/(1-\gamma)^2$.
The main result of this section is devoted to the proof of the following theorem:

Theorem 4.10. *Let T_1 be the triangle at the root of the Chord algorithm's recursion tree, and suppose that n points are inserted into T_1 independently from a γ-balanced distribution, where $\gamma \in [0, 1)$. The expected performance ratio of the Chord algorithm on this instance is $O_\gamma(\log\log n)$.*

Combined with Proposition 4.9, the theorem yields the desired upper bound of $O_\gamma(\log m + \log\log(1/\epsilon))$. The proof has the same overall structure as the proof of Theorem 4.3, but the details are more elaborate. We emphasize below the required modifications to the argument.

Since the performance ratio of the Chord algorithm is at most $2n$ on any instance with n points, we will assume that n is lower bounded by a sufficiently large absolute constant ($n \geq 12$ suffices for our analysis). We start by giving an area shrinkage lemma, similar to Lemma 4.5. (See Figure 11 for an illustration.)

Lemma 4.11. *Let $T_i = \triangle(a_i b_i c_i)$ be a triangle processed by the Chord algorithm at recursion depth at most $\lceil \log_2 \ln n \rceil - 1$. Denote $q_i = \mathrm{Comb}(\lambda_{a_i b_i})$. Let $T_{i,l} = \triangle(a_i a_i' q_i)$, $T_{i,r} = \triangle(b_i b_i' q_i)$ and $T_i' = \triangle(a_i' c_i b_i')$. For all $c > 0$, with probability at least $1 - \ln n^{-\frac{c(1-\gamma)^2}{2}}$ conditioning on the information available to the algorithm,*

$$S(T_{i,l}), S(T_{i,r}) \leq \sqrt{S(T_i) S_1} \cdot \sqrt{\frac{c \cdot \ln\ln n}{n}}; \text{ and } S(T_i') \leq S_1 \frac{c \cdot \ln\ln n}{n}.$$

Proof. We follow the proof of Lemma 4.5 with the appropriate modifications. Let T_i be the triangle maintained by the algorithm at some node of the recursion tree, and suppose that T_i is at recursion depth at most $\lceil \log_2 \ln n \rceil - 1$. The information available to the algorithm when it processes T_i (before it makes the query $\mathrm{Comb}(\lambda_{a_i b_i})$) is:

- The location of all points q_j, $j \in [i-1]$ is known; by our assumption on the depth it follows that $i \leq 2 \ln n$.
- There is no point below the line (defined by) $a_j c_j$, or below the line $c_j b_j$, for all $j \in [i]$.

Given this information, the probability that, among the remaining $n_i \geq n - 2\ln n$ points (whose location is unknown), none falls inside a triangle T of area $S(T)$ inside T_i, is at most

$$\left(1 - (1-\gamma)^2 \frac{S(T)}{S_1}\right)^{n_i}.$$

Indeed, let $T_i^* \subseteq T_1$ be the subset of the root triangle which is available for the location of q_i; this is the convex set below the line $a_1 b_1$, to the right of all lines $a_j c_j$, for $j \in [i]$, and above all lines $c_j b_j$, for $j \in [i]$. The probability that a point whose location is unknown falls inside $T \subseteq T_i^*$ is

$$\frac{\mathcal{D}[T]}{\mathcal{D}[T_i^*]} \geq \frac{(1-\gamma)\mathcal{U}[T]}{\mathcal{U}[T_i^*]/(1-\gamma)} \geq \frac{(1-\gamma)\mathcal{U}[T]}{\mathcal{U}[T_1]/(1-\gamma)} = (1-\gamma)^2 \frac{S(T)}{S_1}.$$

