A SUBEXPONENTIAL ALGORITHM FOR ABSTRACT OPTIMIZATION PROBLEMS

BERND GÄRTNER *

Abstract. An *Abstract Optimization Problem* (AOP) is a triple $(H, <, \Phi)$ where H is a finite set, $<$ a total order on 2^H and Φ an oracle that, for given $F \subseteq G \subseteq H$, either reports that $F = \min_<\{F' \mid F' \subseteq G\}$ or returns a set $F' \subseteq G$ with $F' < F$. To solve the problem means to find the minimum set in H. We present a randomized algorithm that solves any AOP with an expected number of at most

$$e^{2\sqrt{n}+O(\sqrt[4]{n}\ln n)}$$

oracle calls, $n = |H|$. In contrast, any deterministic algorithm needs to make $2^n - 1$ oracle calls in the worst case.

The algorithm is applied to the problem of finding the distance between two n-vertex (or n-facet) convex polyhedra in d-space, and to the computation of the smallest ball containing n points in d-space; for both problems we give the first subexponential bounds in the arithmetic model of computation.

Key words. computational geometry, smallest enclosing ball, distance between convex polyhedra, local optimization, randomized algorithm

AMS subject classifications. 68Q20, 68Q25, 68U05, 90C25, 90C27

1. Introduction.

Three geometric optimization problems. Recently, Sharir & Welzl [23] have described an abstract class of problems – so-called *LP-type problems* – that are efficiently solvable by a simple randomized algorithm. The typical LP-type problems are geometric optimization problems, in spirit related to the 'master' problem of *linear programming*:

> **(LP)** Given a convex polyhedron $\mathcal{P}$, specified by n halfspaces in d-space, and a d-vector v, find a point $p \in \mathcal{P}$ extreme in direction v.

The geometric formulation is chosen in order to keep notation consistent with another important LP-type problem, namely finding the *distance between convex polyhedra*:

> **(POLYDIST)** Given two convex polyhedra $\mathcal{P}$ and $\mathcal{Q}$, specified by n points (or n halfspaces) in d-space, find points $p \in \mathcal{P}$, $q \in \mathcal{Q}$ with $||p - q|| = \mathrm{dist}(\mathcal{P}, \mathcal{Q}) := \min\{||p' - q'|| \mid p' \in \mathcal{P}, q' \in \mathcal{Q}\}$.

The *minimum spanning ball problem* looks somewhat different, but nevertheless fits into the LP-type framework:

> **(MINIBALL)** Given n points in d-space, determine the center and radius of the smallest ball containing all the points.

POLYDIST and **MINIBALL** can easily be cast in the form of a convex program with only one constraint being nonlinear, and the general method of Grötschel, Lovász

* Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany (gaertner@inf.fu-berlin.de). Part of this work was done while the author was member of the Graduiertenkolleg "Algorithmische Diskrete Mathematik", supported by the Deutsche Forschungsgemeinschaft, grant We 1265/2-1; work has also been supported by the ESPRIT Basic Research Action Program 7141 of the EU (ALCOM II). A preliminary version has appeared in Proc. 33rd Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 464–472.

& Schrijver [11] can be applied to give polynomial algorithms for all three problems in the *Turing machine model*. This means that they can be solved in time polynomial in n, d and the encoding length of the input numbers.

The *arithmetic model* (or RAM model) – widely used in computational geometry – expresses the runtime in terms of the overall number of elementary (arithmetic) operations that are performed during the algorithm, where an arithmetic operation is charged unit cost, regardless of the sizes of the numbers involved. Thus the runtime in the arithmetic model is a function of the *number* of input numbers only and does not depend on their encoding lengths. Usually the arithmetic model is enhanced with the 'fairness assumption' that unit cost applies only to operations involving numbers which are not much larger than the input numbers (which here means that the encoding length does not exceed the maximum encoding length over all input numbers by more than a constant multiple). Bounds obtained in this model are called *combinatorial bounds*, and in contrast to the Turing machine model, there are no polynomial combinatorial bounds (also called *strongly polynomial* bounds) known for any of the three problems listed above (for details concerning the computational models see [11, pp. 32–33]).

We will be concerned with the arithmetic model of computation in this paper, and the bounds we develop are combinatorial. Although we do not succeed in proving polynomial bounds, we achieve substantial progress by showing that **POLYDIST** and **MINIBALL** have subexponential combinatorial complexity (for **LP** this had already been established, see below); all previous bounds were exponential.

The generic LP-type problem can be solved in a *local optimization* fashion, i.e. if a proposed solution is not optimal yet, one can locally improve on it by solving a 'small' subproblem. This basic property underlies the algorithm in [23] that solves the above problems in time linear in n but exponential in d. This bound is asymptotically optimal if d is constant and still reasonable if d is small compared to n [1]. However, as soon as d gets only moderately large with respect to n, the bound becomes unsatisfactory, and until recently no algorithms were known to have combinatorial complexity less than exponential in d *and* n for any of the three problems listed above. Kalai [12] and Matoušek, Sharir & Welzl [19] then independently came up with subexponential bounds for **LP**. The bound in [19] was obtained by tuning the analysis of the algorithm in [23], and although this algorithm can solve any LP-type problem, the subexponential analysis is valid only for **LP**. This is due to the fact that – at least under certain standard assumptions – the 'small' instances of **LP** ($n = d + 1$) can easily be solved in polynomial time, while for **POLYDIST** and **MINIBALL** the small problems are not substantially easier than the general ones.

In this paper we will show that the subexponential bound for **LP** is actually induced only by the local optimization property and does not rely on additional convenient features of the small instances. Consequently, **POLYDIST** and **MINIBALL** can be solved as efficiently as **LP** by our method. The tool is the framework of the *Abstract Optimization Problems* (AOPs) that captures the spirit of local optimization in a generic setting.

Basically, an AOP consists of a finite set H with a total order on 2^H and an oracle that answers the following queries: for given $F \subseteq G \subseteq H$, does there exist a set $F' \subseteq G$ with $F' < F$? If the answer is yes, such a set is returned as a witness. To solve the problem means to find the minimum set in the total order, and we want to

[1] Such assumptions are frequently made in computational geometry, and some bounds cited here make full sense only if d is substantially smaller than n.

bound the number of oracle queries needed to do this for any given AOP.

The algorithm we develop is *randomized*, i.e. we count the number of oracle queries, averaged over internal coin flips performed by the algorithm. We do not average over an input distribution, the expectation we get is valid for any input (in particular, there is no input that forces the algorithm to perform poorly). Moreover, randomization is crucial: if information about the linear order $<$ can be obtained from the oracle only, no deterministic algorithm can beat the trivial bound of $2^{|H|}-1$ oracle queries in the worst case. Although this does not mean much for a specific instance of an AOP, it gives evidence that randomized algorithms may be potentially more powerful than deterministic ones in this situation.

The paper is organized as follows: in the rest of the introduction we give a brief survey on results concerning the combinatorial complexity of the **LP**, **POLYDIST** and **MINIBALL** problems; in Section 2 we will discuss Sharir & Welzl's LP-type problems and point out why the subexponential bound for **LP** established in [19] does not hold for **POLYDIST** and **MINIBALL**. In Section 3 we formally introduce our abstract framework and state the main result of the paper that implies the subexponential bounds for **POLYDIST** and **MINIBALL**. Section 4 contributes the more technical part; it proves the deterministic lower bound and the randomized upper bound by presenting an algorithm in the abstract framework. Section 5 provides a concluding discussion.

