

# The Domination Heuristic for LP-type Problems

Taras Galkovsky  
Taras Shevchenko University  
Kyiv, Ukraine  
galkovsky@gmail.com

Bernd Gärtner  
ETH Zürich  
Zürich, Switzerland  
gaertner@inf.ethz.ch

Bogdan Rublev  
Taras Shevchenko University  
Kyiv, Ukraine  
b\_v@univ.kiev.ua

## Abstract

Certain geometric optimization problems, for example finding the smallest enclosing ellipse of a set of points, can be solved in linear time by simple randomized (or complicated deterministic) combinatorial algorithms. In practice, these algorithms are enhanced or replaced with heuristic variants that are faster but do not come with a theoretical runtime guarantee.

In this paper, we introduce a new speed-up heuristic that can easily be integrated into the known linear-time algorithms, without decreasing their worst-case performance. The heuristic can actually be defined for every problem in the well-known abstract class of LP-type problems; its effectiveness in practice depends on whether and how fast the heuristic can be implemented for the specific problem at hand, and on whether the input distribution is favorable.

We provide test results showing that for two concrete problems, the new heuristic may lead to significant speedups compared to state-of-the-art implementations that are available in the Computational Geometry Algorithms Library CGAL.

## 1 Introduction

The first (expected) linear-time algorithm for the problem of finding the smallest enclosing ellipsoid of a set of points in  $d$ -dimensional space is due to Welzl [15] and was motivated by an earlier randomized algorithm for linear programming by Seidel [14].

Here is the essential idea of the algorithm (described for dimension 2): to find the smallest enclosing ellipse  $E(P)$  of a set  $P$  of  $n$  points, choose a point  $p \in P$  at random, and recursively compute  $E := E(P \setminus \{p\})$ . If  $p$  happens to lie inside  $E$ , we are done; otherwise, we can conclude that  $p$  must be on the boundary of  $E(P)$ , and we therefore recursively compute the smallest enclosing ellipse  $E(P \setminus \{p\}, \{p\})$  of  $P$  that has  $p$  on the boundary. A generic recursive call computes  $E(P, R)$  for a set  $R$  of boundary points, where the problem is easy for  $|R| = 5$ , since an ellipse is uniquely determined by 5 points.

Despite its backtracking flavor, this algorithm

achieves expected runtime  $O(n)$  for any fixed dimension  $d$ . The key observation is that the probability for  $p$  not being contained in  $E$  is at most  $c(d)/n$ , where  $c(d)$  is a function depending solely on  $d$ :  $c(d) = d(d+3)/2$ . This upper bound implies that the computation of  $E(P \setminus \{p\}, R \cup \{p\})$  happens only with small probability. The same algorithm also works for the similar problem of finding the smallest enclosing *ball* of a set of points.

Concerning practical performance, it was already observed by Welzl in his original paper [15] that the above algorithm is rather slow. Welzl suggested a *move-to-front* variant of his algorithm that performs much better in practice but does not come with a theoretical runtime bound anymore, short of a trivial bound.

Further speedups in practice can be achieved by heuristics that take the geometry into account (in contrast, *move-to-front* is purely combinatorial). For example, the fast algorithm for computing smallest enclosing balls that is available at <http://www.inf.ethz.ch/personal/gaertner/miniball.html> uses Welzl's *move-to-front* variant for small problem instances, and a *farthest-point* heuristic for large instances. Given a candidate ball that is not yet enclosing, the algorithm searches for the “worst outlier” and uses it to determine a better candidate ball [5].

In recent work, Osipov et al. have further improved the practical performance of this algorithm by enhancing it with a *filtering approach* [11]. They show that if the “worst outlier” has distance  $\varepsilon$  to the current ball, then one can remove all the points from further consideration that are inside the current ball *and* have distance  $\varepsilon$  or more from its boundary.

The disadvantage of such geometric heuristics is that they do not generalize to more abstract settings. One particular such setting is that of *LP-type problems* due to Matoušek, Sharir and Welzl [9]. Essentially, an LP-type problem is an optimization problem over an abstract set of constraints  $H$  (points in the case of smallest enclosing ellipsoids and balls). Given the optimal solution subject to a subset  $G$  of the constraints, the LP-type framework requires a test whether this solu-

tion violates a given constraint  $h \in H \setminus G$  (test whether a point is outside the smallest enclosing ellipse/ball of a subset); there is no abstract notion of “how much” a constraint is violated.

The class of LP-type problems includes a large number of practically relevant geometric optimization problems [9]. Given just two problem-specific primitive operations, the randomized algorithm of Matoušek, Sharir and Welzl can be used to solve every LP-type problem in expected linear time (linear in the number of constraints  $H$ , given that the *combinatorial dimension*  $\delta$  of the problem ( $\delta = 5$  for ellipses) is fixed; see [9] for details).

The generic LP-type algorithm is actually an improvement of the above algorithms of Seidel[14] and Welzl[15]; it still works for smallest enclosing ellipsoids and balls but also (and this is its main strength) for other problems with somewhat less structure.

Just like Welzl’s algorithm, the generic LP-type algorithm can typically be improved in practice, at the cost of losing the theoretical guarantees. For smallest enclosing ellipsoids and balls, this has independently been done by Lyashko, Petunin and Rublev[12, 13, 8] and Friedman [2]; on the level of LP-type problems, both approaches coincide. Here is the overview of their approach (described for ellipses in dimension 2): maintain a set  $B$  of at most five points along with their smallest enclosing ellipse  $E$ . Then iterate through the remaining points, and whenever a point  $p$  outside  $E = E(B)$  is found, update  $E$  to  $E(B \cup \{p\})$  and reset  $B$  to the set of boundary points of this new ellipse. Computation of the smallest enclosing ellipsoid  $E(B \cup \{p\})$  can be done for example with Welzl’s algorithm as a subroutine. As long as at least one update took place during the iteration, continue with another iteration through the points. Upon termination, this method has computed the smallest enclosing ellipse of the whole point set.