Choosing $S(T) \stackrel{\text{def}}{=} S_1 \frac{c \cdot \ln\ln n}{n}$, the probability becomes

$$(1-\gamma)^2 \cdot \frac{c \cdot \ln\ln n}{n}.$$

Hence, the probability that T is empty is at most

$$\left(1 - c(1-\gamma)^2 \frac{\ln\ln n}{n}\right)^{n_i} \leq e^{-c(1-\gamma)^2 \frac{\ln\ln n}{n} n_i} \leq \left(\frac{1}{\ln n}\right)^{\frac{c(1-\gamma)^2}{2}}.$$

The proof of the lemma is concluded by identical arguments as in the proof of Lemma 4.5. ∎

Using Lemma 4.11 and the union bound, we can show that, with probability at least

$$1 - \frac{2}{(\ln n)^{\frac{c(1-\gamma)^2}{2}-1}},$$

the following are satisfied:

- All triangles maintained by the algorithm at depth $\lceil \log_2 \ln n \rceil$ of its recursion tree have area at most
$$S_1 \cdot \frac{e \cdot c \cdot \ln \ln n}{n}.$$
- For every node (triangle) i in the first $\lceil \log_2 \ln n \rceil - 1$ levels of the recursion tree
$$S(T_i') \leq S_1 \frac{c \cdot \ln \ln n}{n},$$
where T_i' is defined as in the statement of Lemma 4.11.

The proof of the second assertion above follows immediately from Lemma 4.11 and the union bound. The first assertion is shown similarly to the analogous assertion of Theorem 4.3. For the above we assumed that $c \in \left(\frac{1}{\ln \ln n}, \frac{n}{\ln \ln n}\right)$.

Now let us call $\mathcal{A}_c$ the event that the above assertions are satisfied. We can show the following.

Lemma 4.12. *Suppose $c \leq \frac{n}{4 \cdot \ln n \cdot \ln \ln n}$. Conditioning on the event $\mathcal{A}_c$, the expected performance ratio of the Chord algorithm is $O_{c,\gamma}(\log \log n)$.*

Proof. The proof is in the same spirit to the proof of Lemma 4.6, but more care is needed. We need to argue that, under $\mathcal{A}_c$, the expected number of points falling inside a triangle at depth $\lceil \log_2 \ln n \rceil$ of the recursion tree is $O_{c,\gamma}(\log \log n)$. Using rationale similar to that used in the proof of Lemma 4.11 above, we have the following: Let T_i be the triangle maintained by the algorithm at a node i at depth $\lceil \log_2 \ln n \rceil$ of the recursion tree. Let also p be a point whose location is unknown to the algorithm (conditioning on the information known to the algorithm after processing the first $\lceil \log_2 \ln n \rceil - 1$ levels of the recursion tree). The probability that the point p falls inside T_i is

$$\frac{\mathcal{D}[T_i]}{\mathcal{D}[T_i^*]} \leq \frac{\mathcal{U}[T_i]/(1-\gamma)}{(1-\gamma)\mathcal{U}[T_i^*]},$$

where T_i^* is the region below the line $a_1 b_1$, above the lines $a_j c_j$, for all j in the first $\lceil \log_2 \ln n \rceil$ levels of the recursion tree, and above the lines $c_j b_j$, for all j in the first $\lceil \log_2 \ln n \rceil$ levels of the recursion tree. To upper bound the probability that p falls inside T_i, we need a lower bound on the size of the area $S(T_i^*)$. Such bound can be obtained by noticing that

$$S(T_i^*) = S_1 - \sum_j S(T_j'),$$

where the summation ranges over all j in the first $\lceil \log_2 \ln n \rceil - 1$ levels of the recursion tree. Hence,

$$\begin{aligned} S(T_i^*) &\geq S_1 - 2 \ln n \cdot S_1 \frac{c \cdot \ln \ln n}{n} \\ &= S_1 \cdot \frac{n - 2 \cdot c \cdot \ln n \cdot \ln \ln n}{n} \geq \frac{S_1}{2}, \end{aligned}$$

where we used that $c \leq \frac{n}{4 \cdot \ln n \cdot \ln \ln n}$. Hence, the probability that a point falls inside T_i is at most

$$\frac{\mathcal{U}\left[T_i\right]/(1-\gamma)}{(1-\gamma)\mathcal{U}[T_i^*]} \leq \frac{1}{(1-\gamma)^2} \cdot \frac{S_1 \cdot \frac{e \cdot c \cdot \ln \ln n}{n}}{S_1/2} \leq \frac{2 \cdot e \cdot c \cdot \ln \ln n}{(1-\gamma)^2 n}.$$