Linear programming. **LP** problems are probably the best-understood optimization problems. There exists a vast amount of literature; the reader is referred to [21] for an introduction. There are several methods to solve LP-problems that are efficient in practice – the most popular one being the *simplex algorithm* that was introduced by Dantzig [6] in 1951. Its good performance on practical problems is in contrast to a result by Klee & Minty [16] who showed that a frequently used pivot rule leads to exponential behavior in d in the worst case (modifying the Klee-Minty construction, this was extended to other pivot rules, see Klee & Kleinschmidt [15]). The simplex algorithm is a 'combinatorial' algorithm in the sense that its behavior depends on the ordering of the vertices of the polyhedron along the optimization direction but not on their specific coordinates.

In contrast to this, the algorithms of Khachiyan (ellipsoid method [14]) and Karmarkar (interior point method [13]) are approximation algorithms which arrive at the desired optimum in time depending on the input precision. Both algorithms give weakly polynomial time bounds (i.e. polynomial in the Turing machine model), thus showing that linear programming belongs to the complexity class **P**. This caused considerable excitement in 1979 when Khachiyan's result became known. The quest for a strongly polynomial algorithm continues, but it is not generally believed that polynomial combinatorial bounds can be achieved.

Nevertheless, there has been substantial progress on the combinatorial complexity of linear programming over the last decade, mainly coming from computational geometry. Megiddo [18] was the first to show that **LP** can be solved in time linear in the number n of constraints (with a doubly exponential dependence on d). Subsequently the dependence on d has been improved step by step while keeping the bound linear in n, see Dyer [7], Clarkson [1], Seidel [22], Sharir & Welzl [23]. The currently best (randomized) algorithm combines the recent subexponential results in [12] and [19] with an algorithm by Clarkson [2]. This gives a bound of

$$O(d^2 n + e^{O(\sqrt{d \log d})}).$$

The best deterministic algorithm is due to Chazelle & Matoušek [4] and is obtained by 'derandomizing' Clarkson's algorithm. Its runtime is $O(d^{O(d)}n)$.

Distance between convex polyhedra. This problem – and the important special case that one polyhedron is a single point – is an instance of a quadratic optimization problem, and it has applications e.g. in motion planning (collision testing); there are heuristics for it without time analysis (see e.g. Wolfe [26] for the special case, Sekitani & Yamamoto [24] for the general case). A first nontrivial randomized time bound of $O(n^{\lfloor d/2 \rfloor})$ was given by Clarkson [3] (provided the polyhedra are specified by n points). Applying the algorithm in [23], this can be improved to give the so far best bound of $O(d^3 2^d n)$, which is linear in n but still exponential in d. Our algorithm will establish the same subexponential bound as stated above for linear programming.

Minimum spanning ball. It was observed early that **MINIBALL** (which is a prototype problem in facility location) has a structure similar to **LP** and thus can also be solved in time linear in the number of points by the techniques in [18] and [7] (with the same dependence on d as in the case of **LP**). As observed by Welzl [25], the **LP** algorithm of Seidel [22] also applies to **MINIBALL**, and the same holds for the algorithms of Clarkson and Sharir & Welzl. As in the case of **POLYDIST**, subexponential runtime could not yet be shown but will be established in this paper.

2. Basics and Terminology.

LP-type problems. To begin with, let us briefly review the concept of the *LP-type problems* introduced in [23], where the reader can also find how **LP** itself fits into the framework. Consider the **MINIBALL** problem first, and let H be a set of n points in d-space. For $G \subseteq H$ denote by $w(G)$ the radius of the smallest ball containing the points in G. It is well known that this ball is unique and that for $e \in H$, $w(G) < w(G \cup \{e\})$ if and only if e lies outside the ball determined by G. From this it is easily seen that the following two properties hold for all $F \subseteq G \subseteq H$:

(i) $w(F) \leq w(G)$.
(ii) if $w(F) = w(G)$, then $w(F) < w(F \cup \{e\}) \Leftrightarrow w(G) < w(G \cup \{e\})$, for all $e \in H$.

In general, any pair (H, w) satisfying (i) and (ii) is called an LP-type problem. A *basis* is a set $B \subseteq H$ with $w(B') < w(B)$ for all $B' \subsetneq B$. A basis of $G \subseteq H$ is a basis $B \subseteq G$ such that $w(B) = w(G)$. The *combinatorial dimension* of (H, w) is the maximum cardinality of any basis. In this framework, **MINIBALL** has combinatorial dimension at most $d+1$, because any minimum ball spanned by a set $G \subseteq H$ is already determined by at most $d+1$ points of G on the boundary.

POLYDIST gives rise to an LP-type problem as follows (we assume for the rest of the paper, that polyhedra $\mathcal{P}$ and $\mathcal{Q}$ are given by point sets P and Q, $P \cap Q = \emptyset$, $|P \cup Q| = n$. So $\mathcal{P}$ and $\mathcal{Q}$ are actually polytopes with $\mathcal{P} = \text{conv}(P), \mathcal{Q} = \text{conv}(Q)$; the case where $\mathcal{P}$ and $\mathcal{Q}$ are specified by halfspaces is similar but requires more technicalities).

For $P' \cup Q' \subseteq P \cup Q$, let

$$w(P' \cup Q') := \text{dist}(\text{conv}(P'), \text{conv}(Q')).$$

For P' or Q' empty, w is set to ∞. Now properties (i) and (ii) (for $w(F) = w(G) < \infty$) hold for the pair $(P \cup Q, w)$, but with '$<$' and '$\leq$' replaced by '$>$' and '$\geq$', respectively. Property (i) is obviously satisfied, so let us prove property (ii): assume

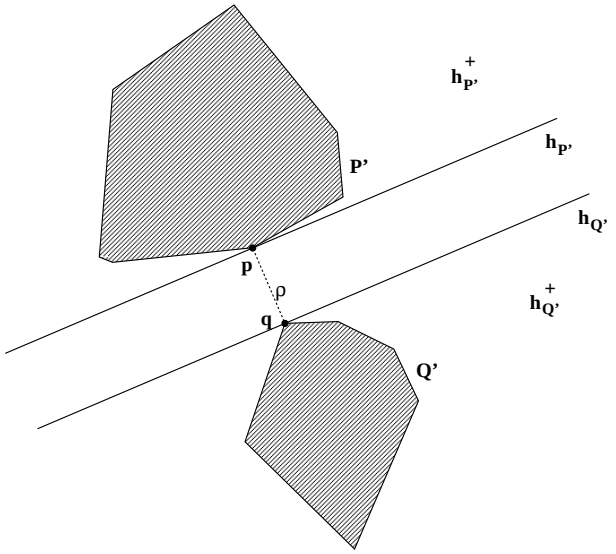

FIG. 1. *The* **POLYDIST** *problem*

$w(P' \cup Q') = \rho, 0 < \rho < \infty$. Then there exists a unique pair $h_{P'}, h_{Q'}$ of parallel supporting hyperplanes of distance ρ which are perpendicular to any vector $p-q$ with $p \in \text{conv}(P')$, $q \in \text{conv}(Q')$ and $||p-q|| = \rho$.

Denote by $h^+_{P'}$ and $h^+_{Q'}$ the closed halfspaces containing P' and Q', respectively. For a point $e \in P - P'$ $(Q - Q')$ we have $w(P' \cup Q' \cup \{e\}) < w(P' \cup Q')$ if and only if e does not lie in $h^+_{P'}$ $(h^+_{Q'})$ (Figure 1). This implies (ii).

