

# How good is the Chord algorithm?

Constantinos Daskalakis\*

Ilias Diakonikolas†

Mihalis Yannakakis†

## 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), 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 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  $\epsilon$ -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  $\epsilon$ -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  $\epsilon$ -Pareto sets and  $\epsilon$ -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  $\epsilon$ -Pareto set (and also  $\epsilon$ -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  $\epsilon$ -Pareto and  $\epsilon$ -convex

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

†Department of Computer Science, Columbia University. Email: {ilias,mihalis}@cs.columbia.edu. Research supported by NSF grants CCF-04-30946, CCF-07-28736, and by an Alexander S. Onassis Foundation Fellowship to the first author.

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 weights  $w_i \geq 0$  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  $\epsilon$ -convex Pareto set for a desired small  $\epsilon$ . It is shown in [DY2] that a necessary and sufficient condition for the polynomial time constructibility of an  $\epsilon$ -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  $\epsilon$  with the fewest number of calls to the *Comb* routine. That is, given  $\epsilon > 0$ , compute an  $\epsilon$ -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  $\text{OPT}_\epsilon(I)$  be the number of points in the smallest  $\epsilon$ -convex Pareto set for an instance  $I$ . Clearly, every algorithm that computes an  $\epsilon$ -convex Pareto set needs to make at least  $\text{OPT}_\epsilon(I)$  calls. The *performance (competitive) ratio* of an algorithm  $\mathcal{A}$  that computes a  $\epsilon$ -convex Pareto set using  $\mathcal{A}(I)$  calls for each instance  $I$  is  $\sup_I \frac{\mathcal{A}(I)}{\text{OPT}_\epsilon(I)}$ . 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 vari-

ous 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  $f_2 + \lambda f_1$ , where  $\lambda$  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  $\leq \epsilon$ , 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 we can determine from previous information that the maximum possible error in an interval is  $\leq \epsilon$  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  $\lambda$ . The length of the shortest path is a piecewise linear function of  $\lambda$  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 nonconvex 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  $\epsilon$ -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  $\epsilon$ : 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  $\epsilon$ -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  $\text{OPT}_\epsilon$  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  $\epsilon$ -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  $\epsilon$ , 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  $\epsilon$  (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  $\epsilon$  as possible, using as small a number  $N$  of calls as we can, ideally, as close as possible to the minimum number  $\text{OPT}_\epsilon(I)$  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  $\text{OPT}_\epsilon(I)$ ), 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  $\epsilon$ -convex Pareto set (i.e. one with  $\text{OPT}_\epsilon(I)$  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, its solution set can be post-processed to select a subset that is a  $\epsilon$ -convex Pareto set of size  $\leq 2\text{OPT}_\epsilon(I)$ .

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. Many proofs are deferred to the full version.

## 2 Model and Statement of Results

**2.1 Basic Definitions.** We describe the framework of a bi-objective optimization problem  $\Pi$  to which our results are applicable. Let  $x$  and  $y$  be the two objective functions (to be minimized, for simplicity) and  $\mathcal{I}$  be the objective space (the set of 2-vectors of objective values of the solutions). 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 optimization problems of interest, but also contains many continuous problems (e.g. convex programs, etc).

Let  $p, q \in \mathbb{R}_+^2$ . We say that  $p$  dominates  $q$  if  $p_i \leq q_i$ , for  $i = 1, 2$ . We say that  $p$   $\epsilon$ -covers  $q$  ( $\epsilon \geq 0$ ) if  $p_i \leq (1 + \epsilon)q_i$ , for  $i = 1, 2$ . The convex Pareto set of  $\mathcal{I}$ , denoted by  $\text{CP}(\mathcal{I})$ , is the minimum subset of  $\mathcal{I}$  whose convex combinations dominate  $\mathcal{I}$ . An  $\epsilon$ -convex Pareto set  $\text{CP}_\epsilon(\mathcal{I})$  of  $\mathcal{I}$  (henceforth  $\epsilon$ -CP) is a subset of  $\mathcal{I}$  whose convex combinations  $\epsilon$ -cover  $\mathcal{I}$ . By definition, the set  $S = \{s_i\}_{i=1}^k \subseteq \mathbb{R}_+^2$ ,  $x(s_i) < x(s_{i+1})$ , is an  $\epsilon$ -CP for  $\mathcal{I}$  if and only if the polygonal chain  $\langle s_1, \dots, s_k \rangle$   $\epsilon$ -covers  $\text{CP}(\mathcal{I})$ .

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

Let  $\Pi$  be a bi-objective optimization problem in the aforementioned framework. We access the objective space  $\mathcal{I}$  of  $\Pi$  via an oracle Comb that (exactly or approximately) minimizes non-negative linear combinations  $y + \lambda x$  of the objectives. We use the convention that, for  $\lambda = +\infty$ ,

we minimize the  $x$  objective. Let  $\delta$  be the accuracy of the Comb oracle. Then, for  $\lambda \in \mathbb{R}^+$ , we will denote by  $\text{Comb}_\delta(\lambda)$  a routine that returns a point  $q \in \mathcal{I}$  with the following property: Consider the line  $\ell(q, \lambda)$  through  $q$  with slope  $-\lambda$ , i.e.  $\ell(q, \lambda) = \{(x, y) \in \mathbb{R}^2 \mid y + \lambda x = y(q) + \lambda x(q)\}$ . Then there exists no solution point (in  $\mathcal{I}$ ) below the line  $(1 + \delta)^{-1} \cdot \ell(q, \lambda) \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{R}^2 \mid y + \lambda x = (y(q) + \lambda x(q))/(1 + \delta)\}$ . In other words, the  $\text{Comb}_\delta$  routine is an “approximate optimization oracle” for the objective space  $\mathcal{I}$ . Geometrically we “sweep” the space with a line of absolute slope  $\lambda$  until the line “hits”  $\mathcal{I}$ .

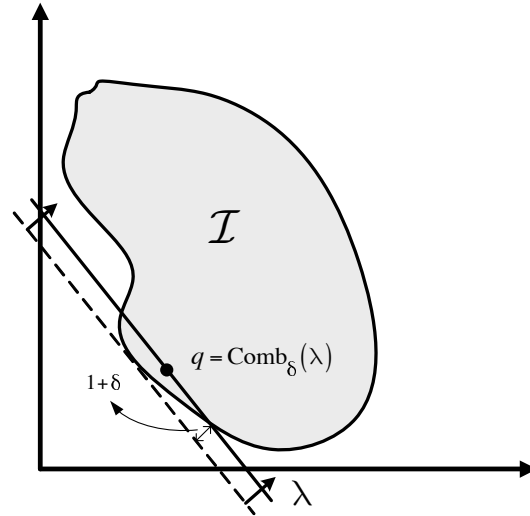


Figure 1: Illustration of  $\text{Comb}_\delta(\lambda)$  routine. The shaded region represents the (set of solution points in the) objective space  $\mathcal{I}$ . There exist no solution points below the dotted line.