Therefore, the expected number of points falling in T_i is at most

$$\frac{2 \cdot e \cdot c \cdot \ln \ln n}{(1-\gamma)^2}.$$

The final part of the proof is a charging argument identical to the one in Lemma 4.6. ■

We have thus established the following:

Lemma 4.13. *For $c \in \left(\frac{1}{\ln \ln n}, \frac{n}{4 \cdot \ln n \cdot \ln \ln n}\right)$, there exists an event $\mathcal{A}_c$, with $\Pr[\mathcal{A}_c] \geq 1 - \frac{2}{(\ln n)^{0.5c(1-\gamma)^2-1}}$, such that the expected performance ratio of the Chord algorithm conditioning on $\mathcal{A}_c$ is $O_{c,\gamma}(\log \log n)$.*

We can also show the analogue of Lemma 4.8. Namely,

Lemma 4.14. *For $c' \in \left(\frac{1}{\ln n}, \frac{\ln n}{6}\right)$, there exists an event $\mathcal{B}_{c'}$, with $\Pr[\mathcal{B}_{c'}] \geq 1 - \frac{2}{n^{0.5c'(1-\gamma)^2-1}}$, such that the expected performance ratio of the Chord algorithm conditioning on $\mathcal{B}_{c'}$ is $O_{c',\gamma}((\log n)^3)$.*

Proof. The proof is similar to the proof of Lemma 4.13, except that the bound is now a bit trickier. For $c' \in \left(\frac{1}{\ln n}, \frac{\ln n}{6}\right)$, let $\mathcal{B}_{c'}$ be the event that

- all the triangles maintained by the algorithm at depth $\lceil \log_2 n \rceil$ of its recursion tree have area at most $S_1 \cdot \frac{e \cdot c' \cdot \ln n}{n}$.
- for every node i inside the first $\lceil \log_2 n \rceil - 1$ levels of the recursion tree

$$S(T_i') \leq S_1 \frac{c' \cdot \ln n}{n},$$

  where $S(T_i')$ is defined as in the statement of Lemma 4.11.

Using arguments similar to those in the proof of Lemma 4.11 and the union bound, we obtain that

$$\Pr[\mathcal{B}_{c'}] \geq 1 - \frac{2}{n^{0.5c'(1-\gamma)^2-1}}.$$

Now let $d^* \stackrel{\text{def}}{=} \lceil \log_2 n \rceil$ and $\mathcal{T}_{d^*}$ be the recursion tree of the algorithm pruned at level d^*. We also define the set $\mathcal{L}'_{d^*}$ as in the proof of Lemma 4.6, (but with d^* replaced by $\lceil \log_2 n \rceil$). As in that proof, any ϵ-CP set needs to use at least $|\mathcal{L}'_{d^*}|$ points. Moreover, the total number of nodes inside $\mathcal{T}_{d^*}$ is at most $2d^*|\mathcal{L}'_{d^*}|$. Whenever the algorithm processes a triangle, a planar region of area at most $S_1 \cdot \frac{c' \cdot \ln n}{n}$ is removed from T_1 (the root triangle). Therefore, after finishing the processing of the first $\lceil \log_2 n \rceil - 1$ levels of the tree, a total area of at most

$$2\lceil \log_2 n \rceil S_1 \cdot \frac{c' \cdot \ln n}{n} |\mathcal{L}'_{d^*}|$$

is removed from T_1.