In this paper, we show that this method can be further improved for many inputs, by using a filtering approach related to the one of Osipov et al. [11]. Namely, before a point  $p$  is tested for containment in  $E = E(B)$ , we test whether it is contained in the convex hull of the set  $B$ . If so, we can be sure that  $p$  will not contribute to the final ellipse and can therefore be removed altogether. The benefit is that all subsequent iterations will not see  $p$  anymore. We call this the *convex hull heuristic*, and it is obviously valid for all convex bounding volumes.

Whether this is effective largely depends on the distribution of input points, and on the cost of the *violation test* “ $p \notin E(B)$ .” At this point, we would like to make it clear that we are mainly interested

in *exact* computations that deliver the mathematically correct result and not an approximation of it. This is exactly the realm of CGAL, the Computational Geometry Algorithms Library ([www.cgal.org](http://www.cgal.org)). In case of smallest enclosing ellipses, the exact test “ $p \notin E(B)$ ” is fairly involved, since  $E(B)$  may have irrational coordinates and is therefore not explicitly computed.

Even if violation tests are relatively cheap (like for smallest enclosing balls of points or other balls), it may pay off to switch on the convex hull heuristic. In any case, the input distribution must be favorable in order to realize any speedups at all.

From a theoretical point of view, the most interesting feature of the convex hull heuristic—and this is what sets our filtering approach apart from the one by Osipov et al. [11]—is that it can be generalized to arbitrary LP-type problems, and that it can be incorporated into existing algorithms without affecting the theoretical guarantees. We will get to this in Section 3.3. In a nutshell, the generalization (which we call the *domination heuristic*) is this: a constraint  $h \in H$  is called *dominated* by  $B \subseteq H$  if the following holds: whenever  $h$  is violated by the optimal solution subject to some subset  $G$  of constraints, then this solution also violates an element of  $B$ . Under this definition, a point inside the convex hull of other points is dominated by these points with respect to smallest enclosing ellipses, say.

The domination heuristic (which is strictly combinatorial) simply throws away constraints that have been identified as being dominated during the computations. It depends on the concrete LP-type problem whether an efficient domination test is available at all, and whether it will actually remove constraints. But in any case, the expected asymptotic runtime will not increase.

The remainder of the paper is organized as follows. In Section 2, we review the algorithms of Welzl [15] and of Rublev [12, 13] for smallest enclosing ellipsoids, and we enhance them with the convex hull heuristic; we provide test results that show the performance gain (or sometimes loss) under various input distributions.

In Section 3, we present the generalization to LP-type problems. We review the generic (recursive) LP-type algorithms of Matoušek, Sharir and Welzl[9], and the simple iterative one resulting from generalizing the methods of Rublev[12, 13] and Friedman [2]. We show that even when the algorithm of Matoušek, Sharir and Welzl is equipped with the domination heuristic, the expected runtime of  $O(n)$  still holds.

As an example of practical usefulness and ease of use of the general domination heuristic, we describe its application to the problem of finding the smallest enclosing sphere of spheres in Section 4.

## 2 Smallest Enclosing Ellipses

We start from Welzl’s algorithm (see Algorithm 2.1 [15]) for computing smallest enclosing ellipses, written down formally. As a subroutine, it needs to solve the constant-size problem of computing the smallest ellipse  $E_0(S)$  with a set  $S$  of at most 5 points on its boundary. The algorithm returns a *basis*  $S$ ,  $R \subseteq S \subseteq P \cup R$  with the property that  $E_0(S) = E(P, R)$ . To compute  $E(P)$ , we call the algorithm with  $R = \emptyset$ .

ALGORITHM 2.1. Welzl’s algorithm (also used as function in other algorithms)

**Data:** Disjoint sets  $P$  and  $R$  of points,  $|R| \leq 5$   
**Result:** Basis of the smallest ellipse that contains  $P$  and has  $R$  on its boundary

```

SmEll( $P, R$ ) begin
   $Q \leftarrow \emptyset$ ;
   $S \leftarrow R$ ;
   $E \leftarrow E_0(S)$ ;
  foreach  $p \in P$  in random order do
    if  $p \notin E$  then
       $S \leftarrow \text{SmEll}(Q, R \cup \{p\})$ ;
       $E \leftarrow E_0(S)$ ;
    end
     $Q \leftarrow Q \cup \{p\}$ ;
  end
  return  $S$ ;
end

```

One problem that leads to poor performance of this algorithm in practice is that the recursive call throws away the ellipse computed so far and starts from scratch, only exploiting the additional information that  $p$  has to lie on the boundary. Rublev’s algorithm [12, 13] tries to reuse the ellipse so far: whenever a point is found to be outside, the working basis  $S$  and the corresponding ellipse  $E_0(S)$  are updated to cover the basis of previous ellipse and the new point  $p$ .

The pseudocode is given in Algorithm 2.2. The proof of correctness can be found in [12, 13].

**2.1 Computational Complexity** For both algorithms, the runtime is dominated by the number of *violation tests* “ $p \notin E$ ”. For Algorithm 2.1, the expected number of such tests is  $O(n)$ , where  $n = |P|$  [15]. We have no good bound for Rublev’s algorithm.

One might be tempted to think that every iteration of the main loop in Rublev’s algorithm adds one element of the final basis to the working basis  $S$ , but this is not true. In fact, experiments show that points from the final basis could be added and removed from the working basis several times. It remains an open

problem whether there exists any nontrivial bound on the number of outer loop iterations.

ALGORITHM 2.2. Rublev’s algorithm

**Data:** Point set  $P$   
**Result:** Smallest enclosing ellipse of set  $P$

```

begin
   $S \leftarrow \emptyset$ ;
   $E \leftarrow \emptyset$ ;
  done  $\leftarrow$  false;
  while not done do
    done  $\leftarrow$  true;
    foreach  $p \in P$  do
      if  $p \notin E$  then
         $S \leftarrow \text{SmEll}(S, \{p\})$ ;
         $E \leftarrow E_0(S)$ ;
        done  $\leftarrow$  false;
      end
    end
  end
  return  $E$ 
end

```

**2.2 The Convex Hull Heuristic** Like in many other geometric algorithms, correct results can only be guaranteed for Algorithms 2.1 and 2.2 if multiprecision arithmetic is used. In fact, the package `Min_ellipse_2` in the CGAL library implements the (faster) move-to-front variant of Algorithm 2.1 [15], using exact arithmetic. It turns out that the violation tests “ $p \notin E$ ” are the major bottleneck. This is on the one hand due to the fact that violation tests are the most frequent operations, but on the other hand, exact violation tests are not easy because the involved ellipse  $E$  will in general have irrational coordinates [6].