We assume that either  $\delta = 0$  (i.e. we have an exact routine), or we have a PTAS, i.e. can efficiently compute  $\text{Comb}_\delta$  for all  $\delta > 0$ . The existence of a PTAS for  $\text{Comb}_\delta$  routine is necessary and sufficient for the efficient computability of an  $\epsilon$ -CP set [DY2].

**Notation.** We now introduce some notation used throughout the paper. For  $\delta = 0$ , i.e. when the optimization is exact, we omit the subscript and denote the Comb routine by  $\text{Comb}(\lambda)$ . We denote by  $\text{OPT}_\epsilon(\mathcal{I})$  the size of an optimum  $\epsilon$ -convex Pareto set for the given instance, i.e. an  $\epsilon$ -convex Pareto set with the minimum number of points. Note that, obviously every algorithm that constructs an  $\epsilon$ -CP, must certainly make at the very least  $\text{OPT}_\epsilon(\mathcal{I})$  calls to Comb, just to get  $\text{OPT}_\epsilon(\mathcal{I})$  points – which are needed at a minimum to form an  $\epsilon$ -CP ; this holds even if the algorithm somehow manages to always be lucky and call Comb with the right values of  $\lambda$  that identify the points of an optimal  $\epsilon$ -CP. (Having obtained the points of an optimal  $\epsilon$ -CP, another  $\text{OPT}_\epsilon(\mathcal{I})$  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  $\epsilon$ -CP. Hence, the “offline” optimum number of calls is at most  $2 \cdot \text{OPT}_\epsilon(\mathcal{I})$ .) Let  $\text{CHORD}_\epsilon(\mathcal{I})$  be the number of Comb calls required by the chord algorithm. The worst-case performance ratio of the algorithm is  $\sup_{\mathcal{I}} \text{CHORD}_\epsilon(\mathcal{I})/\text{OPT}_\epsilon(\mathcal{I})$ . If the inputs are drawn from some probability distribution  $\mathcal{D}$ , our measure will be the expected performance ratio  $\mathbb{E}_{\mathcal{I} \in \mathcal{D}}[\text{CHORD}_\epsilon(\mathcal{I})/\text{OPT}_\epsilon(\mathcal{I})]$  as a measure (note that we shall omit the subscript when it will be clear from the context).

We denote by  $pq$  the line segment with endpoints  $p$  and  $q$ ,  $(pq)$  denotes its length and  $\Delta(pqr)$  is the triangle defined by  $p, q, r$ . Also  $S(A)$  denotes the area of  $A$ . We now 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  $p$  to  $q$  is defined  $\Delta x(p, q) = \max\{x(q) - x(p), 0\}$ . The horizontal distance from  $p$  to the line segment  $\ell$  is  $\Delta x(p, \ell) = \Delta x(p, p_\ell)$ , where  $p_\ell$  is the  $y$ -projection of  $p$  on  $\ell$ . Also  $[n] := \{1, 2, \dots, n\}$  and  $[i, j] := \{i, i + 1, \dots, j\}$ .

REMARK 2.1. All the results of this paper on the performance of the Chord algorithm hold under the assumption that we have a PTAS for the Comb routine. However, for the simplicity of the exposition, we describe the algorithm and prove our upper and lower bounds for the case of an exact Comb routine. The case of an approximate routine is postponed for the full version.

**2.2 The Chord Algorithm.** We have set the stage to formally describe the algorithm. Let  $\Pi$  be a bi-objective problem with an efficient exact Comb routine. Given  $\epsilon > 0$  and an instance  $\mathcal{I}$  of  $\Pi$  (implicitly via Comb), we would like to construct an  $\epsilon$ -convex Pareto set 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 we describe the algorithm in detailed pseudo-code. The pseudo-code corresponds exactly to the description of the algorithm in the introduction.

The basic routine Chord is called recursively from the main algorithm. This recursive description will be useful in the analysis that follows.

An illustration of one iteration of the algorithm (i.e. recursive call of the Chord routine) is given in Figure 2 below. The points  $a_i, b_i$  are solution points (in  $\mathcal{I}$ ) and are the results of previous recursive calls. The point  $q_i = \text{Comb}(\lambda_{a_i b_i})$  is the solution point computed in the current call. The algorithm will recurse on the triangles  $\Delta(a_i a'_i q_i)$  and  $\Delta(q_i b'_i b_i)$ . Note that the line  $a'_i b'_i$  is parallel to  $a_i b_i$  and  $\angle a_i c_i b_i \in [\pi/2, \pi)$ .

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  $\text{CP}(\mathcal{I})$ . In particular,

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

$a = \text{Comb}(+\infty); b = \text{Comb}(0);$   
 $c = (x(a), y(b));$

**Return** 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  $lr; q = \text{Comb}(\lambda_{lr});$

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

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

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

$Q_l = \text{Chord}(\{l, q, s_l\}, \epsilon);$

$Q_r = \text{Chord}(\{q, r, s_r\}, \epsilon);$

**Return**  $Q_l \cup Q_r$ .

Table 1: Pseudo-code for Chord algorithm.

the computed points define a polygonal chain that is an “upper” approximation to  $\text{CP}(\mathcal{I})$  and the supporting lines at these points define a “lower” approximation.

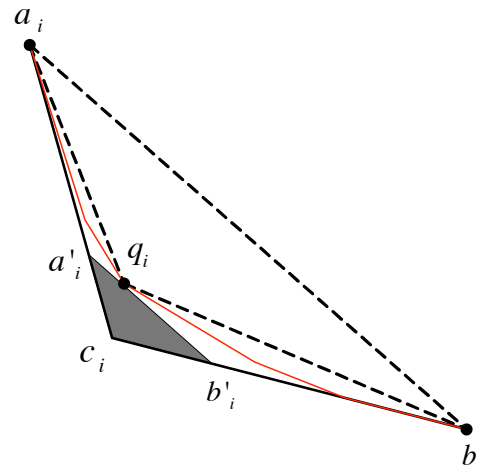


Figure 2: Illustration of the Chord algorithm.