It is a straightforward exercise to show that the following holds:

LEMMA 2.1. *In the LP-type framework, the combinatorial dimension of* **POLYDIST** *defined by polytopes $\mathcal{P}$ and $\mathcal{Q}$ in d-space satisfies $\delta \leq d+2$ (if $\mathcal{P}$ and $\mathcal{Q}$ are disjoint, $\delta \leq d+1$).*

Here is the main result of [19] [2]:

THEOREM 2.2. *Let (H, w) with $|H| = n$ be an LP-type problem of combinatorial dimension δ, and denote by t_{small} the time necessary to compute a basis of G and its value $w(G)$ for $|G| = \delta + 1$. Then a basis of H and $w(H)$ can be computed in time*

$$O(t_{small}\ ne^{2\sqrt{\delta \ln n}}).$$

As the theorem shows, the time bound for an LP-type problem crucially depends on the complexity of the 'small' instances, and it is not clear that solving them is a substantially easier task than solving the whole problem. For d-dimensional **LP**, the small problem basically consists of solving a linear program with $d+1$ constraints. Assuming that the program is bounded and in general position, its solution is determined as the intersection of exactly d of the constraint hyperplanes – which implies a polynomial procedure (for details see [19]).

In case of **POLYDIST** and **MINIBALL**, the small problems are

(**SMALL_POLYDIST**) Given two convex polyhedra $\mathcal{P}$ and $\mathcal{Q}$, specified by at most $d+3$ points $P \cup Q$ in d-space, determine their

[2] The bounds are actually slightly better (and more complicated) than what we cite here.

distance and a basis, i.e. a minimal subset $P' \cup Q'$ determining the same distance.

and

(**SMALL_MINIBALL**) Given a set H of at most $d+2$ points in d-space, determine the radius of the smallest ball containing H and a basis, i.e. a minimal subset B determining the same ball.

In contrast to **LP**, it is no longer true that the cardinality of a basis is known a priori, even under nondegeneracy assumptions. In case of **POLYDIST**, a basis of $P \cup Q$ may consist of any number of points between 2 and $d+2$. Consequently, straightforward checking of every candidate basis as in the case of **LP** may require the examination of $\Theta(2^d)$ subsets just to solve **SMALL_POLYDIST**, which gives nothing better than an exponential algorithm for the whole problem. The same difficulty arises in the **MINIBALL** problem: a minimum spanning ball may be determined by any number of points between 2 and $d+1$ on its boundary, so again **SMALL_MINIBALL** is not efficiently solvable by the trivial method. By embedding **SMALL_POLYDIST** into the AOP framework we will be able to solve it in time $e^{O(\sqrt{d})}$, and the same complexity will be achieved for **SMALL_MINIBALL**, which will turn out to be a special case of **SMALL_POLYDIST**. Plugging this into Theorem 2.2 will give subexponential bounds also for the corresponding 'large' problems.

3. Abstract Optimization Problems. Let us repeat the definition of an AOP in a formal way in order to have the accurate terminology available.

DEFINITION 3.1. *An AOP is a triple $(H, <, \Phi)$ where H is a finite set, $<$ is a total order on 2^H, and Φ is a mapping*

$$\Phi : \mathcal{H} \to H,\ \mathcal{H} := \{(F,G) \mid F \subseteq G \subseteq H\}$$

with the following property:

$$\Phi(F,G) = F \text{ if and only if } F = \min_{<}\{F' \mid F' \subseteq G\},$$
$$G \supseteq \Phi(F,G) < F \text{ otherwise.}$$

For $G \subseteq H$ let $\mathrm{opt}(G)$ *denote* $\min_<\{F \mid F \subseteq G\}$. Solving *the AOP means to find* $\mathrm{opt}(H)$.

We achieve the following results (proofs are postponed to the technical section):

THEOREM 3.2. **(Deterministic lower bound)** *For any* deterministic *algorithm $\mathcal{A}$ that solves all AOPs on a set H, $|H| = n$, there exists an AOP $(H, <, \Phi)$ that cannot be solved by $\mathcal{A}$ with less than $2^n - 1$ oracle queries.*

THEOREM 3.3. **(Randomized upper bound)** *There exists a* randomized *algorithm that solves any AOP on a set H, $|H| = n$, with an expected number of at most $e^{2\sqrt{n}+O(\sqrt[4]{n}\ln n)}$ oracle queries.*

Using the terminology from the previous section, we will now demonstrate how **POLYDIST** defined by point sets P and Q fits into the AOP framework: the ground set H is $P \cup Q$, and the total order $<$ on 2^H is defined as follows: the bases (in the LP-type sense) are ordered by their w-values with ties broken arbitrarily, and any non-basis is larger than any basis. This definition ensures that $\mathrm{opt}(H)$ indeed is a basis of $P \cup Q$ and the non-bases are not of interest.

It remains to describe the oracle Φ. Φ will only be called on pairs (F, G) where F is a basis and will deliver only bases. We need some more terminology: for a

point p and $P' \subseteq P$ with $p \in \text{conv}(P')$ (equivalently for q, Q') let $f(p, P') \subseteq P'$ be inclusion-minimal with $p \in \text{conv}(f(p, P'))$. For $F' = P' \cup Q' \subseteq P \cup Q$ let (p_F, q_F) be a pair of points realizing the distance between the affine hulls of P' and Q', i.e. $||p_{F'} - q_{F'}|| = \text{dist}(\text{aff}(P'), \text{aff}(Q'))$ (Figure 2 (a)). This point pair need not be unique in general; if $P' \cup Q'$ is a basis, however, it is.

Now we can implement the oracle $\Phi(F, G)$, $F = P' \cup Q' \subseteq P \cup Q = G$. Its idea is to start with points p and q realizing $w(F)$; provided that one can improve over F at all (which is the case if and only if a point of P lies in the complement of $h^+_{P'}$ or a point of Q lies in the complement of $h^+_{Q'}$ – we say that F is *violated* by that point), a loop is performed in which p and q move along straight lines – thereby decreasing $||p - q||$ – until a stable position, i.e. a new basis, is obtained. In the generic step of the loop there are points p, q and sets P', Q' such that $p \in \text{conv}(P'), q \in \text{conv}(Q')$. By definition, $||p - q|| \geq ||p_{F'} - q_{F'}||$, where $F' = P' \cup Q'$, so by moving p and q simultaneously along straight lines towards $p_{F'}$ and $q_{F'}$, respectively, their distance decreases in a monotone fashion. The movement stops if either the destination points are reached (in which case F' is a new basis) or one of p and q hits the boundary of $\text{conv}(P')$ or $\text{conv}(Q')$, respectively; in this case, the loop continues after setting P' to $f(p, P')$ and Q' to $f(q, Q')$ which decreases $|P' \cup Q'|$ by at least one (Figure 2 (b)). Here is the pseudocode[3] for the procedure just described.

```
Φ(F, G)                                  ▷F basis, F = P' ∪ Q' ⊆ P ∪ Q = G
 1   (p, q) := (p_F, q_F)                ▷p_F, q_F ∈ conv(P'), conv(Q'), unique
 2   for all e ∈ G − F
 3       do if e violates F
 4          then case e of
 5                       ∈ P : P' ← P' ∪ {e}
 6                       ∈ Q : Q' ← Q' ∪ {e}
 7                  loop F' ← P' ∪ Q'
 8                       (p_λ, q_λ) := (p, q) + λ((p_F', q_F') − (p, q))
 9                       μ ← max{λ | p_λ ∈ conv(P'), q_λ ∈ conv(Q')}
10                       if μ > 1
11                          then return F'
12                       (p, q) ← (p_μ, q_μ)
13                       (P', Q') ← (f(p, P'), f(q, Q'))
14   return F
```

From the discussion above, termination of the procedure follows. To see the correctness, observe that the pair $(p_{F'}, q_{F'})$ in line 8 is unique in any iteration of the loop, since $\text{aff}(P')$ and $\text{aff}(Q')$ are neither parallel (unless $|P'| = 1$ or $|Q'| = 1$, in which case uniqueness holds anyway) nor do they have a nontrivial intersection. From this it follows that the set F' finally returned in line 11 is indeed a basis.