Therefore, reducing the necessary number of violation tests will lead to an immediate speedup of all algorithms (by Welzl and Rublev). Here is the simple but crucial observation:

**PROPERTY 2.1.** *If during Algorithm 2.1 or 2.2, the considered point  $p$  is contained in the convex hull of the working basis  $S$ , then  $p$  can be removed from further consideration.*

*Proof.* The convex hull of  $S$  is the smallest convex set that contains all points from  $S$ . The ellipse  $E(P \setminus \{p\}, R)$  (for Algorithm 2.2,  $R = \emptyset$ ) is some convex set that contains  $S$  (because of  $S \subseteq (P \setminus \{p\}) \cup R$ ), and therefore it also contains  $p$ . It follows that  $E(P \setminus \{p\}, R) = E(P, R)$ , meaning that  $p$  can be ignored without changing the output of the algorithm.

The convex hull computation introduces at most constant overhead, since it involves at most five points; it has the potential, though, of removing many points. The resulting variant of Welzl’s method is Algorithm 2.3.

ALGORITHM 2.3. Welzl’s algorithm with convex hull heuristic

**Data:** Disjoint sets  $P$  and  $R$  of points,  $|R| \leq 5$

**Result:** Basis of the smallest ellipse that contains  $P$  and has  $R$  on its boundary

```

SmEll_H( $P, R$ ) begin
   $Q \leftarrow \emptyset$ ;
   $S \leftarrow R$ ;
   $E \leftarrow E_0(S)$ ;
   $H \leftarrow \text{ConvexHull}(S)$ ;
  foreach  $p \in P$  in random order do
    if  $p \notin H$  then
      if  $p \notin E$  then
         $S \leftarrow \text{SmEll}_H(Q, R \cup \{p\})$ ;
         $E \leftarrow E_0(S)$ ;
         $H \leftarrow \text{ConvexHull}(S)$ ;
      end
       $Q \leftarrow Q \cup \{p\}$ ;
    end
  end
  return  $S$ ;
end

```

Obviously, there are inputs for which Algorithm 2.3 will not improve over Algorithm 2.1, for example points in convex position. For points randomly distributed within a square or disk, through, major savings can be expected (see the benchmark section below).

In the same way, we can enhance Rublev’s Algorithm 2.2 with the convex hull heuristic, resulting in Algorithm 2.4.

**2.3 Benchmarks** In this section we provide experimental results that show how the convex hull heuristic performs for the problem of computing smallest enclosing ellipses. As far as theoretical guarantees are concerned, we want to make it clear that in the worst case (points in convex position), no savings are possible at all, and that the actual runtime may even increase (not more than by a constant factor, though, see Section 3). But the results below show that the heuristic performs much better if less points are on the convex hull.

We therefore study the behavior of the heuristic on various input distributions that essentially differ by the fraction of points that are on the convex hull, and we try to see whether there is indeed a correlation between this fraction and the runtime in practice.

Each benchmark set consists of  $N = 10,000$  points, generated from the following random distributions: uniform in the unit square, uniform on some integer rectangular grid (many duplicates), uniform in the unit disk, uniform in an annulus of tiny width (both inner and outer radii are within  $10^{-8}$  of 1). Also small and large synthetic *exactly* cocircular sets were used to check the behavior of the algorithms in extreme (mostly theoretical) cases. The classes of input sets in the mentioned order form a sequence with increasing relative size of the convex hull.

ALGORITHM 2.4. Rublev’s algorithm with convex hull heuristics

**Data:** Point set  $P$

**Result:** Smallest enclosing ellipse of set  $P$

```

 $S \leftarrow \emptyset$ ;
 $E \leftarrow \emptyset$ ;
 $H \leftarrow \emptyset$ ;
done  $\leftarrow$  false;
while not done do
  done  $\leftarrow$  true;
  foreach  $p \in P$  do
    if  $p \in H$  then
       $P \leftarrow P \setminus \{p\}$ ;
    else
      if  $p \notin E$  then
         $S \leftarrow \text{SmEll}(S, \{p\})$ ;
         $E \leftarrow E_0(S)$ ;
         $H \leftarrow \text{ConvexHull}(S)$ ;
        done  $\leftarrow$  false;
      end
    end
  end
end
return  $E$ 

```

For testing, we used a state-of-the-art implementation of Welzl’s algorithm 2.1 included in the `Min_ellipse_2` package of the CGAL library. Actually, that implementation differs from Algorithm 2.1 by the use of the move-to-front heuristic. In the same way, we have also equipped our new Algorithm 2.3 with the move-to-front heuristic. Our implementation therefore differs only in the added convex hull heuristic.

On the other hand, the lesser known Rublev-Friedman algorithm 2.2 and its variant with convex hull heuristic 2.4 were implemented for use in the CGAL environment, sharing all primitive operations with the `Min_ellipse_2` package. Thus we have ensured that all algorithms being tested were implemented in a homogeneous way.

In Table 1 we provide the runtimes for each of the implementations on our classes of input point sets.