The pseudo-code above 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).

Consider the recursion tree built by the Chord algorithm. In the analysis we use the following convention: There is no node in the tree if, at the corresponding step, the routine terminates without calling the Comb routine (i.e. if

$\mathcal{RD}(s, lr) \leq \epsilon$  in the above).

**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 2.1.** *The worst-case performance ratio of the chord algorithm (wrt the ratio distance) is  $\Theta\left(\frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\right)$ . Furthermore, no algorithm can have performance ratio better than  $\Omega(\log m + \log\log(1/\epsilon))$ .*

The lower bound is proved in Section 3.1 (Theorem 3.1). In Section 3.2 we show a slightly weaker upper bound of  $O(m + \log(1/\epsilon))$ ; this has the advantage that its proof is simple and intuitive. The proof of the asymptotically tight upper bound is more involved and is deferred to the full version. The lower bound against general algorithms is sketched in Section 3.1.2 (Theorem 3.2).

**REMARK 2.2.** 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  $\epsilon$ , is  $\Theta\left(\frac{\log(L/\epsilon)}{\log\log(L/\epsilon)}\right)$ . Furthermore, every algorithm has worst-case performance ratio at least  $\Omega(\log\log(L/\epsilon))$ .

We also analyze the Chord algorithm with respect to the horizontal distance metric (or by symmetry the vertical distance). We prove that in this setting the performance ratio of the algorithm is unbounded. In fact, we can prove a strong lower bound in this case: Any algorithm has an unbounded performance ratio (see Theorem 3.3).

Our second main result is a tight analysis of the Chord algorithm in an average case setting (wrt the ratio distance). Our random instances are drawn from two classes of 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 “un-concentrated” product distribution. We now formally define these distributions.

**DEFINITION 2.1.** A (spatial, homogeneous) Poisson Point Process (PPP) of intensity  $\lambda$  on a bounded subset  $S \subseteq \mathbb{R}^d$  is a collection of random variables  $\{N(A) \mid A \subseteq S \text{ is Lebesgue measurable}\}$  (representing the number of points occurring in every subset of  $S$ ), such that: (i) for any Lebesgue measurable  $A$ ,  $N(A)$  is a Poisson random variable with parameter  $\lambda \cdot S(A)$ ; (ii) for any collection of disjoint subsets  $A_1, \dots, A_k$  the random variables  $\{N(A_i), i \in [k]\}$  are mutually independent.

**DEFINITION 2.2.** Let  $S$  be some bounded Lebesgue-measurable subset of  $\mathbb{R}^2$ , and let  $\mathcal{D}$  be a distribution over  $S$ . The distribution  $\mathcal{D}$  is called  $\gamma$ -balanced,  $\gamma \in [0, 1)$ , if for all Lebesgue measurable subsets  $S' \subseteq S$ ,  $\mathcal{D}(S') \in \left[ (1 - \gamma) \cdot \mathcal{U}(S'), \frac{\mathcal{U}(S')}{(1 - \gamma)} \right]$ , where  $\mathcal{U}$  is the uniform distribution over  $S$ .

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

**THEOREM 2.2.** *For the aforementioned classes of random instances, the expected performance ratio of the Chord algorithm is  $\Theta(\log m + \log\log(1/\epsilon))$ .*

The upper bound can be found in Section 4.1 and the lower bound in Section 4.2. We present detailed proofs for the case of PPP. The case of Product distributions is only sketched; detailed proofs will appear in the full version.

We note that similar results apply also for the Hausdorff distance.

## 3 Worst-Case Analysis

**3.1 Lower Bounds.** In this section we prove the aforementioned lower bounds. In Section 3.1.1 we prove a tight lower bound for the chord algorithm. In Section 3.1.2 we sketch the proof of our general lower bounds.

**3.1.1 Lower Bound for Chord Algorithm.** In fact, we prove a stronger result that also rules out the possibility of constant factor bi-criteria approximations, i.e. the lower bound applies even if the algorithm is allowed error  $O(\epsilon)$  and compare against the optimal  $\epsilon$ -approximation.

**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  $\text{OPT}_\epsilon(\mathcal{I}_{LB}) = O(1)$  and  $\text{CHORD}_{\mu\epsilon}(\mathcal{I}_{LB}) = \Omega\left(\frac{1}{\mu} \cdot \frac{m+\log(1/\epsilon)}{\log m+\log\log(1/\epsilon)}\right)$ .*

*Proof.* 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  $(ac) \gg (bc)$ . For such instances one can force the algorithm to select many “redundant” points (hence, perform many calls to Comb) to guarantee an  $\epsilon$ -approximation, 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  $\epsilon$  is sufficiently small. Initially, the only available information is that the instance lies in the right triangle  $\triangle(acb)$ , where  $c = (1, 1)$ . Observe that, for an instance

with these endpoints, the original error (i.e.  $\mathcal{RD}(c, ab)$ ) is roughly  $2\epsilon$  and one intermediate point  $q^*$  always suffices to  $\epsilon$ -cover, i.e. the optimal size is (at most) 3. We want to define a sequence of points  $\{q_1, \dots, q_j\}$  – all of which will be vertices of the convex pareto set for the corresponding instance – that force the algorithm to select *all* the  $q_i$ 's (in order of increasing  $i$ ), until it finds  $q^* = q_j$ . That is, we want the algorithm to monotonically converge to the optimal point by visiting *all* the vertices of the instance in order.

