An Efficient, Exact, and Generic Quadratic Programming Solver for Geometric Optimization*

Bernd Gärtner

Institut für Theoretische Informatik, ETH Zürich
ETH Zentrum, CH-8092 Zürich, Switzerland

gaertner@inf.ethz.ch

Sven Schönherr

Institut für Informatik, Freie Universität Berlin
Takustr. 9, D-14195 Berlin, Germany
and
Institut für Theoretische Informatik, ETH Zürich
ETH Zentrum, CH-8092 Zürich, Switzerland

sven@inf.ethz.ch

ABSTRACT

We present a solver for quadratic programming problems, which is tuned for applications in computational geometry. The solver implements a generalization of the simplex method to quadratic programs. Unlike existing solvers, it is efficient if the problem is dense and has few variables or few constraints. The range of applications covers well-known problems like *smallest enclosing ball*, or *polytope distance*, but also linear programming problems like *smallest enclosing annulus*. We provide an exact implementation with only little overhead compared to pure floating-point code. Moreover, unlike all methods for these problems that were suggested (and implemented) before in computational geometry, the runtime in practice is not exponential in the dimension of the problem, which for example allows to compute smallest enclosing balls in dimensions up to 300 (beyond that, the exact arithmetic becomes the limiting factor).

The solver follows the generic programming paradigm, and it will become part of the European computational geometry algorithms library CGAL.

1. INTRODUCTION

Many geometric optimization problems can be formulated as instances of *linear programming* (LP) or *quadratic programming* (QP), where either the number of variables or the number of constraints is small. LP is concerned with the minimization of a linear function subject to linear (in)equality constraints, while QP deals with convex quadratic objective functions of the form

$$x^T Dx + c^T x,$$

with D being a positive-semidefinite matrix. Examples are the following:

Smallest enclosing ball.

Given n points in d-dimensional space, find the ball of smallest volume containing all the points. This is a classical problem of computational geometry, with many applications, for example in bounding volume heuristics.[1] For fixed dimension d, $O(n)$ algorithms are known [15], and efficient implementations exist [5, 14]. We show that the problem can be formulated as a QP problem, although this is not obvious from its definition. Thus, our QP solver leads to an extremely fast solution in low dimensions, and is much more efficient than previous codes in moderately high dimensions.

Polytope distance.

Given two polytopes P, Q, defined by a total of n vertices (or n halfspaces) in d-dimensional space, find two points $p \in P, q \in Q$ with smallest Euclidean distance. For fixed d, an $O(n)$ solution follows from the fact that this problem can be formulated as an *LP-type problem* [9]. It has explicitly been addressed by Wolfe for the special case where one polytope is defined by a single point [17] and by Sekitani and Yamamoto in the general case [13]. The latter algorithm applies the technique Welzl has used for the smallest enclosing ball problem [15], while the former works in a simplex-type fashion; in fact, when specialized to the situation in Wolfe's paper, our algorithm is an implementation of his method. This problem—mostly occurring in collision detection—immediately fits into the QP framework.

Smallest enclosing annulus.

Given n points in d-dimensional space, find the annulus (region between concentric spheres with radii R and r, $R \geq r$) that contains the points and minimizes the difference $R^2 - r^2$. For $d = 2$, this is the annulus of smallest area, and in any

*supported by Project ESPRIT IV LTR No. 28155 GALIA (Swiss participation supported by the Swiss Federal Office for Education and Science).

[1] Over the years, the first author has obtained requests for source code, from an amazingly wide range of application areas, see [5].

dimension, the optimal annulus can be used to test whether the input points lie approximately on a sphere. Roundness tests have in general received some attention recently [8]. The problem is also of theoretical importance, because it can be used in an approximation algorithm for the minimum-width annulus, a much harder problem [1]. The smallest enclosing annulus problem is an LP problem, but it can of course also be considered as QP with a 'degenerate' objective function, having $D = 0$. We provide test results also for this problem, mainly to show that we incur no loss of efficiency in comparison with a dedicated LP solver in the same scenario [4]. On the contrary, care has been taken to optimize the code, which is reflected in the run-times.