Each value in the table is the runtime of the respective implementation, averaged over 100 random sets (the same sets were used for different implementations). Since absolute runtimes are hard to compare, the table is organized as follows: only the absolute runtime of the base implementation (the `Min_ellipse_2` package of CGAL) is provided; for the other implementations we specify the relative speedup multiplier (the bigger the multiplier, the faster is the implementation).

distribution	in unit square	on grid	in unit disk	in unit annulus	small cocircular	large cocircular
CGAL	22.0s	2.7s	8.2s	7.2s	0.27s	0.59s
CGAL+h	8.3x	2.8x	2.3x	0.9x	0.7x	0.7x
Rublev	2.3x	0.6x	1.8x	1.7x	0.5x	0.5x
Rublev+h	10.7x	1.5x	2.7x	1.5x	0.4x	0.4x

Table 1: Smallest enclosing ellipse implementations (“+h” means that heuristic was applied to the corresponding base algorithm)

As expected, the results of Table 1 show that no globally best solution exists. Both implementations with convex hull heuristic applied perform well against their base counterparts on sets with low convex hull size. On the other hand, when most (or even all) of the points lie on the convex hull, the heuristic slows down the implementation. Note that the order of classes of input sets (sorted by increasing relative size of convex hull) correlates with the order of speedup of the implementations when the convex hull heuristic is applied. There is no clear winner as far as the choice of the base implementation (CGAL vs. Rublev) is concerned.

Despite its non-effectiveness in the worst case, the results clearly show that it is possible to achieve considerable speedups by applying the convex hull heuristic to classical algorithms for solving the smallest enclosing ellipse problem. This specifically holds for any input point set that is generated randomly from a non-zero volume. It would be interesting to see whether for such random inputs, one can even give theoretical guarantees on the expected number of points that are being removed by the convex hull heuristic.

We conclude from our experiments that for the problem of computing smallest enclosing ellipses, the convex hull heuristic pays off, unless the input is “almost” in convex position.

### 3 An Abstract Framework

This section discusses how the convex hull heuristic described in the previous section can be generalized to the whole abstract class of *LP-type problems* [7, 10]. This class includes the smallest enclosing ellipse problem but also many other geometric optimization problems [9].

**3.1 LP-type Problems** Let us consider abstract minimization problems specified by pairs  $(H, w)$ , where  $H$  is a finite set, and  $w : 2^H \rightarrow \mathcal{W} \cup \{-\infty\}$  is a function with values in a linearly ordered set  $(\mathcal{W} \cup \{-\infty\}, \leq)$ , the value  $-\infty$  (standing for ‘unbounded’) preceding all values in  $\mathcal{W}$ . The elements of  $H$  are called *constraints*, and for  $G \subseteq H$ ,  $w(G)$  is called the (optimal) *value* of  $G$ . The goal is to compute the value  $w(H)$  of  $H$ , using certain primitive operations to which we get below.

$(H, w)$  is called an LP-type problem if the following two conditions are satisfied.

**CONDITION 3.1. (Monotonicity)** For any  $F, G$  with  $F \subseteq G \subseteq H$ , we have  $w(F) \leq w(G)$ .

**CONDITION 3.2. (Locality)** For any  $F \subseteq G \subseteq H$  with  $-\infty \neq w(F) = w(G)$  and any  $h \in H$ ,  $w(G) < w(G \cup \{h\})$  implies that also  $w(F) < w(F \cup \{h\})$ .

If  $w(G) < w(G \cup \{h\})$ , we say that constraint  $h$  is *violated* by  $G$ . Monotonicity is a natural requirement when we are talking about minimization problems: adding more constraints cannot decrease the optimal value. Locality essentially says that optimality can be tested locally: an equivalent formulation is that whenever  $w(F) < w(G)$  for  $F \subseteq G$ , then there is  $h \in G$  such that the value of  $F$  can locally be increased by switching to  $F \cup \{h\}$ .

When we write the smallest enclosing ellipse problem as an LP-type problem, the set of constraints is the set of input points, and the value of a subset is the volume of its smallest enclosing ellipse (value  $-\infty$  arises if the affine hull of the subset is not the whole plane). A constraint (point) is violated by a subset if it lies outside its smallest enclosing ellipse. Uniqueness of the smallest enclosing ellipse [15] is easily seen to imply the locality property.

The fact that makes Welzl’s method efficient is that the smallest enclosing ellipse is determined by no more than five points. The abstract counterpart is the *combinatorial dimension* of an LP-type problem. A *basis* is a subset  $B \subseteq H$  such that  $w(B \setminus \{h\}) < w(B)$  for all  $h \in B$ . This means, a basis is an inclusion-minimal set defining a certain value. Now, the maximum cardinality of any basis is called the *combinatorial dimension* of  $(H, w)$ , and is denoted by

$\delta = \delta(H, w)$ . For smallest enclosing ellipse problem we have  $\delta = 5$ .

*Solving* an LP-type problem means to find a basis  $B$  of  $H$ , a basis  $B$  such that  $w(B) = w(H)$ . From such a basis, the value  $w(G)$  is usually easy to compute. We assume that the following primitive operations are available:

**Violation test** Given a basis  $B$  and a constraint  $h \notin B$ , decide whether  $w(B) < w(B \cup \{h\})$

**Basis computation** Given a basis  $B$  and a violating constraint  $h$ , compute a basis of  $B \cup \{h\}$ .

**3.2 LP-type Algorithms** In the abstract setting of LP-type problems, there is no notion of “fixing points on the boundary” like it is employed in Welzl’s algorithm 2.1. Consequently, this algorithm does not generalize to the LP-type setting. But Rublev’s Algorithm 2.2 has an immediate LP-type counterpart, see Algorithm 3.1.

ALGORITHM 3.1. Rublev/Friedman algorithm for LP-type problems

**Data:** LP-type problem  $(H, w)$  specified by primitive operations

**Result:** Basis of  $H$

```

begin
  B ← ∅;
  done ← false;
  while not done do
    done ← true;
    foreach h ∈ H do
      if w(B) < w(B ∪ {h}) then
        B ← basis(B ∪ {h});
        done ← false;
      end
    end
  end
  return B
end

```

But there is also an algorithm in the LP-type setting that comes with theoretical runtime guarantees, and this is Algorithm 3.2 due to Matoušek, Sharir and Welzl[9]. The algorithm is randomized and recursive; given a pair  $(G, B)$ , where  $B \subseteq G$  is a basis (not necessarily of  $G$  yet), the algorithm computes a basis of  $G$ . To start off, it requires some initial basis ( $B = \emptyset$  will do).