Let  $\lambda_{ab} = 1/(2\epsilon)$  be the slope of  $ab$ . In the first step, the algorithm calls  $\text{Comb}(\lambda_{ab})$  to find a solution point at maximum ratio distance from  $ab$ . We want this call to return  $q_1$ . The idea is to subdivide the (length of the) edge  $ac$  geometrically with ratio  $k$  – for an appropriately selected  $k$  – and place  $q_1$  on  $ac$  so that  $(q_1c) = (ac)/k$ . Let  $\ell(q_1)$  be the line parallel to  $ab$  through  $q_1$ . By definition, this line *supports* the objective space. Denote  $q_1^* = \ell(q_1) \cap bc$ . Then the error of the approximation  $\{a, q_1, b\}$  equals  $\mathcal{RD}(q_1^*, q_1b)$  (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  $\mathcal{RD}(q_1^*, q_1b) \leq \epsilon$ . On the other hand, if  $k \geq 1/\epsilon$ , we are also done since  $y(q_1) \leq 1 + \epsilon$ , hence  $\mathcal{RD}(q_1^*, q_1b) \leq \epsilon$ . If  $\omega(1) \leq k \leq o(1/\epsilon)$ , it can be argued that  $\mathcal{RD}(q_1^*, q_1b) \approx (q_1^*b) = 2\epsilon \cdot (1 - 1/k)$ . (Observe that the RHS is actually the horizontal distance of  $q_1^*$  from  $q_1b$ .) Hence, for this regime the error decreased by only an additive  $2\epsilon/k \ll \epsilon$  and the algorithm needs to recurse on the triangle  $\Delta(q_1q_1^*b)$ . Note that  $\lambda_{q_1b} = \lambda_{ab}/k = (2\epsilon)^{-1}/k$ . The algorithm now calls  $\text{Comb}(\lambda_{q_1b})$  and this call will return  $q_2$ . To select this point we repeat our “geometric subdivision trick”. Recall that there are no points below the line  $q_1q_1^*$ . Let  $q_2'$  be the projection of  $q_2$  on  $ac$ . We select  $q_2$  on the segment  $q_1q_1^*$  so that  $(q_2'c) = (q_1c)/k$ . Let  $\ell(q_2)$  be the (supporting) line parallel to  $q_1b$  through  $q_2$ . Similarly, the error of the approximation  $\{a, q_1, q_2, b\}$  equals  $\mathcal{RD}(q_2^*, q_2b)$  where  $q_2^* = \ell(q_2) \cap bc$ . Now, if  $(q_2'c) = (ac)/k^2 \gg \epsilon$ , we can approximate  $\mathcal{RD}(q_2^*, q_2b) \approx (q_2^*b) \approx 2\epsilon \cdot (1 - 2/k)$ , i.e. the error has decreased by another additive  $2\epsilon/k$ . 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  $(q_i'c) = (q_{i-1}'c)/k$ . The iterative process can continue, as long as  $(q_i'c) \gg \epsilon$ . 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  $\{q_1, \dots, q_j\}$  until it covers by  $\mu \cdot \epsilon$ , while the last point  $q_{j+1}$  (along with the endpoints) suffice to  $\epsilon$ -cover.

The formal proof comes in two steps. We first analyze

the chord algorithm wrt to the horizontal distance metric. We show that the performance ratio of the chord algorithm is unbounded in this setting (this also holds for the vertical distance by symmetry). In particular, for every  $k \in \mathbb{N}$ , there exists an instance  $\mathcal{I}_G$  (actually in the unit square) so that the chord algorithm has ratio  $k$  for additive error  $1/2$ . 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 for the ratio distance.

**Step 1:** The instance  $\mathcal{I}_G(H, L, k, j)$  lies in the triangle  $\Delta(abc)$ , 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 in  $Q$ . We set  $q_0 = a$  and  $q_{j+2} = b$ . The set of points  $\{q_1, \dots, q_{j+1}\}$  is defined recursively as follows:

1. The point  $q_1$  has  $x(q_1) = x(a)$  and  $y(q_1) = y(c) + (y(a) - y(c))/k$ .
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 having  $y(q_i) = y(c) + (y(q_{i-1}) - y(c))/(k + i - 1)$ .

We want to compute an  $\epsilon_L$ -approximation – recall that the error is measured wrt the horizontal distance – with  $\epsilon_L(L, k, j) \stackrel{\text{def}}{=} L \cdot \frac{k-1}{k+j-1}$ . Also denote  $\epsilon'_L(L, k, j) \stackrel{\text{def}}{=} L \cdot \frac{k-1}{k} \cdot \frac{j}{k+j-1}$ . Note that  $\epsilon'_L/\epsilon_L = j/k$ .

We show the following:

**LEMMA 3.1.** *The chord algorithm applied to the instance  $\mathcal{I}_G$  (wrt horizontal distance) selects the sequence of points  $\langle q_1, q_2, \dots, q_j \rangle$  to get error  $\epsilon_L$ , while the set  $\{a, q_{j+1}, b\}$  attains error  $\epsilon'_L$ .*

*Sketch of 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 triangle similarity 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 to noting that the error of the approximation  $\{q_1, \dots, q_i\}$  is  $\Delta x(q_i^*, q_ib)$ , which is  $> \epsilon_L$  for  $i < j$  and  $= \epsilon_L$  for  $i = j$ . The details of the proof are deferred to the full version. ■

**Step 2:** We now define the instance  $\mathcal{I}_{LB}$ . Fix  $H^* := 2^m - 1$ ,  $L^* := (\mu + 1) \cdot \epsilon$ ,  $j^* := \Theta\left(\frac{1}{\mu} \cdot \frac{\log(H^*/\epsilon)}{\log \log(H^*/\epsilon)}\right)$  and  $k^* := \mu \cdot j^* + 1$ . We set  $\mathcal{I}_{LB}(\epsilon, m, \mu) := \mathcal{I}_G(H^*, L^*, k^*, j^*)$ . Also, let  $\epsilon_{L^*} := \epsilon_L(L^*, k^*, j^*)$  and  $\epsilon'_{L^*} := \epsilon'_L(L^*, k^*, j^*)$ . Observe that, under this choice of parameters, we have  $\epsilon_{L^*} \geq \mu \cdot \epsilon$  and  $\epsilon'_{L^*} < \epsilon$ . We show:

LEMMA 3.2. *The chord algorithm applied to  $\mathcal{I}_{LB}$  selects (a superset of) the points  $\{q_1, \dots, q_{j^*/8}\}$  to attain ratio distance  $\epsilon_{L^*}$ , while the set  $\{a, q_{j^*+1}, b\}$  is an  $\epsilon'_{L^*}$ -convex Pareto set.*

*Sketch of Proof:* The proof amounts to proving that for the particular instance, the horizontal distance is a very good approximation to the ratio distance. Hence, the behavior of the algorithm in these metrics is similar. The reason we lose a constant factor in the number of points (i.e.  $j^*/8$  as opposed to  $j^*$ ) is due to the error term in the approximation between the metrics. We omit the details. ■

This completes the proof of Theorem 3.1. ■

**3.1.2 General Lower Bounds.** We can prove a (weaker) lower bound against any algorithm that rules out the possibility of a constant factor approximation for the problem. In particular, we have

THEOREM 3.2. *For any algorithm (even randomized), there exists an instance such that the ratio of the algorithm is  $\Omega(\log m + \log \log(1/\epsilon))$ .*

*Sketch of Proof:* The proof uses the construction of the previous subsection as a black box. It works by essentially reducing the computation of an  $\epsilon$ -CP (given access to Comb) to a comparison-based binary search on a set of cardinality  $j^*$ .