$||p-q||$ strictly decreases in every iteration of the loop, so the basis finally returned has smaller w-value than F.

We are interested in the runtime of this oracle, when called on a 'small' problem, i.e. $|G| \leq d + 3$. The following bound is certainly not best possible; the interesting fact is that it is polynomial.

LEMMA 3.4. *Let G be a set of at most $d+3$ points in d-space. The oracle $\Phi(F, G)$ can be implemented to run in time $O(d^5)$.*

[3] The variant we use here is adapted from the book *Introduction to Algorithms* by T. H. Cormen, C. E. Leiserson & R. L. Rivest, The MIT Press, Cambridge, MA, 1990

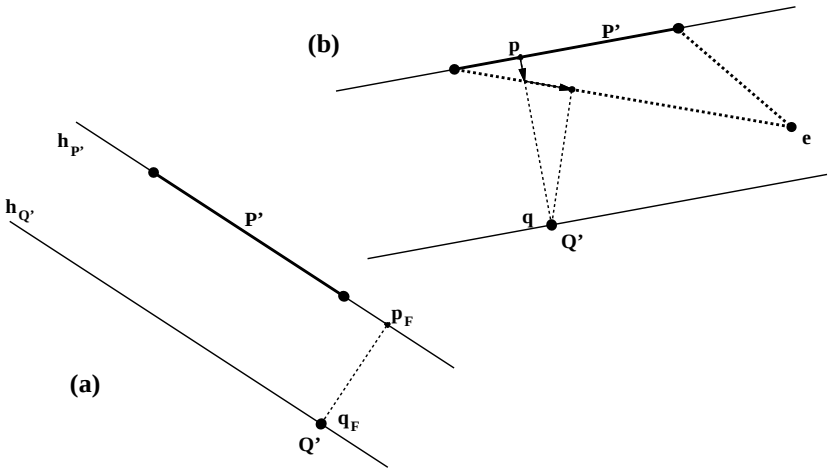

FIG. 2. *The* **POLYDIST** *oracle*

Proof. The first phase (checking whether an improvement over F is possible) amounts to the computation of one scalar product per point $e \in G - F$ and therefore can be done in time $O(d^2)$. The loop is executed $O(d)$ times; in each iteration, points $p_{F'}$, $q_{F'}$ can be computed in time $O(d^3)$ by solving a system of linear equalities. In order to find μ, we have to intersect two lines with the bounding hyperplanes of $\mathrm{conv}(P')$ and $\mathrm{conv}(Q')$, respectively. From the fact that F was a basis it follows that these polytopes are simplices, so there are no more than $d+1$ bounding hyperplanes per polytope, and each intersection can be computed in time $O(d^3)$ again. This process also gives $f(p, P')$ and $f(q, Q')$ as a by-product. □

By arbitrarily choosing two vertices in the beginning, one from each polytope, we get an initial basis for the oracle. By Theorem 3.3 a **SMALL_POLYDIST** problem can be solved in time $e^{O(\sqrt{d})}$, and together with Theorem 2.2 this gives

THEOREM 3.5. *The distance between two n-vertex polytopes in d-space can be computed in time $O(ne^{(2+o(1))(\sqrt{d \ln n})})$.*

We remark that the same result holds for two n-facet convex polyhedra. Note that the contribution from the small problem is hidden in the $o(1)$ term of the exponent.

We get the same bound for **MINIBALL**:

THEOREM 3.6. *The smallest enclosing ball of a set of n points in d-space can be computed in time $O(ne^{(2+o(1))(\sqrt{d \ln n})})$.*

For this it suffices to show that a **SMALL_MINIBALL** problem can be solved as efficiently as **SMALL_POLYDIST**, and as it turns out, both problems are strongly related; the following correspondence can be found e.g. in [20]:

THEOREM 3.7. *Let P be a set of $d+1$ affinely independent points in d-space with circumcenter q_0. The center q_1 of the smallest ball containing P is the point in* $\mathrm{conv}(P)$ *with minimum distance to q_0.*

Thus, in order to solve **SMALL_MINIBALL** on $d+2$ points, compute – by solving $d+2$ **SMALL_POLYDIST** problems with one polytope being a single point – all the smallest balls spanned by $d+1$ of the points, and compare their radii. In case the input consists of d points or less, the problem restricts to a lowerdimensional one inside the affine hull of the points. Note that the circumcenter of $d+1$ points can be computed in time $O(d^3)$.

4. Bounds for Abstract Optimization Problems. This section contains the proofs of Theorems 3.2 and 3.3. The deterministic lower bound follows from an adversary argument. For the randomized upper bound we present an algorithm along with a careful analysis. Let us start with the lower bound.

4.1. The Deterministic Lower Bound. Let H be an n-element set and suppose we have a deterministic algorithm to solve any AOP $(H, <, \Phi)$.

We start the algorithm on a problem $(H, <_0, \Phi_0)$ with $<_0$ and Φ_0 not yet determined, and we argue that an adversary answering the oracle queries can construct $<_0$ and Φ_0 'online' in such a way that the algorithm is forced to step through at least $2^n - 1$ queries. When supplied with a query pair (F, G), the adversary will output an answer $F' = \Phi_0(F, G)$ according to two simple rules:

> (i) the answer F' is consistent with the previous ones, i.e. there exists an AOP such that the current and all previous queries have been answered correctly with respect to this AOP.
>
> (ii) $F' = F$ if and only if there is no other consistent answer.

It is easy to see that the adversary always has a consistent answer, so the algorithm steps through a sequence of queries with pairs (F, G) and finally stops. Suppose that less than $2^n - 1$ queries have been performed. Then there are two sets F_1 and F_2 which have never been the first component of a query pair. We will show that it is consistent with all answers to assume that $F_1 = \mathrm{opt}(H)$. The same holds for F_2, so whatever the algorithm outputs, there is an AOP that is not correctly solved. Hence the adversary can force the algorithm to step through at least $2^n - 1$ queries, which means that it performs that many queries on the AOP that has implicitly been constructed by the adversary.

We are left to prove that $F_1 = \mathrm{opt}(H)$ is consistent. Clearly, this choice can fail only if some answer has revealed the existence of a smaller set. Since there was no query pair (F_1, G), the only remaining possibility for this to happen is that some query (F, G) with $F_1 \subseteq G$ has been answered by F, thus establishing $F < F_1$. But in the first query of this type, F_1 was not bounded from below yet, so it could have been returned instead of F, a contradiction to rule (ii).

4.2. The Randomized Upper Bound. We present a randomized algorithm that solves any given AOP $(H, <, \Phi)$ on an n-element set H with an expected number of at most $e^{2\sqrt{n}+O(\sqrt[4]{n}\ln n)}$ calls to the oracle Φ. Up to an $O(n)$ overhead caused by set operations, the actual runtime of the algorithm will asymptotically be dominated by the time spent on the oracle calls, so this is a reasonable measure of complexity. The algorithm will eventually output $\mathrm{opt}(H)$, but the generic step in the recursive procedure is the computation of $\mathrm{opt}(G)$ for $G \subseteq H$. Together with G, a set $F \subseteq G$ is maintained; throughout the section we will refer to F as an *estimate* for the solution that will be improved over and over again until it coincides with the desired minimum. Our algorithm combines ideas of both Kalai's and Matoušek, Sharir & Welzl's subexponential **LP** algorithms and substantially generalizes their applicability.

The section proceeds in stages: in a first stage we introduce a trivial algorithm and the basic terminology; the second stage presents an algorithm that – although it works only modulo a hypothetic subroutine – features the heart of the final algorithm and its subexponential analysis. Stage three describes a 'working' algorithm that will be obtained by 'approximating' the subroutine to a reasonable extent.