The following has been shown by Matoušek, Sharir and Welzl[9]

**THEOREM 3.1.** *Given an LP-type problem  $(H, w)$  with  $|H| = n$  and fixed combinatorial dimension  $\delta$ , Algo-*

*rihm 3.2 requires an expected number of  $O(n)$  primitive operations to solve it.*

If  $\delta$  is fixed, each primitive operation takes constant time (even if done in a brute-force fashion), so that the algorithm takes expected linear time. For a number of concrete LP-type problems, this was the first known linear-time algorithm. The best known dependence on  $d$  is *subexponential* (meaning that the logarithm is sublinear) [9, 4].

ALGORITHM 3.2. MSW( $G, B$ ) algorithm for LP-type problems

**Data:** LP-type problem  $(H, w)$  specified by primitive operations;  $G \subseteq H$ ;  $B \subseteq G$  a basis

**Result:** Basis of  $G$

```

MSW(G, B) begin
  F ← B;
  foreach h ∈ G \ B in random order do
    F ← F ∪ {h};
    if w(B) < w(B ∪ {h}) then
      B ← basis(B ∪ {h});
      B ← MSW(F, B)
    end
  end
  return B
end

```

**3.3 The Domination Heuristic** In this section, we want to generalize the convex hull heuristic of Section 2.2 to the abstract setting of LP-type problems. Assume that a point  $p$  is in the convex hull of a set  $S$ . The convex hull heuristic works for the following reason: whenever  $p$  is outside the smallest enclosing ellipse of some subset, then there is also some point from  $S$  that is outside (an immediate consequence of convexity of ellipses). This leads to the abstract concept of *domination*.

**DEFINITION 3.1.** *Let  $(H, w)$  be an LP-type problem,  $B \subseteq H$  and  $h \in H$ .  $h$  is called dominated by  $B$  if the following holds: for every set  $G \subseteq H$  such that  $w(G) < w(G \cup \{h\})$ , there exists an element  $h' \in B$  such that  $w(G) < w(G \cup \{h'\})$ .*

Again, it easily follows that dominated elements can be removed without changing the result.

**LEMMA 3.1.** *Let  $(H, w)$  be an LP-type problem,  $h \in G \subseteq H$ . If  $h$  is dominated by some set  $B \subseteq G \setminus \{h\}$ , then  $w(G) = w(G \setminus \{h\})$ .*

*Proof.* Since  $B \subseteq G \setminus \{h\}$ , we have  $G \setminus \{h\} \cup \{h'\} = G \setminus \{h\}$  for all  $h' \in B$ , hence  $w(G \setminus \{h\} \cup \{h'\}) = w(G \setminus \{h\})$ . This means that no element of  $B$  violates  $G \setminus \{h\}$ , and

since  $h$  is dominated by  $B$ ,  $h$  cannot violate  $G \setminus \{h\}$ , either.

This lemma implies the correctness of Algorithm 3.3, which is Algorithm 3.2 enhanced with the *domination heuristic*. Here,  $\text{dom}(h, B)$  is a shorthand for “ $h$  is dominated by  $B$ ”.

In a similar way we can also apply the domination heuristic to the Rublev-Friedman algorithm 3.1 for LP-type problems. The resulting modified method is Algorithm 3.4.

In order to be able to execute Algorithm 3.3, we need to stipulate a new primitive, namely  $\text{dom}(h, B)$ . Having an exactly such primitive may be difficult; even in the case of smallest enclosing ellipses, we do not get this: the fact that  $p$  is in the convex hull of  $S$  is a *sufficient* condition for  $p$  being dominated by  $S$ , but not a necessary condition. But the following conservative primitive can be implemented and is enough to ensure correctness of Algorithm 3.3.

**Domination test** Given a basis  $B$  and a constraint  $h \notin B$ ,  $\text{dom}(h, B)$  returns *true* only if  $h$  is dominated by  $B$ ; if  $\text{dom}(h, B)$  returns *false*,  $h$  may or may not be dominated by  $B$ .

This primitive can of course always be implemented (simply return *false*), but it makes sense only if it can actually “recognize” some dominations.

ALGORITHM 3.3. MSW\_D( $G, B$ ) algorithm for LP-type problems, with domination heuristic

**Data:** LP-type problem  $(H, w)$  specified by primitive operations;  $G \subseteq H$ ;  $B \subseteq G$  a basis  
**Result:** Basis  $B$  such that  $w(B) = w(G)$

```
MSW_D( $G, B$ ) begin
   $F \leftarrow B$ ;
  foreach  $h \in G \setminus B$  in random order do
    if not  $\text{dom}(h, B)$  then
       $F \leftarrow F \cup \{h\}$ ;
      if  $w(B) < w(B \cup \{h\})$  then
         $B \leftarrow \text{basis}(B \cup \{h\})$ ;
         $B \leftarrow \text{MSW\_D}(F, B)$ ;
      end
    end
  end
  return  $B$ 
end
```

**3.4 Complexity Analysis** We want to argue that Algorithm 3.3 still requires only  $O(n)$  primitive operations for constant combinatorial dimension. This seems intuitively clear (how can the removal of a dominated

constraint generate more work?), but we are not aware of any direct argument along these lines. After all, the removal of a constraint may have the effect that the algorithm “goes along a different path” in the future, and this path may take longer.

In order to argue formally, we have to go into the analysis of Algorithm 3.2 and show that this analysis still works for Algorithm 3.3. The following analysis is adapted from [3]. It does not yield the best possible dependence on  $\delta$ , but in return it is much simpler.

ALGORITHM 3.4. Rublev/Friedmann algorithm for LP-type problems, with domination heuristic

**Data:** Set of constraints  $H$   
**Result:** Basis  $B$  violated by none of the constraints from  $H$

```
begin
   $B \leftarrow \emptyset$ ;
  done  $\leftarrow$  false;
  while not done do
    done  $\leftarrow$  true;
    foreach  $h \in H$  do
      if  $h$  is dominated by  $B$  then
         $H := H \setminus \{h\}$ ;
      else
        if  $h$  is violated by  $B$  then
           $B \leftarrow \text{basis}(B, h)$ ;
          done  $\leftarrow$  false;
        end
      end
    end
  end
  return  $B$ 
end
```

**Hidden dimension.** Given a pair  $(G, B)$ , where  $B \subseteq G$  is a basis (not necessarily of  $B$  yet), we call  $h \in B$  *enforced* in  $(G, B)$  if  $w(B) > w(G \setminus \{h\})$ , i.e. if  $B$  already has higher value than  $G \setminus \{h\}$ . If  $h$  is enforced in  $(G, B)$ , this implies that  $h$  will be contained in *every* basis  $B'$  encountered during the call to  $\text{MSW\_D}(G, B)$ . Here we use the fact that  $w(B') \geq w(B)$  for any such basis.