Recall that we consider algorithms that have access to Comb and measure the number of calls made by the algorithm as compared to the minimum possible for the given instance. The proof uses the lower bound  $\mathcal{I}_{LB}$  from the previous theorem essentially as a black box. For simplicity, let  $\mu = 1$  and  $Q = \{q_1, \dots, q_{j^*}\}$  be the corresponding set of points. Suppose the error we care about is  $\epsilon > \mathcal{RD}(q_{j^*}^*, q_{j^*} b)$ . Define the family of “prefixes”  $\mathcal{Q} = \{Q_i\}_{i=1}^{j^*}$  of  $Q$ , where  $Q_i = \{q_1, \dots, q_i\}$ ,  $i \in [j^*]$ . It follows from the properties of the set  $Q$  that the convex polygonal instance corresponding to each  $Q_i$  has the following property: its last point (i.e.  $q_i$ ) suffices to  $\epsilon$ -cover (together with the endpoints), while the set  $Q_i \setminus q_i$  does not.

Consider a general algorithm  $\mathcal{A}$ . In order to find an  $\epsilon$ -convex Pareto, it must be able to “distinguish” among  $Q_i$ ’s. Since otherwise, it cannot detect the rightmost point of the given instance, hence cannot guarantee an  $\epsilon$ -approximation. This essentially means that the algorithm must perform a binary search on the slopes  $\{\lambda_{q_i b}\}_{i=1}^{j^*}$ . Having reduced the problem to a comparison-based binary search on a set

of cardinality  $j^*$  a lower bound of  $\Omega(\log j^*)$  follows (for randomized algorithms as well). ■

Our next theorem says that, if our metric is the horizontal/vertical distance, then every algorithm with access to a Comb routine has an unbounded performance ratio. This holds even for instances that lie in the unit square and for additive error  $1/2$ . The proof of the following theorem builds on the lower bound construction for the chord algorithm (Section 3.1.1, Step 1). The details will be in the full version.

THEOREM 3.3. *For any  $k \in \mathbb{N}$  and any algorithm  $\mathcal{A}$ , there exists an instance for which the performance ratio of  $\mathcal{A}$  is  $\Omega(k)$ . This is true even for instances that lie in the unit square and for error (horizontal distance)  $1/2$ .*

**3.2 Upper Bound.** In this section we prove that the performance ratio of the chord algorithm wrt the ratio distance is  $O(m + \log(1/\epsilon))$ . We do this for the sake of the exposition; the proof of the tight upper bound is more involved and will appear in the full paper.

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 subdivision. 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  $2\epsilon$  and decreases by an additive  $2\epsilon/k$  in every iteration, where  $k \approx \log(1/\epsilon)/\log \log(1/\epsilon)$ . 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 (given access to Comb), as follows from our general lower bound (Theorem 3.2)<sup>†</sup>.

Our approach is somewhat indirect. We prove that the area between the “upper and lower approximation” decreases by a constant factor in every step of the algorithm. This can be interpreted as a “potential function type argument”. It is not hard to argue that, when the area 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  $\epsilon$ . We then use a simple charging argument to show that this suffices to yield the desired performance guarantee. Formally, we prove:

THEOREM 3.4. *Let  $T_1$  be the triangle at the root of the chord algorithm’s recursion tree and denote  $\alpha \stackrel{\text{def}}{=} \dots$*

<sup>†</sup>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.

$\min\{x(q), y(q) \mid q \in T_1\}$ . The chord algorithm finds an  $\epsilon$ -convex Pareto set after at most  $O\left(\log(S(T_1)/S_0)\right) \cdot \text{OPT}_\epsilon$  calls to Comb, where  $S_0 \stackrel{\text{def}}{=} \epsilon^2 \cdot \alpha^2/2$ .

The desired result follows from the above, since  $T_1 \subseteq [2^{-m}, 2^m]$ , which implies  $S(T_1) \leq 2^{2m}$  and  $\alpha \geq 2^{-m}$ .

*Proof.* The first thing one needs to argue is that, upon termination, the algorithm has found an  $\epsilon$ -CP set. This can be shown by an easy induction:

LEMMA 3.3. *The set of points  $Q$  computed by the algorithm is an  $\epsilon$ -convex Pareto set.*

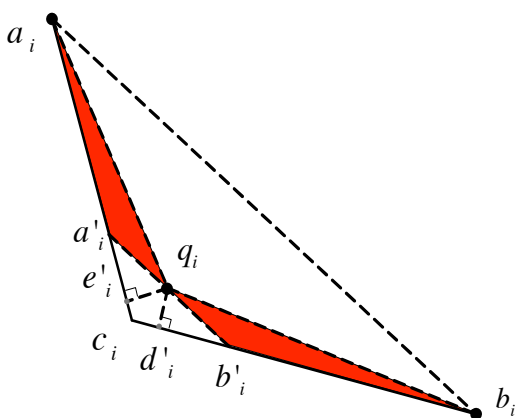


Figure 3: Illustration of the area shrinkage property of the Chord algorithm.

To upper bound the performance ratio we need a few lemmas. Our first lemma quantifies the area shrinkage property. We remark that this is a statement independent of  $\epsilon$ .

LEMMA 3.4. *Let  $T_i = \Delta(a_i b_i c_i)$  be the triangle processed by the chord algorithm at some recursive step. Denote  $q_i = \text{Comb}(\lambda_{a_i b_i})$ . Let  $T_{i,l} = \Delta(a_i a'_i q_i)$ ,  $T_{i,r} = \Delta(b_i b'_i q_i)$  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 3 for an illustration. Let

$$y = (a'_i c_i)/(a_i c_i) = (b'_i c_i)/(b_i c_i) \in (0, 1).$$

Then,

$$S(\Delta(a'_i b'_i c_i)) = y^2 S(T_i).$$

We have

$$\begin{aligned} S(T_{i,l}) + S(T_{i,r}) &= ((a_i a'_i) \cdot (q_i e'_i) + (b_i b'_i) \cdot (q_i d'_i))/2 \\ &= (1-y) \cdot ((a_i c_i) \cdot (q_i e'_i) + (b_i c_i) \cdot (q_i d'_i))/2 \\ &= (1-y) \cdot (S(\Delta(a_i c_i q_i)) + S(\Delta(b_i c_i q_i))) \\ &= (1-y) \cdot (S(T_{i,l}) + S(T_{i,r}) + S(\Delta(a'_i b'_i c_i))) \end{aligned}$$

which gives

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

as desired.  $\blacksquare$

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

LEMMA 3.5. *Consider the recursion tree  $\mathcal{T}^\ddagger$  built by the algorithm and let  $\mathcal{L}'$  be the number of lowest non-leaf nodes. Then  $\text{OPT}_\epsilon \geq |\mathcal{L}'|$ .*