Getting Started. Just to get acquainted, we give the obvious deterministic method to obtain $\mathrm{opt}(G)$ in presence of an estimate F.

```
AOP_DET(F, G)
1   repeat F' ← F
2          F ← Φ(F, G)
3   until F = F'
4   return F'
```

In order to solve the problem on G, AOP_DET has to call the oracle $2^{|G|}-1$ times in the worst case, and as we have seen in the previous section, it shares this exponential behavior with any algorithm that is deterministic or calls the oracle only on pairs of the form $(*, G)$. Consequently, the method we will describe now is randomized and uses oracle queries of the form $\Phi(*, G')$ for certain subsets $G' \subseteq G$. It will have one very intuitive property in common with AOP_DET: the better the estimate F, the faster the algorithm for G. A natural measure for the quality of F in this context is the *rank* of F with respect to G, defined by

$$\mathrm{rank}(F, G) := \#\{F' \subseteq G \mid F' < F\}.$$

A somewhat coarser but related measure is the *dimension* of a pair (F, G):

DEFINITION 4.1. *For $F \subseteq G$, an element $e \in G$ is* enforced *in (F, G) if $F < \mathrm{opt}(G - \{e\})$ (and this implies $e \in F$). Otherwise it is* free. *The* domain *of (F, G) is the set of free elements, i.e.*

$$\mathcal{D}(F, G) := \{e \in G \mid \exists F' \subseteq G - \{e\}, F' \leq F\}.$$

For a free element e, an estimate $F' \subseteq G - \{e\}$ with $F' \leq F$ is called a witness *for e. Finally, the* dimension *of (F, G) is the size of its domain,*

$$\dim(F, G) := |\mathcal{D}(F, G)|.$$

The enforced elements in (F, G) are exactly the ones contained in *every* estimate $F' \subseteq G$ with $F' \leq F$, while the free elements have at least one witness $F' \subseteq G - \{e\}$ with $F' \leq F$.

As an example consider Figure 3: subsets of G are visualized by elements of $\{\blacksquare, \square\}^{|G|}$. (F, G) enforces the element 2 while (F', G) enforces 2 and 5. Consequently, $\mathcal{D}(F, G) = \{1, 3, 4, 5\}$, $\mathcal{D}(F', G) = \{1, 3, 4\}$ and $\dim(F, G) = 4, \dim(F', G) = 3$. Note that we always have $\dim(F, G) \geq \log_2(\mathrm{rank}(F, G) + 1)$.

The following monotonicity Lemma is an immediate consequence of the definitions and (although quite obvious) forms the background of the analysis in the next stage.

LEMMA 4.2.

(i) If $F' \leq F$ then $\mathcal{D}(F', G) \subseteq \mathcal{D}(F, G)$.
(ii) If $F \subseteq G \subseteq G'$ then $\mathcal{D}(F, G) \subseteq \mathcal{D}(F, G')$.

The basic algorithm. Now we are able to describe a first basic version of our algorithm. We feel that the main idea behind the subexponential analysis can be explained most clearly by assuming that the following subroutine is available.

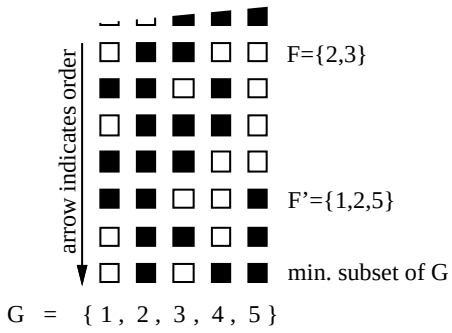

FIG. 3. *Part of an AOP on 5 elements*

SAMPLE_DOMAIN(F, G)
1 **if** $\mathcal{D}(F, G) = \emptyset$
2 **then error** "empty"
3 choose a random e from $\mathcal{D}(F, G)$
4 choose a witness $F_e \leq F$ with $F_e \subseteq G - \{e\}$
5 **return** (e, F_e)

So SAMPLE_DOMAIN chooses an element which is free in (F, G) at random and outputs it, along with a corresponding witness. Note that the elements in $G - F$ are free with witness F; however, there may be other free elements which are not immediately accessible since they are hidden in F, and it is not clear whether one can find them efficiently. Nevertheless, let us assume for this stage that SAMPLE_DOMAIN comes for free.

Using this subroutine, we can formulate a procedure AOP_SD(F, G) to find opt(G) in presence of estimate F.

AOP_SD(F, G)
1 $(e, F_e) :=$ SAMPLE_DOMAIN(F, G)
2 **if error** "empty"
3 **then return** F
4 $F \leftarrow$ AOP_SD$(F_e, G - \{e\})$
5 $F' \leftarrow \Phi(F, G)$
6 **if** $F = F'$
7 **then return** F'
8 **return** AOP_SD(F', G)

The analysis. Termination and Correctness easily follow by observing that the first recursive call solves a subproblem on a smaller set and in the second one we have rank$(F', G) <$ rank(F, G). For fixed H let $T(k)$ be the worst case expected number of oracle queries performed in a call to AOP_SD(F, G) with $\dim(F, G) = k$ and $G \subseteq H$. We get $T(0) = 0$ and for $k > 0$

THEOREM 4.3.

$$T(k) \leq T(k-1) + 1 + \frac{T(k-1) + \ldots + T(0)}{k}.$$

Proof. First of all, it is not hard to see that T is monotone in k. Now we can argue as follows: if (F,G) has dimension k then $\dim(F_e, G-\{e\}) \leq k-1$, since

$$\mathcal{D}(F_e, G-\{e\}) \cup \{e\} \subseteq \mathcal{D}(F_e, G) \subseteq \mathcal{D}(F,G).$$

This implies that the expected number of oracle queries necessary in the first recursive call is bounded by $T(k-1)$. Another query is done in line 5; the last term finally gives a bound for the second recursive call. To see this, consider $\mathcal{D}(F,G) = \{e_1, ..., e_k\}$, ordered in such a way that

$$\text{opt}(G-\{e_1\}) \geq \text{opt}(G-\{e_2\}) \geq ... \geq \text{opt}(G-\{e_k\}).$$

If e is chosen to be e_i by SAMPLE_DOMAIN, we get

$$F' < \text{opt}(G-\{e_j\}), \quad \text{for all } j \leq i,$$

so by Definition 4.1, $e_1, ..., e_i$ will be enforced in (F', G). This means

$$\mathcal{D}(F', G) \subseteq \mathcal{D}(F,G) - \{e_1, ..., e_i\},$$

so $\dim(F', G) \leq k-i$ and an expected number of no more than $T(k-i)$ oracle queries is performed by AOP_SD(F', G). Since i is equally likely to be any number between 1 and k, we obtain the desired average. □

The heart of the argument is the fact that by choosing a *random* element for the recursion in line 4 of AOP_SD the dimension halves on the average, while choosing e deterministically can only guarantee a progress of one in the worst case. This basic observation also underlies the subexponential LP-algorithms of Kalai [12] and Matoušek, Sharir & Welzl [19].

In order to find an explicit bound for $T(k)$, we majorize $T(k)$ by $t(k)-1$ where $t(k)$ satisfies

$$t(k) = t(k-1) + \frac{1}{k}\sum_{i=0}^{k-1} t(i)$$

with $t(0) = 1$.