Because of  $h \in B$ , the number of enforced elements is bounded by the combinatorial dimension. This motivates the following

DEFINITION 3.2. *The hidden dimension of  $(G, B)$  is  $\delta$  (combinatorial dimension of the LP-type problem) minus the number of enforced elements in  $(G, B)$ .*

Let  $T(n, k)$  denote the maximum expected number of violation tests during a call to  $\text{MSW\_D}(G, B)$ , where  $G$

has size at most  $n$ , and  $(G, B)$  has hidden dimension at most  $k$ . It is not hard to see that this number asymptotically dominates the expected runtime for fixed  $\delta$ . We start with a simple bound for small  $n$ .

LEMMA 3.2. For  $n \leq \delta + 1$  and for all  $k$ ,

$$T(n, k) \leq 2^{k+2}.$$

*Proof.* Let “ $w(B_t) < w(B_t \cup \{h_t\})$ ”,  $t = 1, 2, \dots$  be the sequence of violation tests in chronological order during some run of the algorithm. We know that  $B_t \supseteq E$  for all  $t$  where  $E$  is the set of enforced elements. Since  $|E| \geq \delta - k$ , the number of distinct sets  $B_t \cup \{h_t\}$  that can occur is bounded by  $2^{k+1}$ . Let us consider a set that repeats, i.e.

$$B_t \cup \{h_t\} = B_s \cup \{h_s\},$$

where we let  $s > t$  be the first index for which  $B_t \cup \{h_t\}$  repeats. Inspection of the algorithm shows that no basis  $B$  is ever tested twice with the same constraint  $h$ , so we have  $B_t \neq B_s$ . We can therefore charge the repetition to the basis  $B_t$ . In this way, every basis is charged at most once (since a basis itself never repeats), so the total number of repetitions is bounded by  $2^k$ . The length of the sequence is then at most  $2^{k+1} + 2^k \leq 2^{k+2}$ .

Here is the actual  $O(n)$  bound.

LEMMA 3.1. For  $n \geq \delta + 1$  and for all  $k$ ,

$$T(n, k) \leq 2^{k+2}(n - \delta).$$

*Proof.* The key ingredient is the following recurrence relation. For  $n > \delta$ , we have

$$(3.1) \quad \begin{aligned} T(n, k) &\leq T(n-1, k) + 1 \\ &+ \frac{1}{n-\delta} \sum_{\ell=1}^{\min(k, n-\delta)} T(n, k-\ell). \end{aligned}$$

To see this, consider any pair  $(G, B)$  with  $|G| \leq n$  and hidden dimension at most  $k$  for which  $T(n, k)$  is attained. W.l.o.g. we may assume that  $|G| = n$ . With  $h \in G \setminus B$  being the last element considered in  $\text{MSW}_D(G, B)$ , we can imagine the algorithm being first called recursively for the pair  $(G \setminus \{h\}, B)$ . Since elements that are enforced in  $(G, B)$  are also enforced in  $(G \setminus \{h\}, B)$ , the expected number of violation tests in this recursive call is bounded by  $T(n-1, k)$ . Then there is one violation test on the ‘top level’ that accounts for the ‘1’.

To estimate the expected number of violation tests in the recursive call  $\text{MSW}_D(F, B)$  (if it takes place at

all), let us first fix an arbitrary basis  $C$  of  $G$ , i.e. a set satisfying  $w(C) = w(G)$  and  $|C| \leq \delta$ . For  $h \notin C$ , no further recursive call will take place, since then  $w(G) \geq w(G \setminus \{h\}) \geq w(C) = w(G)$  by monotonicity. This means,  $h \in G \setminus B$  is a candidate for triggering a recursive call *only* if  $h \in C$ . The number of  $h$  in  $C$  is at most  $\delta$ , but since at least  $\delta - k$  elements are enforced (and hence lie in  $B \cap C$ ), the candidate set shrinks to size at most  $k$ . (If  $k > |G - B|$ , we certainly have at most  $|G - B|$  candidates.) Let us order the  $c \leq \min(k, |G - B|)$  candidates in such a way that

$$w(G \setminus \{h_1\}) \leq w(G \setminus \{h_2\}) \leq \dots \leq w(G \setminus \{h_c\}).$$

If  $h = h_\ell$ , the basis  $B'$  employed in the second recursive call  $\text{MSW}_D(F, B)$  (if it takes place) satisfies

$$w(B') > w(G \setminus \{h_\ell\}) \geq w(G \setminus \{h_i\}), \quad i \leq \ell.$$

This in turn implies that  $h_1, \dots, h_\ell$  are enforced in  $(G, B')$  and therefore also in  $(F, B')$ . Since all previously enforced elements stay enforced, the hidden dimension of  $(F, B')$  is at most  $k - \ell$ . Moreover,  $h = h_\ell$  with probability  $1/|G - B|$ , so we get that the expected number of violation tests in  $\text{MSW}_D(F, B)$  with  $B = B'$  is bounded by

$$\frac{1}{|G - B|} \sum_{\ell=1}^{\min(k, |G - B|)} T(n, k - \ell),$$

which by monotonicity of  $T(n, k)$  in  $k$  is at most

$$\frac{1}{n - \delta} \sum_{\ell=1}^{\min(k, n - \delta)} T(n, k - \ell).$$

This proves the recurrence relation (3.1).

Note that the only difference to the ‘classical’ analysis is that a potentially much smaller subset  $F$  (resulting from the removal of dominated constraints) replaces  $G$  in the recursive call; but since  $|F| \leq n$  and  $F \subseteq G$ , the original bounds on the problem size and the hidden dimension still apply.