*Proof.* Each such node in the tree corresponds to a triangle  $T_i = \Delta(a_i b_i c_i)$  with the property that the ratio distance of the convex Pareto set from the segment  $a_i b_i$  is strictly greater than  $\epsilon$  (o/w the node would be a leaf). Hence, each such triangle must contain a point of the optimal  $\epsilon$ -convex Pareto set. Since all these triangles are disjoint, the result follows.  $\blacksquare$

Finally, we need a lemma that relates the ratio distance within a triangle to its area:

LEMMA 3.6. *Consider a triangle  $T_i = \Delta(a_i b_i c_i)$  such that  $T_i \subseteq T_1$ . If  $S(T_i) \leq \epsilon^2 \cdot \alpha^2/2$ , then  $\mathcal{RD}(c_i, a_i b_i) \leq \epsilon$ .*

The above lemma follows essentially by observing that the worst-case for the area-error trade-off is when  $c_i = (\alpha, \alpha)$ , and the triangle is right and isosceles (i.e.  $(a_i c_i) = (a_i b_i)$ ).

Now we have all the tools we need prove the theorem. First, by Lemma 3.4, when a node of the tree is at depth  $\log(S(T_1)/S_0)$ , it will have area at most  $S_0$ . By Lemma 3.6, this implies that the depth of the tree  $\mathcal{T}$  is  $O(\log(S(T_1)/S_0))$ . Now, Lemma 3.5 implies that  $\text{CHORD}_\epsilon \leq O(\log(S(T_1)/S_0)) \cdot \text{OPT}_\epsilon$ , which concludes the proof.  $\blacksquare$

REMARK 3.1. We briefly sketch the differences for the case of an approximate Comb routine. First we note that, in this case, the description of the Chord algorithm (Table 1) has to be slightly modified to guarantee that the set of computed points is an  $\epsilon$ -CP. In particular, in the Chord routine, we need to check whether  $\mathcal{RD}(q, lr) \leq \epsilon'$  for some  $\epsilon' < \epsilon$ . As a consequence, to prove the desired upper

$\ddagger$ We use the following convention: There is no node in the tree if, for a triangle, the routine terminates without calling the Comb routine.

bound, in addition to the above lemmas we need a way to relate  $\text{OPT}_{\epsilon'}$  and  $\text{OPT}_{\epsilon}$ . This is provided to us by a planar geometric lemma from [DY2] that roughly says that as  $\epsilon$  decreases, the size of the optimal  $\epsilon$ -CP (for the same instance) does not increase too fast.

## 4 Average Case Analysis

In this section we prove our average case results.

**4.1 Upper Bounds.** We present only the case of the Poisson point process in the body of the paper. The proofs for unconcentrated product distributions are only sketched and will appear in the full version. The analysis for both cases has the same overall structure, however each case has its difficulties, as elaborated below.

**Overview of the proof.** We now give a brief overview of the proof. 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 product distributions. As in the worst-case, we resort to an indirect measure of the algorithm's progress, namely the area of the triangles maintained in the algorithm's recursive tree. We think that this feature of our analysis is quite interesting and indicates that this measure is quite robust.

We first show (see Lemma 4.1 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 an exponential decrease) with high probability. It follows then 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 competitive ratio is  $\log \log n$  in this case.

To analyze the expected competitive ratio in the complementary event, we break it into a "good" event, under which the competitive ratio is  $\log n$  with high probability, and a "bad" event, where the competitive ratio is potentially unbounded (in the Poisson case) or at most  $n$  (for the case of product distributions). The potential unboundedness of the competitive ratio in the Poisson case creates complications in bounding the expected competitive ratio of the algorithm over the full space. We overcome this difficulty by bounding the upper tail of the Poisson distribution (see Lemma 4.6).

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 harder. 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. Consider for example the uniform distribution: we select  $n$  independent random points, each uniformly distributed in  $[2^{-m}, 2^m]^2$ . We are given an  $\epsilon > 0$  and we run the chord algorithm on this random instance. It is easy to see that, if  $n$  is larger than some  $\text{poly}(2^m/\epsilon)$ , then the algorithm makes a constant number of calls in expectation. Hence, we can assume wlog that  $n = O(\text{poly}(2^m/\epsilon))$ . A similar bound also holds for the intensity  $\lambda$  of the PPP.

**4.1.1 Poisson Point Process.** We prove the following theorem – which combined with the aforementioned discussion yields the desired upper bound of  $O(\log m + \log \log(1/\epsilon))$ .

**THEOREM 4.1.** *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  $\lambda$ ,  $\lambda S(T_1) > c > e$ , where  $c$  is an absolute constant. The expected performance ratio of the Chord algorithm on this instance is  $O\left(\log \log(\lambda S(T_1))\right)$ .*

*Proof.* Let us denote  $S_1 = S(T_1)$ . We can assume that  $\lambda S_1 > 10$ , since otherwise the expected total number of points inside  $T_1$  is  $O(1)$  and the bound trivially holds. We show next that the area of the triangles maintained by the algorithm decreases rapidly at every recursive step. Namely,

**LEMMA 4.1.** *Let  $T_i = \Delta(a_i b_i c_i)$  be the triangle processed by the chord algorithm at some recursive step. Denote  $q_i = \text{Comb}(\lambda_{a_i b_i})$ . Let  $T_{i,l} = \Delta(a_i a'_i q_i)$  and  $T_{i,r} = \Delta(b_i b'_i q_i)$ . For all  $c > 0$ , with probability at least  $1 - \frac{1}{(\ln \lambda S_1)^c}$  conditioning on the information available to the algorithm,*

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

*Proof.* It follows from the properties of the chord algorithm that, before the Comb routine on input  $T_i$  is invoked, the following information is known to the algorithm, conditioning on the history:

- there exist solution points at the locations  $q_j$ , for all  $j = 1, 2, \dots, i - 1$ ;
- there is no point to the left of (below) the line defined by  $a_j$  and  $c_j$ , for all  $j = 1, 2, \dots, i$ ;
- there is no point below the line defined by  $c_j$  and  $b_j$ , for all  $j = 1, 2, \dots, i$ ;