We distinguish two cases. If $|\mathcal{L}'_{d^*}| \geq \frac{n}{\lceil \ln n \rceil^3}$, then the size of the optimum is at least $\frac{n}{\lceil \ln n \rceil^3}$ points. Since there is a total of n points (and the algorithm never performs more than $2n$ Comb calls), it follows that in this case the performance ratio is $O(\log^3 n)$.

On the other hand, if $|\mathcal{L}'_{d^*}| \leq \frac{n}{\lceil \ln n \rceil^3}$, then the total area that has been removed from T_1 is at most $2S_1 \cdot \frac{c'}{\ln n}$. Hence, the remaining area is at least $S_1/2$, assuming $c' \leq \ln n/6$. Given this bound it follows that the expected number of points inside a triangle at level $\lceil \log_2 n \rceil$ of the recursion tree is at most

$$\frac{2 \cdot e \cdot c' \cdot \ln n}{(1-\gamma)^2}.$$

Using the aforementioned and noting that the performance ratio "paid" within the first $\lceil \log_2 n \rceil - 1$ levels of the recursion tree is at most $O_{c',\gamma}(\log n)$, we can conclude the proof via arguments parallel to those in the proof of Lemma 4.6. ■

Now let us choose $c = \frac{8}{(1-\gamma)^2}$ and $c' = \frac{4}{(1-\gamma)^2}$. From Lemmas 4.13 and 4.14 we have that

$$\Pr[\mathcal{A}_c] \geq 1 - \frac{2}{(\ln n)^3} \text{ and } \Pr[\mathcal{B}_{c'}] \geq 1 - \frac{2}{n}.$$

Given this, we can conclude the proof Theorem 4.10. The argument is the same as the end of the proof of Theorem 4.3, except that we can now trivially bound the performance ratio of the algorithm by $2n$ in the event $\overline{\mathcal{A} \cup \mathcal{C}}$.

4.2 Lower Bounds

In this section we show that our upper bounds on the expected performance of the algorithm are tight up to constant factors. In particular, for the case of the Poisson point process we prove:

Theorem 4.15. *Let T_1 be the triangle at the root of the Chord algorithm's recursion tree, and suppose that points are inserted into T_1 according to a Poisson Point Process with intensity ν. There exists an infinite family of instances (parameterized by ϵ and m) on which the expected performance ratio of the Chord algorithm is $\Omega(\log\log(\nu S(T_1)))$. In particular, we can select the parameters so that $S(T_1) = \mathrm{poly}(2^m/\epsilon)$, which yields a lower bound of $\Omega(\log m + \log\log 1/\epsilon)$.*

Proof. The lower bound construction is reminiscent to the worst-case instance of Section 3.1. In particular, the initial triangle $T_1 = \triangle(a_1 b_1 c_1)$ (at the root of the recursion tree) will be right and $(a_1 c_1) >> (b_1 c_1)$. To avoid clutter in the expressions, we present the proof for the case $m = 1$. The generalization for all values of m is straightforward.

We fix $a_1 = (1, 2)$, $b_1 = (1 + 2\epsilon, 1)$ and $c_1 = (1, 1)$ and select the intensity of the Poisson process to be $\nu \stackrel{\text{def}}{=} 1/\epsilon^2$. Note that for this setting of the parameters we have that $\nu S(T_1) = 1/\epsilon$, we thus obtain an $\Omega(\log\log(1/\epsilon))$ lower bound.

Given the endpoints a_1, b_1, it is clear that $\mathrm{OPT}_\epsilon \leq 3$ with probability 1. Hence, it suffices to show that the Chord algorithm makes $\Omega(\log\log(1/\epsilon))$ Comb calls in expectation before it terminates. To show this, we are in turn going to prove that for ϵ being a sufficiently small positive constant, with constant probability, there exists a path $\mathcal{P}$ of length $\Omega(\log\log(1/\epsilon))$ in the recursion tree of the algorithm.