LEMMA 4.4.

$$t(k) = \sum_{i=0}^{k} \binom{k}{i} \frac{1}{i!}.$$

Proof. An easy way to see this goes via a nice combinatorial interpretation of $t(k)$ (suggested by P. Flajolet): Consider a permutation $\pi \in S_k$. A subset $R \subseteq \{1, ..., k\}$ is called an *increasing chain* in π iff $\forall\, x, y \in R : x < y$ implies $\pi(x) < \pi(y)$. Denote by $s(\pi)$ the number of increasing chains in π. Then $t(k) = E[s]$, the expected number of increasing chains in a random permutation $\pi \in S_k$. The proof of this fact is by induction. For $k = 0$ we have one increasing chain, namely the empty set. Assume $k > 0$; by the inductive hypothesis the expected number of increasing chains not containing k is $t(k-1)$. The ones containing k are in one-to-one correspondence with the increasing chains in $\{1, ..., \pi^{-1}(k)-1\}$, whose expected number is $t(\pi^{-1}(k)-1)$.

Since $\pi^{-1}(k)$ is equal to i with probability $1/k$, for any $i \in \{1, ..., k\}$, we recover the original recurrence for $t(k)$. On the other hand,

$$t(k) = E[s] = \sum_R \text{prob}(R \text{ is an increasing chain})$$
$$= \sum_{i=0}^{k} \sum_{|R|=i} \text{prob}(R \text{ is an increasing chain}) = \sum_{i=0}^{k} \binom{k}{i} \frac{1}{i!},$$

and the lemma follows. □

COROLLARY 4.5.

$$t(k) \leq e^{2\sqrt{k}}.$$

Proof. Using the inequality $\binom{k}{i} \leq k^i/i!$ we obtain

$$t(k) = \sum_{i=0}^{k} \binom{k}{i} \frac{1}{i!} \leq \sum_{i=0}^{k} \frac{k^i}{i!^2} = \sum_{i=0}^{k} (\frac{\sqrt{k}^i}{i!})^2 \leq (\sum_{i=0}^{k} \frac{\sqrt{k}^i}{i!})^2 \leq (\sum_{i=0}^{\infty} \frac{\sqrt{k}^i}{i!})^2 = e^{2\sqrt{k}}. \quad \square$$

The bound is almost tight – we will come back to this later. Concerning the performance of AOP_SD we get

THEOREM 4.6. AOP_SD *solves any AOP* $(H, <, \Phi)$ *with an expected number of at most* $e^{2\sqrt{|H|}} - 1$ *oracle queries.*

How to sample from the domain. In order to turn the procedure AOP_SD from the previous paragraph into a working algorithm, we have to do something about the subroutine SAMPLE_DOMAIN. As we have already indicated, the way to deal with it will be to find a reasonably cheap way to 'approximate' it. The idea is simple: rather than sampling from the whole domain of a given pair (F, G), we will identify a subset D of the domain – along with the corresponding witnesses – and sample from D only. After plugging in this version of SAMPLE_DOMAIN, the expected performance of AOP_SD will drop off, depending on the size of D, which we assume to be a function $r(k)$ of $k = \dim(F, G)$; the recurrence of Theorem 4.3 then becomes

$$T(k) \leq T(k-1) + \frac{T(k-1) + ... + T(k-r(k))}{r(k)}.$$

This bound is exponential for $r(k) = O(1)$ but becomes better the closer $r(k)$ is to k. On the other hand, the larger $r(k)$ is, the harder is the task of finding the set D. A reasonable balancing is achieved by choosing $r(k)$ proportional to k, say $r(k) = \lceil ck \rceil$, for some fixed $0 < c < 1$ (the exact value will be determined later and will depend on $n = |H|$). We will see that the additional effort to find $D \subseteq \mathcal{D}(F, G)$ of this size is small, and the algorithm basically preserves its expected performance as analyzed above.

To construct D when called on a pair (F, G), the algorithm proceeds incrementally; since we know that at least the elements in $G - F$ are free with witness F, we can start off by setting D to $G - F$. In case D is already large enough, we just sample from D and proceed as before. Otherwise we will have to enlarge D by at least one more free element hidden in F; to this end we will step through a sequence of improving oracle queries (in a way to be described in a minute), until a witness for a yet unknown free element is found (or we already end up in $\text{opt}(G)$).

Suppose that in the generic step certain elements in G have already been identified as free in (F, G), and we need to find another free one. Here is the way to do it: call the algorithm recursively with (F', G), where F' is the current estimate, but supply an additional parameter E that contains all the elements of G whose status is yet unknown (note that $E \subseteq F'$). This recursive call now has two ways to terminate: either it finds the desired solution $\mathrm{opt}(G)$ or – while improving its estimate – discovers an $F'' \subseteq G$ which fails to contain E. This, however, means that the elements in $E - F''$ have been uncovered as free elements with witness F'', so the call has accomplished its task. The key observation is that as long as D is small, the set E of elements with unknown status will be large, and since the recursive call with parameter E terminates as soon as the first estimate F'' appears that is no longer wedged between E and G, it actually operates only on $G - E$ instead of G, which makes it substantially cheaper (this method is a generalization of the idea behind the pivoting strategy Kalai uses in his **LP** algorithm).

A working algorithm. The generic call will have three parameters $E \subseteq F \subseteq G$, where in the beginning $E = \emptyset$. Let us formulate the procedure $\textsc{Aop}(E, F, G)$ that will either return $\mathrm{opt}(G)$ or deliver an estimate $F' < F$ with $E \not\subseteq F'$ to a higher level in the recursion. The set D is implicitly maintained, comments will refer to it.

```
Aop(E, F, G)                                  ▷ returns opt(G) or F' < F, E ⊈ F'
 1   if E = G                                 ▷ E = F = G ?
 2      then return Φ(F, G)                   ▷ Φ(F, G) = F or E ⊈ Φ(F, G)
 3   E' ← F                                   ▷ elements of unknown status
 4   for all e ∈ G − E'                       ▷ D := G − E', initial free elements
 5      do F_e ← F                            ▷ set witness
 6   while |G − E'| < ⌈c|G − E|⌉              ▷ D still to small, try to enlarge it
 7      do F ←Aop(E', F, G)
 8         if E ⊈ F                           ▷ return to higher level in recursion
 9            then return F
10         if E' ⊈ F                          ▷ new free element(s) found
11            then for all e ∈ E' − F
12                    do F_e ← F              ▷ set witness
13                 E' ← E' ∩ F                ▷ update E' (D := D ∪ (E' − F))
14            else return F                   ▷ line 7 has already computed opt(G)
15   choose a random e ∈ G − E'               ▷ sample from D
16   F ←Aop(E, F_e, G − {e})
17   if E ⊈ F                                 ▷ return to higher level in recursion
18      then return F
19   F' ← Φ(F, G)                             ▷ once we get here, F = opt(G − {e})
20   if E ⊈ F' or F = F'                      ▷ check both termination criteria
21      then return F'
22   return Aop(E ∪ {e}, F', G)               ▷ repeat with better estimate
```

Termination of Aop follows by observing that the recursive calls solve smaller problems (measured in terms of $|G - E|$). The correctness of the procedure is not obvious at first sight, but it suffices to inductively check the invariant that E is contained in every estimate up to the terminating one. This is not a priori clear in the recursive call of line 22, where E is replaced with $E \cup \{e\}$; to see that this is justified, observe that if the procedure gets through to line 22 at all, the call in ine 16 must have actually computed $F = \mathrm{opt}(G - \{e\})$; moreover, $F' < F$. This, however,

means that e is enforced in (F', G), so for every future estimate $F'' \subseteq G$ with $F'' \leq F'$ we will have $e \in F''$, i.e.

$$E \not\subseteq F'' \Leftrightarrow E \cup \{e\} \not\subseteq F'',$$

and the invariant is guaranteed.

It should be mentioned that the estimate F_e plugged into the recursive call in line 16 may be worse than some estimates computed during the **while**-loop. The important property, however, is that F_e is at least as good as the original F we started with.