See Figure 4 for an illustration. It follows from the properties of the Poisson Point Process that, conditioned on the above information, the number of points in a triangle of area  $S$  inside  $T_i$  follows a Poisson distribution with

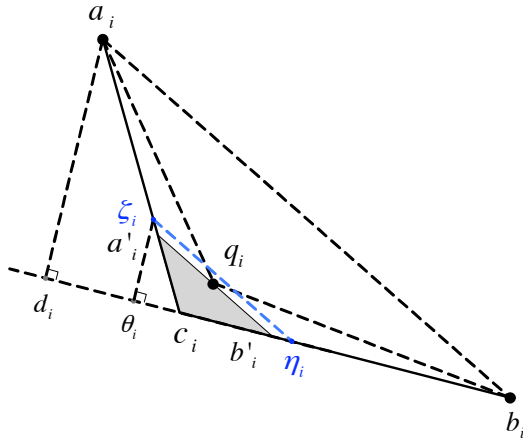


Figure 4: Illustration of the average case area shrinkage property of the algorithm.

parameter  $\lambda \cdot S$ . 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  $S^* \stackrel{\text{def}}{=} \frac{c_i \ln \ln \lambda S_1}{\lambda}$ , it follows that, with probability at least  $1 - \frac{1}{(\ln \lambda S_1)^c}$ , the point  $q_i$  is contained in the triangle  $T^*$ . We bound from above the area of  $T_{i,l}$  by the area of  $T'_{i,l} = \triangle(a_i c_i \eta_i)$  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,

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

Finally,

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

This concludes the proof of the lemma.  $\blacksquare$

Let us choose  $c \in \left( \frac{1}{\ln \ln \lambda S_1}, \frac{\lambda S_1}{\ln \ln \lambda S_1} \right)$ , and let  $S(T)$  be the area of a triangle  $T$  maintained by the algorithm at depth  $d$  of the recursion. It follows from Lemma 4.1 that, with probability at least  $1 - \frac{d}{(\ln \lambda S_1)^c}$ ,

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

where to bound the probability of the above event we have taken a union bound only over the events on the path of the recursion tree connecting  $T$  to the root of the recursion. Now consider the top  $d^* := \lceil \log_2 \ln \lambda S_1 \rceil$  levels of the recursion tree of the algorithm. The tree has at most  $2 \cdot \ln \lambda S_1$  internal nodes. Notice that the tree in general has many internal nodes with out-degree 1. Intuitively, for each such node the algorithm performs a redundant call to Comb. Using Lemma 4.1 and a union bound it follows that, with overall probability at least  $1 - \frac{2 \cdot \ln \lambda S_1}{(\ln \lambda S_1)^c}$ , the area of every triangle at depth  $d^*$  of the recursion tree is at most  $S^{**} := (e \cdot c \cdot \ln \ln \lambda S_1) / \lambda$ , 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  $S^{**}$ . We just argued that the probability of the event  $\mathcal{A}$  is at least  $1 - \frac{2}{(\ln \lambda S_1)^{c-1}}$ . We now show the following:

LEMMA 4.2. *Conditioning on the event  $\mathcal{A}$ , the expected performance ratio of the algorithm is  $O(\log \log \lambda S_1)$ .*

*Proof.* Let  $\mathcal{T}$  be the recursion tree of the algorithm pruned at level  $d^*$ , let  $\mathcal{V}$  be the set of nodes of  $\mathcal{T}^\ddagger$ , and let  $\mathcal{L}_{d^*}$  be the subset of nodes in  $\mathcal{V}$  that lie at depth  $d^*$  from the root. (Note that the set  $\mathcal{L}_{d^*}$  is a subset of the leaves of  $\mathcal{T}$ .) 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  $T$ . We denote by  $\mathcal{L}'$  the set of lowest non-leaf nodes of the tree  $\mathcal{T}$ . As in the worst-case analysis, we have  $\text{OPT}_\epsilon \geq |\mathcal{L}'|$ . Also note that  $|\mathcal{V}| \leq 3d^* |\mathcal{L}'|$  and that the Chord algorithm makes a call to Comb for every node in the tree.

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  $\mathcal{A}$ . Moreover, conditioning on the information  $\mathcal{F}$ , for all  $T \in \mathcal{L}_{d^*}$ ,  $X_T$  follows a Poisson distribution with parameter  $\lambda \cdot S(T)$ . So, given that the event  $\mathcal{A}$  holds, we have  $\mathbb{E}[X_T | \mathcal{F}] \leq \lambda \cdot S^{**}$ .

Also recall that the number of Comb calls 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 calls performed by the algorithm is at most

$$\begin{aligned} \mathbb{E}[\text{CHORD}_\epsilon | \mathcal{F}] &\leq |\mathcal{V}| + 2 \cdot \sum_{T \in \mathcal{L}_{d^*}} \mathbb{E}[X_T | \mathcal{F}] \\ &\leq |\mathcal{V}| + 2 |\mathcal{L}_{d^*}| \cdot \lambda \cdot S^{**} \\ &\leq |\mathcal{V}| + 4 |\mathcal{L}'| \cdot \lambda \cdot S^{**} \\ &\leq |\mathcal{L}'| \cdot (3d^* + 4\lambda \cdot S^{**}), \end{aligned}$$

<sup>‡</sup>We remind the reader that by convention we do not have a node in the tree for those triangles  $T = \triangle(a_i b_i c_i)$  for which  $\mathcal{RD}(c_i, a_i b_i) \leq \epsilon$ .

where we made use of the fact that  $|\mathcal{L}_{d^*}| \leq 2|\mathcal{L}'|$ . So, conditioning on the information  $\mathcal{F}$  available to the algorithm in the first  $d^*$  levels of its recursion tree (before the processing of any triangle in the  $d^*$ -th level), the expected performance ratio of the algorithm is

$$\begin{aligned} \mathbb{E} \left[ \frac{\text{CHORD}_\epsilon}{\text{OPT}_\epsilon} \middle| \mathcal{F} \right] &\leq \mathbb{E} \left[ \frac{\text{CHORD}_\epsilon}{|\mathcal{L}'|} \middle| \mathcal{F} \right] \\ &\leq (3d^* + 4\lambda \cdot S^{**}) \\ &= O(\log \log \lambda S_1). \end{aligned}$$

Integrating over all possible  $\mathcal{F}$  inside  $\mathcal{A}$  concludes the proof of the lemma.  $\blacksquare$

From Lemma 4.2 we have that

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

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

**LEMMA 4.3.** For  $c \in \left( \frac{1}{\ln \lambda S_1}, \frac{\lambda S_1}{\ln \lambda S_1} \right)$ , there exists an event  $\mathcal{A}$ , with  $\Pr[\mathcal{A}] \geq 1 - \frac{2}{(\ln \lambda S_1)^{c-1}}$ , such that the expected performance ratio of the algorithm conditioning on  $\mathcal{A}$  is  $O(\log \log \lambda S_1)$ .