As shown in Lemma 3.5, for such instances the ratio distance is very well approximated by the horizontal distance. In particular, consider the triangle $T_i = \triangle(a_i b_i c_i)$ (see Figure 12), where $(b_i c_i) \leq 2\epsilon$. For any point $s \in T_i$, we have that

$$\mathcal{RD}(s, a_i b_i) \leq \Delta x(s, a_i b_i) \leq \mathcal{RD}(s, a_i b_i) + 4\epsilon^2 + 2\epsilon/\lambda_{a_i b_i}.$$

Hence, as long as $\lambda_{a_i b_i}$ (the slope of $a_i b_i$) is sufficiently large, we can essentially use the horizontal distance metric as a proxy for the ratio distance. Indeed, this will be the case for our lower bound instance below.

We now proceed to formally describe the construction. The path $\mathcal{P}$ of length $\Omega(\log\log(1/\epsilon))$ will be defined by the triangles with $b_i = b_1$, i.e., the ones corresponding to the right subproblem in every subdivision performed by the algorithm. For notational convenience, we shift the coordinate system to the point c_1, so that $c_1 = (0, 0)$, $a_1 = (0, 1)$ and $b_1 = (2\epsilon, 0)$. (Note that the horizontal distance is invariant under translation.) We label the triangles in the path $\mathcal{P}$ by $T_1, T_2, \ldots$, and we let the vertices of triangle T_i be $a_i = (x_i, y_i)$, $c_i = (x'_i, 0)$, and $b_i = b_1$. Suppose that when the Chord algorithm processes the triangle T_i, the Comb routine returns the point q_i on a line $a'_i b'_i$ parallel to $a_i b_i$ (as in Figure 12). Note that $b'_i = c_{i+1}$ and $q_i = a_{i+1}$. Let $a'_i = (\mathcal{X}_i, \mathcal{Y}_i)$.

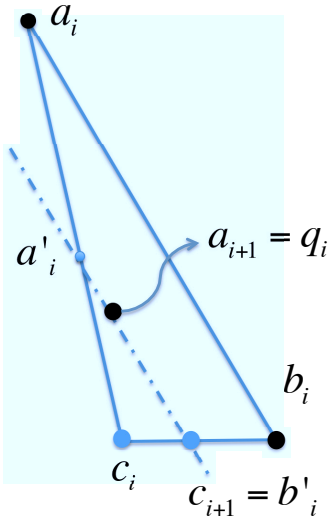

Figure 12: Illustration of the average case lower bound construction.

Let $j^* \stackrel{\text{def}}{=} (1/2)\lceil \log\log(1/\epsilon)\rceil$. The theorem follows easily from the next lemma:

Lemma 4.16. *Let ϵ be a sufficiently small positive constant. With probability at least $1 - 4j^*/(\ln(1/\epsilon))^{1/4}$, for all $i \in [j^*]$,*

$$y_i \geq c'_i \cdot \epsilon^{1-2^{-i+1}}/\ln(1/\epsilon); \qquad \textit{and} \qquad x'_i \leq c''_i \cdot \epsilon^{1+2^{-i+1}} \cdot \ln(1/\epsilon), \tag{14}$$

where $c'_i = 2^{1-2^{-i}}$ and $c''_i = \sqrt{2} \cdot \sum_{j=1}^{i} 2^{2^{-j}}$.

Proof. It is clear that (14) is satisfied for $i = 1$. We are going to show by induction that, if (14) holds for some i, it also holds for $i+1$ with probability at least $1-4/(\ln(1/\epsilon))^{1/4}$. The theorem then follows by a union bound over all $i \in [j^*]$.