Using (3.1) together with Lemma 3.2, we can conclude the proof by induction. For  $n = \delta + 1$ , the claimed bound follows from Lemma 3.2. For  $k = 0$ , (3.1) yields

$$\begin{aligned} T(n, 0) &\leq T(n-1, 0) + 1 \\ &\leq T(\delta + 1, 0) + n - \delta - 1 \\ &\leq n - \delta + 3 \leq 4(n - \delta). \end{aligned}$$

For  $n > \delta + 1, k > 0$ , we inductively get

$$T(n, k) \leq 2^{k+2}(n - \delta - 1) + 1$$

$$\begin{aligned}
& + \frac{1}{n-\delta} \sum_{\ell=1}^{\min(k, n-\delta)} 2^{k+2-\ell} (n-\delta) \\
& \leq 2^{k+2} (n-\delta-1) + 1 \\
& + \sum_{\ell=1}^{k+2} 2^{k+2-\ell} \\
& = 2^{k+2} (n-\delta-1) + 1 + 2^{k+2} - 1 \\
& = 2^{k+2} (n-\delta).
\end{aligned}$$

Summarizing, we obtain

**THEOREM 3.2.** *Given an LP-type problem  $(H, w)$  with  $|H| = n$  and fixed combinatorial dimension  $\delta$ , Algorithm 3.3 solves the problem in expected time  $O(n)$ .*

The analysis above can be improved so that the dependence on  $\delta$  becomes subexponential [9, 4]. This is not hard for *regular* LP-type problem, in which all bases have size exactly  $\delta$  [9]. To handle general LP-type problems, one needs a second algorithm, though [4].

#### 4 Smallest Enclosing Sphere of Spheres

So far, we have shown (Section 2.3) that the domination heuristic can be very effective for smallest enclosing ellipses, where it takes the form of the convex hull heuristic. In this final section, we want to examine another LP-type problem, namely finding the smallest enclosing sphere of a set of *spheres*. This problem is relevant for bounding volume heuristics, and it is theoretically interesting because surprisingly it can *not* be solved by Welzl’s algorithm [1]. For this problem, we need the general LP-type techniques.

Again, we test domination based on convex hull containment. Ideally, we would like to test whether a given sphere is contained in the convex hull of a set of other spheres, but this is not a cheap operation. Since we are allowed to use a conservative (but possibly less effective) domination test, we do the following: we test whether the sphere fits into the convex hull of the *centers* of the spheres in the current basis, by measuring the distance from the sphere center to the convex hull surface. This is a sufficient (but by far not necessary) condition for domination.

As for the smallest enclosing ellipse problem, our aim is to compare the implementations of all four different algorithms mentioned in the previous section. These are the ones by Matoušek, Sharir and Welzl (Algorithm 3.2) and its counterpart with domination heuristic applied (Algorithm 3.3), the Rublev-Friedman Algorithm 3.1 and its counterpart with domination heuristic applied, see Algorithm 3.4.

The CGAL library contains the package `Min_sphere_of_spheres_d` which provides an im-

plementation of Algorithm 3.2. By default, the CGAL implementation uses the geometric *farthest-first heuristic* on top of Algorithm 3.2 [1]. Like the move-to-front heuristic for Welzl’s algorithm, this speeds up Algorithm 3.2 in practice, but the theoretical runtime guarantees are lost.

Unlike for ellipses, we test the original Algorithm 3.2 and the farthest-first heuristic on top of it separately, the reason being that here we cannot beat the farthest-first heuristic by employing the domination heuristic (see the benchmarks below). We can beat Algorithm 3.2, though, and thus obtain the fastest practical algorithm with *provable* runtime guarantees.

The incorporation of the domination heuristic into the `Min_sphere_of_spheres_d` package was trivial. On the other hand, the Rublev-Friedman algorithm 3.1 and its variant with convex hull heuristic, Algorithm 3.4 were implemented for use in the CGAL environment, sharing all primitive operations with `Min_sphere_of_spheres_d` package. Thus we have ensured that all algorithms being tested were implemented in a homogeneous way.

**4.1 Benchmarks** The testing process follows essentially the same pattern as Section 2.3, but we need to choose the ball radii in addition. For our experiments we have chosen ball centers and ball radii independently. For the radii, we have used two kind of distributions: exponential and constant. For the centers, we have chosen the same distributions as in Section 2.3, with the obvious generalizations to 3 dimensions.

We use ball sets consisting of  $N = 10,000$  balls, with centers generated by the following random distributions in dimensions 2: uniform in unit square, uniform on some integer rectangular grid (many duplicates), uniform in the unit disk, and uniform in an annulus of tiny width (both inner and outer radii are within  $10^{-8}$  of one). For dimension 3, we choose ball centers uniformly from a cube, a cubic grid, a unit ball, and an annulus of tiny width.

In Table 2 and Table 3 we provide runtimes for each of the implementations *in two dimensions*. The value in the table is the runtime of the implementation, averaged over 100 random sets (the same sets were used for different implementations). As before, only the absolute runtimes of the base implementation (Algorithm 3.2 in CGAL’s `Min_sphere_of_spheres_d` package) is provided; for the other implementations we specify the relative speedup multiplier (the bigger the multiplier, the faster is the implementation). “MSW+ff” stands for Algorithm 3.2 with the farthest-first heuristic; this is the default algorithm in CGAL.