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

**LEMMA 4.4.** For  $c' \in \left( \frac{1}{\ln \lambda S_1}, \frac{\lambda S_1}{\ln \lambda S_1} \right)$ , there exists an event  $\mathcal{B}$ , with  $\Pr[\mathcal{B}] \geq 1 - \frac{2}{(\lambda S_1)^{c'-1}}$ , such that the expected performance ratio of the algorithm conditioning on  $\mathcal{B}$  is  $O(\log \lambda S_1)$ .

Finally, the performance ratio can be bounded by twice the total number of points in the triangle at the root of the recursion tree. Hence,

**LEMMA 4.5.** The expected performance ratio of the algorithm is  $O(\lambda S_1)$ .

We want to use Lemmas 4.3, 4.4 and 4.5 to deduce that the expected performance ratio of the algorithm is  $O(\log \log \lambda S_1)$ . 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 4.3. It is easy to see that conditioned on  $\mathcal{C}$  the expected performance ratio of the algorithm can still be bounded by  $O(\log \lambda S_1)$ , since this expectation is affected only by whatever happens at level  $\lceil \log_2 \lambda S_1 \rceil$  of the recursion tree and below. On the other hand, using the fact that  $\Pr[\mathcal{A}] \geq 1 - \frac{2}{(\ln \lambda S_1)^{c-1}}$ , it follows that

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

Now we can bound the expectation of the performance ratio as follows:

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

To conclude, we need to bound the last term of the above expression. Note first that  $\mathcal{B} \subseteq \mathcal{A} \cup \mathcal{C}$ . Hence,

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

We again use the (trivial) fact that 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 random variable with parameter  $\lambda \cdot S_1$ . Hence, we have

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

To bound the right hand side of the above we use the following technical lemma.

**LEMMA 4.6.** Let  $X$  be a Poisson( $\lambda$ ) random variable, with  $\lambda \geq 1$ , and let  $\mathcal{E}$  be some event. Then

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

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

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

Now we distinguish two cases. If  $k^* - 1 \geq 2\lambda^2$ , then from Chebyshev's inequality we get:

$$\Pr[X \geq k^* - 1] \leq \frac{1}{\lambda^2}.$$

Hence,

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

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

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

and

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

Hence,

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

This concludes the proof of the lemma. ■

From Lemma 4.6 we obtain

$$\begin{aligned} &\mathbb{E} \left[ \frac{\text{CHORD}_\epsilon}{\text{OPT}_\epsilon} \mid \overline{\mathcal{AUC}} \right] \cdot \Pr[\overline{\mathcal{AUC}}] \leq \\ &\leq \max \left\{ \frac{1}{\lambda S_1}, O((\lambda S_1)^3) \cdot \Pr[\overline{\mathcal{AUC}}] \right\} \\ &\leq \max \left\{ \frac{1}{\lambda S_1}, O((\lambda S_1)^3) \frac{2}{(\lambda S_1)^{c'-1}} \right\}. \end{aligned}$$

Choosing  $c' = 4$ , the above RHS becomes  $O(1)$ . Plugging this into (4.1) with  $c = 2$  gives

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

This concludes the proof of Theorem 4.1. ■

**4.1.2 Product Distributions.** We now state our theorem for product distributions.

**THEOREM 4.2.** *Let  $n$  points be chosen independently from a  $\gamma$ -balanced distribution on  $S_1$ , where  $S_1$  is the triangle at the root of the chord algorithm's recursion tree, and  $\gamma \in [0, 1)$  some constant. The expected performance ratio of the algorithm on this instance is  $O_\gamma(\log \log n)$ .*

The proof has the same overall structure as the proof of Theorem 4.1, but the details are more elaborate. The proof will appear in the full version.

**4.2 Lower Bounds.** We prove lower bounds on the expected performance ratio of the algorithm that match our upper bounds. For the case of the PPP we prove

**THEOREM 4.3.** *There exists a family of instances for which the expected performance ratio of the Chord algorithm is  $\Omega(\log \log \lambda S_1)$ , where  $S_1$  is the area of the triangle at the root of the recursion tree of the algorithm and  $\lambda$  is the intensity of the Poisson Point Process. In particular, we can select the parameters so that  $\lambda S_1 = 2^m/\epsilon$ , which yields a lower bound of  $\Omega(\log m + \log \log 1/\epsilon)$ .*

In analogy to the worst-case construction, the hard instances for the average case setting are “skewed”. Further details are deferred to the full version. Similar lower bounds can also be shown for the case of the uniform 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 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, 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, 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, Leipzig, pp. 261-315, 1913.
- [BHR] R. E. Burkard and 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, 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, T. R. Bortfeld. Approximating convex Pareto surfaces in multiobjective radiotherapy planning. *Med. Phys.*, 33, pp. 3399-3407, 2006.
- [CY] R. Cole, C. Yap. Shape from probing. *J. Algorithms* 8, pp. 19-38, 1987.
- [DP] D. Douglas, 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, M. Yannakakis. Small Approximate Pareto Sets for Bi-objective Shortest Paths and Other Problems. To appear in *SIAM J. on Computing*, 2009.

- [DY2] I. Diakonikolas, M. Yannakakis. Succinct Approximate Convex pareto Curves. *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pp. 74-83, 2008.
- [Ehr] M. Ehrgott. *Multicriteria optimization*, 2nd edition, Springer-Verlag, 2005.
- [EG] M. Ehrgott, X. Gandibleux. An annotated bibliography of multiobjective combinatorial optimization problems. *OR Spectrum* 42, pp. 425-460, 2000.
- [ES] M. J. Eisner, 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, 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. Proc. 41st IEEE Symp. on Foundations of Computer Science, 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, 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, L. Vandenberghe. Trade-off Curves for QoS Routing. In *Proc. INFOCOM*, 2006.
- [VY] S. Vassilvitskii, 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, C. Goh. A method for convex curve approximation. *European J. of Oper. Res.* 97, pp. 205-212, 1997.