By the similarity of the triangles $T_i = \triangle(a_i b_i c_i)$ and $T'_i = \triangle(a'_i b'_i c_i)$ we have

$$\frac{y_i}{\mathcal{Y}_i} = \frac{2\epsilon - x'_i}{x'_{i+1} - x'_i}. \tag{15}$$

From the properties of the Poisson Point Process we have the following: conditioning on the information available to the algorithm when it processes the triangle T_i, if a measurable region inside T_i has area $S^* = \epsilon^2 \ln(1/\epsilon)$, then the number of points inside this region follows a Poisson distribution with parameter $\nu \cdot S^* = \ln(1/\epsilon)$. Hence, with probability at least $1 - \epsilon$, any such region contains at least one point. Hence, with probability at least $1 - \epsilon$ we have that $S(T'_i) \leq S^*$. Note that $S(T'_i) = (1/2)(x'_{i+1} - x'_i)\mathcal{Y}_i$. Using (15) and the induction hypothesis, this implies that, with probability at least $1 - \epsilon$,

$$x'_{i+1} - x'_i \leq \frac{2}{\sqrt{c'_i}} \cdot \ln(1/\epsilon) \cdot \epsilon^{1+2^{-i}};$$

hence

$$x'_{i+1} \leq \left(\frac{2}{\sqrt{c'_i}} + c''_i \right) \cdot \ln(1/\epsilon) \cdot \epsilon^{1+2^{-i}}$$

as desired.

On the other hand, if the area of a region is no more than $S^{**} = \epsilon^2/\sqrt{\ln(1/\epsilon)}$ the probability that a point is contained in that region is at most $\nu \cdot S^{**} = 1/\sqrt{\ln(1/\epsilon)}$. Similarly, this implies that, with probability at least $1 - 1/\sqrt{\ln(1/\epsilon)}$,

$$\mathcal{Y}_i \geq \frac{\sqrt{c'_i} \cdot \epsilon^{1-2^{-i}}}{(\ln(1/\epsilon))^{3/4}}.$$

By the properties of the Poisson Point Process it follows that the point q_i is uniformly distributed on the segment $a'_i b'_i$. Hence, with probability at least $1 - \frac{\sqrt{2}}{(\ln(1/\epsilon))^{1/4}}$, it holds

$$y_{i+1} \geq \frac{\sqrt{2c'_i} \cdot \epsilon^{1-2^{-i}}}{\ln(1/\epsilon)}.$$

A union bound concludes the proof. ■

We now show how the theorem follows from the above lemma. First note that, for all i it holds $\sqrt{2} \cdot \sum_{j=1}^{i} 2^{2^{-j}} \leq 2\sqrt{2} \cdot i$. Hence, by Lemma 4.16 and the choice of j^*, it is easy to check that with probability at least $1 - 4j^*/(\ln(1/\epsilon))^{1/4}$, we have $x'_{j^*} = o(\epsilon)$ and $y_{j^*} = \omega(\epsilon)$. The latter condition implies that the horizontal distance is a very good approximation to the ratio distance. The latter, combined with the first condition, implies that the node (corresponding to the triangle) T_{j^*} is not a leaf of the recursion tree. That is, all the triangles $T_1, \ldots, T_{j^*}$ survive in the recursion tree, since the Chord algorithm does not have a certificate that the points already computed are enough to form an ϵ-CP set. This concludes the proof of the theorem. ■

An analogous result can be shown similarly for the case of n points drawn from a balanced distribution.

5 Conclusions and Open Problems

We studied the Chord algorithm, a simple popular greedy algorithm that is used (under different names) for the approximation of convex curves in various areas. We analyzed the performance ratio of the algorithm, i.e., the ratio of the cost of the algorithm over the minimum possible cost required to achieve a desired accuracy for an instance, with respect to the Hausdorff and the ratio distance. We showed sharp upper and lower bounds, both in a worst case and in an average setting. In the worst case the Chord algorithm is roughly at most a logarithmic factor away from optimal, while in the average case it is at most a doubly logarithmic factor away.

We showed also that no algorithm can achieve a constant ratio in the worst-case, in particular, at least a doubly logarithmic factor is unavoidable. We leave as an interesting open problem to determine if there is an algorithm with a better performance than the Chord algorithm (both in the worst-case and in average case settings), and to determine what is the best ratio that can be achieved. Another interesting direction of further research is to analyze the performance of the Chord algorithm in three and higher dimensions, and to characterize what is the best performance ratios that can be achieved by any algorithm.