Towards the recurrence. The reader might wonder whether this algorithm really matches the rough idea described above. For example, we have promised to make D as large as $\lceil c|\mathcal{D}(F,G)|\rceil$. This holds if $E = \emptyset$, because in this case after the **while**-loop

$$|D| \geq \lceil c|G - E|\rceil = \lceil c|G|\rceil \geq \lceil c|\mathcal{D}(F,G)|\rceil,$$

but if $|E| > 0$, we might end up with a much smaller D. In this case, however, there are elements in $\mathcal{D}(F,G)$ (especially the ones in E) which are actually enforced in the sense that every estimate containing E has to contain these elements as well, and we should no longer consider such elements as free in a call with parameters E, F and G. The following definition for triples takes care of that; it mimics Definition 4.1 for pairs.

DEFINITION 4.7. *For $E \subseteq F \subseteq G$, $e \in G - E$ is* enforced *in (E,F,G) if $F < \text{opt}(E, G - \{e\}) := \min_<\{F' \mid E \subseteq F' \subseteq G - \{e\}\}$, and e is* free *otherwise. The free elements form the* domain *of (E,F,G), i.e.*

$$\mathcal{D}(E,F,G) := \{e \in G - E \mid \exists F' \leq F, E \subseteq F' \subseteq G - \{e\}\}.$$

The dimension *of (E,F,G) is the size of its domain, i.e.*

$$\dim(E,F,G) := |\mathcal{D}(E,F,G)|.$$

This ensures that the domain contains exactly the elements which may potentially end up in D and guarantees that $|D| \geq \lceil c|\mathcal{D}(E,F,G)|\rceil$ after the **while**-loop.

The main recurrence. In order to bound the expected performance of AOP, we can set up a recurrence which will look similar to the one that holds for AOP_SD. However, it will include an additional term for the effort that is necessary to find the set D; even worse, it will depend on two parameters rather than one which makes it somewhat harder to solve. Nevertheless, it basically behaves like the one-parameter version and we will be able to establish a similar bound.

For fixed H and $m \geq k$, let $T(m,k)$ be the worst-case expected number of oracle queries performed in a call to $\text{AOP}(E,F,G)$ with *size* $|G - E| = m$ and dimension $\dim(E,F,G) = k$, $G \subseteq H$. For a statement A let χ_A be 1 if A holds and 0 otherwise. We get $T(0,0) = 1$ and for $m > 0$ we have

THEOREM 4.8.

$$T(m,k) \leq \sum_{i=0}^{\lceil cm\rceil - 1} T(i, \min(i,k)) + \left[T(m-1,k-1) + 1 + \frac{1}{\lceil cm\rceil}\sum_{i=1}^{\lceil cm\rceil} T(m-1,k-i)\right]\chi_{k \geq \lceil cm\rceil}.$$

Proof. The arguments are basically the ones of Theorem 4.3, so we omit some details. Again T is monotone in k; in the worst case, the **while**-loop may be executed once with every value of $i := |G - E'|$ between 0 and $\lceil cm \rceil - 1$, which gives the first term. If $k < \lceil cm \rceil$, the algorithm will not be able to find the required number of free elements and thus cannot get beyond the **while**-loop. Otherwise, the triple processed in line 16 has size $m-1$ and dimension at most $k-1$ (e is no longer free) which gives the $T(m-1,k-1)$ term. One more query may be necessary in line 19. Finally, the last term bounds the expected effort in line 22, averaged over the random choice in line 15 (let $G - E' = e_1, ..., e_l, l \geq \lceil cm \rceil$, ordered in such a way that

$$\mathrm{opt}(E, G - \{e_1\}) \geq ... \geq \mathrm{opt}(E, G - \{e_l\})$$

and observe that $e_1, ..., e_i$ are no longer free in $(E \cup \{e\}, F', G)$ if $e = e_i$ is chosen in line 15). □

Solving the recurrence. It turns out that the dependence of $T(m,k)$ on m is quasipolynomial, while the major contribution comes from k. Let us define two auxiliary functions:

$$f(k) = f(k-1) + \frac{1}{\lceil ck \rceil} \sum_{i=1}^{\lceil ck \rceil} f(k-i),$$

$$g(m) = g(m-1) + \sum_{i=0}^{\lceil cm \rceil - 1} g(i).$$

with $f(0) = g(0) = 1$.

LEMMA 4.9.

$$T(m,k) \leq f(k)g(m).$$

Proof. We proceed by induction on m. For $m = 0$ we have equality, so assume $m > 0$. Then

$$\begin{aligned}
T(m,k) \leq & \sum_{i=0}^{\lceil cm \rceil - 1} f(\min(i,k))g(i) + \\
& \left[f(k-1)g(m-1) + 1 + \frac{1}{\lceil cm \rceil} \sum_{i=1}^{\lceil cm \rceil} f(k-i)g(m-1) \right] \chi_{k \geq \lceil cm \rceil} \\
\leq & f(k) \sum_{i=0}^{\lceil cm \rceil - 1} g(i) - \chi_{k>0} + \\
& g(m-1) \left[f(k-1) + \frac{1}{\lceil ck \rceil} \sum_{i=1}^{\lceil ck \rceil} f(k-i) \right] \chi_{k \geq \lceil cm \rceil} + \chi_{k \geq \lceil cm \rceil} \\
= & f(k)[g(m) - g(m-1)] + g(m-1)f(k)\chi_{k \geq \lceil cm \rceil} + \chi_{k \geq \lceil cm \rceil} - \chi_{k>0} \\
= & f(k)g(m) - g(m-1)[f(k) - f(k)\chi_{k \geq \lceil cm \rceil}] + \chi_{k \geq \lceil cm \rceil} - \chi_{k>0} \\
\leq & f(k)g(m) - g(0)[f(0) - f(0)\chi_{k \geq \lceil cm \rceil}] + \chi_{k \geq \lceil cm \rceil} - \chi_{k>0} \\
\leq & f(k)g(m). \qquad \square
\end{aligned}$$

It remains to bound $f(k)$ and $g(m)$.

LEMMA 4.10.

$$g(m) \le 3m^{\log_{1/c} m+1} - 1, \text{ for } m > 0,$$
$$f(k) \le e^{2\sqrt{k/c}}.$$

Proof. The function $g(m)$ is monotone, so we can argue that $g(m) \le g(m-1) + \lceil cm \rceil g(\lceil cm \rceil - 1)$, which by expanding $g(m-1)$ yields

$$g(m) \le \sum_{i=1}^{m} \lceil ci \rceil g(\lceil ci \rceil - 1) + g(0) \le m\lceil cm \rceil g(\lceil cm \rceil - 1) + 1 \le m^2 g(\lceil cm \rceil - 1) + 1.$$

Now we claim that for $m > 0$, $g(m) \le 3m^{log_{1/c} m+1} - 1$ which holds for $m = 1$, and inductively we obtain

$$g(m) \le m^2[3(cm)^{\log_{1/c} cm+1} - 1] = 3m^2 m^{\log_{1/c} m-1} - m^2 \le 3m^{\log_{1/c} m+1} - 1.$$

$f(k)$ can be majorized by the function $t(k)$ satisfying $t(0) = 1$ and

$$t(k) = t(k-1) + \frac{1}{ck} \sum_{i=1}^{k} t(k-i).$$

By applying the method of generating functions [9, Section 7] it is possible to compute $t(k)$ explicitly. One obtains

$$t(k) = \sum_{i=0}^{k} \binom{k}{i} \frac{(1/c)^i}{i!},$$

a generalization of the bound in Lemma 4.4. The formula can also easily be verified by induction, using standard binomial coefficient identities as can be found in [9, Section 5]. Completely similar to Corollary 4.5 we obtain from this the desired estimate

$$t(k) \le e^{2\sqrt{k/c}}. \qquad \square$$

It takes some more effort to show that $t(k) = \Theta(e^{2\sqrt{k/c}}/\sqrt[4]{k})$ (e.g. by using the saddle point method [10, pp. 74 ff.]), so our simple estimate is tight up to a $\sqrt[4]{k}$ factor.