In Table 4 and Table 5 we provide runtimes for *three*

distribution	in unit square	in unit disk	in unit annulus	on grid
MSW	9.55ms	17.0s	18.6s	7.35s
MSW+dom	1.4x	1.3x	0.8x	1.2x
Rublev	1.7x	2.0x	1.6x	1.7x
Rublev+dom	2.4x	2.3x	1.5x	2.2x
MSW+ff	4.0x	3.5x	3.5x	4.5x

Table 2: Smallest enclosing sphere of spheres implementations in two dimensions (radii are exponentially distributed)

distribution	in unit cube	in unit ball	in unit annulus	on grid
MSW	29.9ms	16.4ms	21.9ms	13.9ms
MSW+dom	1.1x	1.0x	0.8x	0.8x
Rublev	2.1x	1.5x	1.5x	1.9x
Rublev+dom	2.0x	1.3x	1.2x	1.6x
MSW+ff	4.5x	4.0x	3.5x	5.0x

Table 4: Smallest enclosing sphere of spheres implementations in three dimensions (radii are exponentially distributed)

distribution	in unit square	in unit disk	in unit annulus	on grid
MSW	3.57s	6.43s	6.11s	0.82s
MSW+dom	1.2x	1.2x	0.8x	0.7x
Rublev	1.8x	1.8x	1.7x	0.6x
Rublev+dom	1.6x	1.6x	1.4x	0.5x
MSW+ff	3.0x	3.0x	0.7x	1.0x

Table 3: Smallest enclosing sphere of spheres implementations in two dimensions (radii are equal and very small)

distribution	in unit cube	in unit ball	in unit annulus	on grid
MSW	6.5s	13.9s	15.0s	3.0s
MSW+dom	0.7x	0.8x	0.5x	0.7x
Rublev	2.0x	2.0x	2.3x	1.4x
Rublev+dom	1.2x	1.1x	1.2x	0.8x
MSW+ff	4.0x	5.0x	1.0x	2.5x

Table 5: Smallest enclosing sphere of spheres implementations in three dimensions (radii are equal and very small)

*dimensions.* The tables are organized in the same way as their counterparts in two dimensions.

The results reveals that the domination heuristic applied to the smallest enclosing sphere of spheres problem is not as effective as for the smallest enclosing ellipse problem. Nevertheless, in two dimensions, the implementations with domination heuristic applied gain about 20 – 30% speedup on several ball center distributions (in unit square and in unit disk), compared to Algorithm 3.2.

On the other hand, in three dimensions, the implementations with domination heuristic were unable on average to be faster than their base implementations (without heuristic). Here is one explanation: for the smallest enclosing ellipse problem, the typical number of times each point is tested for violation is greater than 3, even under the move-to-front heuristic. In contrast, for the smallest enclosing sphere of spheres problem in two and three dimensions, this number is usually less than 2. This means that the removal of a point through the domination heuristic will save less than one future violation test—not enough to justify the cost of the domination test in dimension 3.

It is also striking to see how effective the farthest-first heuristic is in our tests. It outperforms any other algorithm by a large margin in almost all cases (only Rublev and Rublev+dom are sometimes faster).

The reason is that under the farthest-first heuristic, the typical number of times that a ball is involved in a violation test is close to 1 in many practical cases. In such a situation, the domination heuristic is not only useless but harmful since it unnecessarily slows down the algorithm even in cases where many dominations are detected.

For an abstract LP-type problem, the “typical number of times that a constraint is involved in a violation test” during Algorithm 3.2 is fixed but exponential in the combinatorial dimension, according to known bounds (see Lemma 3.1). Since the smallest enclosing ellipse problem in dimension 2 has already higher combinatorial dimension than the smallest enclosing sphere problem in dimension 3, the potential benefit of removing dominated constraints is clearly higher in the former problem.

At the same time, the cost of the convex hull heuristic only depends on the dimension of the ambient

space. In that sense, the problem of computing smallest enclosing ellipses is much more suitable for the convex hull heuristic.

We tend to believe (but experiments still need to be conducted) that the domination heuristic is most effective for problems with high combinatorial dimension for which a low-complexity domination test is available.

## 5 Conclusion

In this paper, we have developed a heuristic that can speed up the existing linear-time algorithm for LP-type problems in practice, while it at the same time maintains the theoretical runtime guarantee of the algorithm. The heuristic is combinatorial, meaning that it can be formulated purely within the known abstract framework of LP-type problems.

We have presented two applications where the heuristic successfully improves the practical performance of the standard algorithms for many inputs. For one of the two problems (computing smallest enclosing spheres of spheres) a standard geometric heuristic is still faster, though.

It is an open problem to establish a theoretical connection between the combinatorial dimension of the LP-type problem and the effectiveness of the domination heuristic for random input, say.

## Acknowledgment

The authors thank three anonymous ALENEX referees for useful comments that helped to improve the presentation.

## References

- [1] Kaspar Fischer and Bernd Gärtner. The smallest enclosing ball of balls: Combinatorial structure and algorithms. *International Journal of Computational Geometry and Applications (IJCGA)*, 14(4–5):341–387, 2004.
- [2] F. Friedman. Minimal enclosing circle and two and three point partitions of a plane. In *Proc. International Conference on Scientific Computing*, 2006.
- [3] B. Gärtner. *Randomized Optimization by Simplex-Type Methods*. PhD thesis, Freie Universität Berlin, 1995.
- [4] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM J. Comput.*, 24:1018–1035, 1995.
- [5] B. Gärtner. Fast and robust smallest enclosing balls. In *Proc. 7th annu. European Symposium on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 325–338. Springer-Verlag, 1999.
- [6] B. Gärtner and S. Schönherr. Exact primitives for smallest enclosing ellipses. *Information Processing Letters*, 68:33–38, 1998.

- [7] Jacob E. Goodman and Joseph O’Rourke. *Handbook of Discrete and Computational Geometry*, chapter Linear Programming, pages 999–1014. CRC Press, 2004.
- [8] S.I. Lyashko and B.V. Rublev. Minimal ellipsoids and maximal simplexes in 3d euclidean space. *Cybernetics and Systems Analysis*, 39(6):831–834, 2003.
- [9] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.
- [10] Jiri Matousek and Petr Skovron. Three views of lp-type optimization problems.
- [11] V. Osipov, P. Sanders, and M. Zumstrull. Improved algorithms for the smallest enclosing sphere problem. Work in progress, 2008.
- [12] B.V. Rublev and Y.I. Petunin. Minimum-area ellipse containing a finite set of points. i. *Ukrainian Mathematical Journal*, 50(7):1115–1124, 1998.
- [13] B.V. Rublev and Y.I. Petunin. Minimum-area ellipse containing a finite set of points. ii. *Ukrainian Mathematical Journal*, 50(8):1253–1261, 1998.
- [14] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991.
- [15] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer-Verlag, 1991.