References

[AN] Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problems. *Management Science*, pp. 73-78, 1979.

[Ar] Archimedes. Quadratura parabolae. *Archimedis opera omnia*, vol II, J. L. Heiberg (ed.), B. G. Teubner, Leibzig, pp. 261-315, 1913.

[BHR] R. E. Burkard, H. W. Hamacher and G. Rote. Sandwich approximation of univariate convex functions with an application to separable convex programming. *Naval Res. Logistics*, 38, pp. 911-924, 1991.

[CCS] J. Cohon, R. Church and D. Sheer. Generating multiobjective tradeoffs: An algorithm for the bicriterion problem. *Water Resources Research* 15, pp. 1001-1010, 1979.

[CHSB] D. L. Craft, T. F. Halabi, H. A. Shih and T. R. Bortfeld. Approximating convex Pareto surfaces in multiobjective radiotherapy planning. *Med. Phys.*, 33, pp. 3399-3407, 2006.

[CY] R. Cole and C. Yap. Shape from probing. *J. Algorithms* 8, pp. 19-38, 1987.

[DP] D. Douglas and T. Peucker. Algorithms for the reductions of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2), pp. 112-122, 1973.

[DY] I. Diakonikolas and M. Yannakakis. Small Approximate Pareto Sets for Bi-objective Shortest Paths and Other Problems. In *SIAM J. on Computing*, 39(4): 1340-137, 2009.

[DY2] I. Diakonikolas and M. Yannakakis. Succinct Approximate Convex Pareto Curves. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pp. 74-83, 2008.

[Ehr] M. Ehrgott. *Multicriteria optimization*, 2nd edition, Springer-Verlag, 2005.

[EG] M. Ehrgott and X. Gandibleux. An annotated bibliography of multiobjective combinatorial optimization problems. *OR Spectrum* 42, pp. 425-460, 2000.

[ES] M. J. Eisner and D. G. Severance. Mathematical techniques for efficient record segmentaton in large shared databases. *J. ACM* 23(4), pp. 619-635, 1976.

[FBR] B. Fruhwirth, R. E. Burkard and G. Rote. Approximation of convex curves with application to the bicriteria minimum cost flow problem. *European J. of Operational Research* 42, pp. 326-338, 1989.

[FGE] J. Figueira, S. Greco, M. Ehrgott, eds. *Multiple Criteria Decision Analysis: State of the Art Surveys*, Springer, 2005.

[LB] M. Lindenbaum, A. M. Bruckstein. Blind approximation of planar convex sets. *IEEE Trans. on Robotics and Automation* 10(4), pp. 517-529, 1994.

[Mit] K. M. Miettinen. *Nonlinear Multiobjective Optimization*, Kluwer, 1999.

[PY1] C.H. Papadimitriou, M. Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In Proc. 41st IEEE Symp. on Foundations of Computer Science, pp. 86-92, 2000.

[Ra] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* 1, pp. 244-256, 1972.

[Ro] G. Rote. The convergence rate of the sandwich algorithm for approximating convex functions. *Computing*, 48, pp. 337-361, 1992.

[RF] G. Ruhe and B. Fruhwirth. Epsilon-optimality for bicriteria programs and its application to minimum cost flows. *Computing* , 44, pp. 21–34, 1990.

[RW] S. Ruzika and M. M. Wiecek. Approximation Methods in Multiobjective Programming (Survey paper). *J. Opt. Th. and Appl.*, 126(3), pp. 473-501, 2005.

[VV] P. Van Mieghen and L. Vandenberghe. Trade-off Curves for QoS Routing. In *Proc. INFOCOM*, 2006.

[VY] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science* 348, pp. 334-356, 2005. (Preliminary version in *Proc. ICALP*, pp. 1201-1213, 2004.)

[YG] X. Yang and C. Goh. A method for convex curve approximation. *European J. of Oper. Res.* 97, pp. 205-212, 1997.