We have proved that the expected number of oracle queries performed by algorithm AOP when called on a triple (E, F, G) of size m and dimension k, is bounded by

$$T(m,k) \le 3e^{2\sqrt{k/c}} m^{\log_{1/c} m+1}$$

for any fixed $0 < c < 1$. Setting $z := 1/c - 1$ and rewriting the expression shows that

$$T(n,n) \le 3ne^{2\sqrt{n+zn}+\ln^2 n/\ln(1+z)}$$

queries are sufficient on the average to solve an Abstract Optimization Problem on a set H with n elements. By easy calculations, we see that the value z_0 minimizing this bound satisfies

$$z_0 = \frac{\ln n}{\sqrt[4]{n}}(1 + o(1)),$$

so a reasonable choice for z is $z = \ln n / \sqrt[4]{n}$. Exploiting that $\ln(1+z) \geq z - z^2/2$ for positive z gives

$$\frac{\ln^2 n}{\ln(1+z)} \leq \frac{\ln^2 n}{z - z^2/2} = \frac{\ln^2 n}{z} + \frac{\ln^2 n}{2-z} \leq \sqrt[4]{n} \ln n + \ln^2 n.$$

This derivation holds for $z \leq 1$; in case $z > 1$, the inequality follows directly. From the mean value theorem we get

$$\sqrt{n + zn} \leq \sqrt{n} + \frac{z}{2}\sqrt{n} \leq \sqrt{n} + \frac{1}{2}\sqrt[4]{n} \ln n.$$

This leads to an overall upper bound of

$$T(n,n) \leq 3ne^{2\sqrt{n} + 2\sqrt[4]{n}\ln n + \ln^2 n}.$$

THEOREM 4.11. *An Abstract Optimization Problem $(H, <, \Phi)$ with $|H| = n$ can be solved with an expected number of no more than*

$$e^{2\sqrt{n} + 2\sqrt[4]{n}\ln n + O(\ln^2 n)}$$

oracle queries.

5. Discussion. We have given an $e^{(2+o(1))\sqrt{n}}$ expected time randomized algorithm for Abstract Optimization Problems on an n-element set H; the algorithm applies to the minimum spanning ball problem for n points in d-space and to the problem of computing the distance between two d-dimensional n-vertex (or n-facet) convex polyhedra, and for both problems we obtain the first subexponential combinatorial bounds in d. As in the case of linear programming one can obtain a bound of $O(d^2 n + e^{O(\sqrt{d \log d})})$ for both problems by combining our result with the ones of [19] and [2].

Recently, Ludwig [17] has shown that the problem of finding optimal strategies for *simple stochastic games* [5] allows a subexponential solution (where he implicitly proved that the problem can be formulated as an AOP which allows direct application of the procedure AOP_SD from Subsection 4.2); unlike the AOPs we have discussed here, the simple stochastic game problem is of a more combinatorial rather than geometric flavor. In general, however, it seems that local optimization occurs most naturally in geometric settings.

To determine the randomized complexity of Abstract Optimization Problems itself is a challenging problem. It seems that in order to substantially improve on the bound given here one would have to come up with a method that does not rely on the concept of the dimension of an estimate as a measure of progress during the algorithm. Any better upper bound would immediately imply better algorithms for the 'small' instances of the problems we have discussed – and we believe that the ideas behind such progress would carry over to the 'large' instances. On the other hand, it is quite possible that there is a lower bound for AOPs that is in the range of the upper bound given here; however, so far we have not been able to establish any nontrivial lower bound. A natural time class to consider when thinking about upper or lower bounds seems to be $n^{\log^r n}$ for constant r.

Acknowledgments. The author wishes to thank an anonymous referee for comments that helped to improve the presentation. Special thanks to Emo Welzl for many discussions and for reading several versions of the manuscript.

REFERENCES

[1] K. L. Clarkson, *Linear programming in $O(n3^{d^2})$*, Inform. Process. Lett., 22 (1986), pp. 21–24.

[2] ———, *Las Vegas algorithms for linear programming when the dimension is small*, manuscript, 1989.

[3] ———, *New applications of random sampling in computational geometry*, Discr. Comp. Geom., 2 (1987), pp. 195–222.

[4] B. Chazelle and J. Matoušek, *On linear-time deterministic algorithms for optimization problems in fixed dimension*, in Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York and SIAM, Philadelphia, PA, 1993, pp. 281–290.

[5] A. Condon, *The complexity of stochastic games*, Information & Computation, 96(2) (1992), pp. 203–224.

[6] G. B. Dantzig, *Maximization of a linear function of variables subject to linear inequalities*, in Activity Analysis of Production and Allocation, T. C. Koopman (ed.), Cowles Commission Monograph 13, Wiley, New York, 1951, pp. 339–347.

[7] M. E. Dyer, *On a multidimensional search technique and its application to the euclidean one-center problem*, SIAM J. Comp., 15 (1986), pp. 725–738.

[8] ———, *A class of convex programs with applications to computational geometry*, in Proc. 8th Annual Symposium on Computational Geometry, ACM, New York, 1992, pp. 9–15.

[9] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.

[10] D. H. Greene and D. E. Knuth, *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, MA, 1981.

[11] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[12] G. Kalai, *A subexponential randomized simplex algorithm*, in Proc. 24th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1992, pp. 475–482.

[13] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[14] L. G. Khachiyan, *Polynomial algorithm in linear programming*, U.S.S.R. Comput. Math. and Math. Phys., 20 (1980), pp. 53–72.

[15] V. Klee and P. Kleinschmidt, *The d-step conjecture and its relatives*, Math. Operations Research, 12 (1987), pp. 718–755.

[16] V. Klee and G. J. Minty, *How good is the simplex algorithm*, in Inequalities III, O. Shisha (ed.), Academic Press, New York, 1972, pp. 159–175.

[17] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Information & Computation, to appear.

[18] N. Megiddo, *Linear programming in linear time when the dimension is fixed*, J. Assoc. Comput. Mach., 31 (1984), pp. 114–127.

[19] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, in Proc. 8th Annual Symposium on Computational Geometry, ACM, New York, 1992, pp. 1–8.

[20] V. T. Rajan, *Optimality of the Delaunay triangulation in $\mathbb{R}^d$*, in Proc. 7th Annual Symposium on Computational Geometry, ACM, New York, 1991, pp. 357–363.

[21] A. Schrijver, *Theory of linear and integer programming*, Wiley, New York, 1986.

[22] R. Seidel, *Low-dimensional linear programming and convex hulls made easy*, Discr. Comp. Geom., 6 (1991), pp. 423–434.

[23] M. Sharir and E. Welzl, *A combinatorial bound for linear programming and related problems*, in Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 577, Springer-Verlag, Berlin, 1992, pp. 569–579.

[24] K. Sekitani and Y. Yamamoto, *A recursive algorithm for finding the minimum norm point in a polytope and a pair of closest points in two polytopes*, Math. Programming, to appear.

[25] E. Welzl, *Smallest enclosing disks (balls and ellipsoids)*, in New Results and New Trends in Computer Science, H. Maurer (ed.), Lecture Notes in Computer Science 555, Springer-Verlag, Berlin, 1991, pp. 359–370.

[26] P. Wolfe, *The simplex method for quadratic programming*, Econometrica, 27 (1959), pp. 382–398.