Optimal separating hyperplane.

Given two point sets P and Q with a total of n points in d-dimensional space, test whether they can be separated by a hyperplane, and if so, find a separating hyperplane that maximizes the distance to the nearest point. The separation itself can be done by LP, but finding the optimal hyperplane is QP. The problem can be reduced to the polytope distance problem.

Our contributions

It is common to all the four problems mentioned above that d is usually small compared to n, often even constant ($d = 2, 3$), and in order to solve them efficiently, this has to be taken into account. Like in case of LP, existing solvers for QP are not tuned for this scenario. They usually work in the 'operations research' setting, where both the number of constraints and the number of variables is large, but the matrix representing the constraints is sparse. Of course, such solvers can be used to address problems in our scenario, but a performance penalty is unavoidable in this case. Below, we underpin this statement by comparing our results to the ones obtained using the QP-solver of CPLEX.

The distinguishing features of our new solver can be summarized as follows.

A simplex-type method.

It is well-known that the simplex method for linear programming can be generalized to deal with quadratic programming problems. For this, however, one usually blows up the size of the constraint matrix in such a way that both the number of constraints and the number of variables become large [16]; in our scenario, this would mean to give away the crucial advantage of having one parameter d small. As it turns out, if care is taken, the simplex method can smoothly be generalized to QP, where the blow-up is limited by the rank of D, rather than its size. This rank is at most d in our applications, even if there are many variables.

A simplex-type method has another advantage: it generates *basic* solutions. In the smallest enclosing ball problem, for example, this means that not only the optimal radius, but also the input points that determine the optimal ball are computed. In contrast, QP problems are often solved using interior point methods[2], and to obtain basic solutions from that requires additional effort.

[2]for example, CPLEX's quadratic programming solver does this

Efficient exact computations.

As already shown in case of LP [4], simplex-type methods allow an easy scheme to combine exact and floating-point arithmetic, without sacrificing too much performance. Namely, the number of exact arithmetic operations required is usually only a function of the smaller parameter of the problem. We generalize this scheme to the QP case, where it also works very well. To give a concrete figure: computing the smallest enclosing ball of 1,000,000 points in dimension $d = 3$ takes 7.9 seconds with our method (and the result is guaranteed to be correct!), while the highly-tuned floating-point code in [5] is only by a factor of less than four faster.

No 'curse of dimensionality'.

The simplex-type approach is in practice polynomial in both parameters (number of constraints and number of variables) of the QP problem. This means, we can solve the concrete geometric optimization problems mentioned above even in moderately high dimensions ($50 \leq d \leq 300$, depending on the problem). In particular, for the smallest enclosing ball problem this substantially extends the range of feasible dimensions that can be handled by computational geometry codes. Interior point codes (like CPLEX's QP solver) can go much higher, so if that is required, they are the ones to choose (with the disadvantages mentioned above). Still, we obtain a solver that is very efficient in low dimensions, always exact, and can handle higher dimensions as well (with decreasing efficiency, because of the exact arithmetic).

2. QP AND THE GEOMETRIC OPTIMIZATION SCENARIO

Quadratic Programming is the problem of minimizing a convex quadratic function in n variables, subject to m linear (in)equality constraints over the variables. In addition, the variables may have to lie between prespecified bounds. In this general formulation, QP can be written as

$$\begin{array}{lll} \text{(QP)} & \text{minimize} & c^T x + x^T D x \\ & \text{subject to} & Ax \begin{array}{c}\leq\\ \geq\\ =\end{array} b, \\ & & \ell \leq x \leq u. \end{array} \qquad (1)$$

Here, A is an $m \times n$-matrix, b an m-vector, c an n-vector, D a positive semi-definite $n \times n$-matrix[3], and ℓ, u are n-vectors of *bounds* (values $-\infty, \infty$ may occur). The symbol '$\begin{array}{c}\leq\\ \geq\\ =\end{array}$' indicates that any of the m order relations it stands for can independently be '$<$', '$=$' or '$>$'. If $D = 0$, we obtain a linear program as a special case of QP.

If a vector $x^* = (x_1^*, \ldots, x_n^*)^T$ exists that satisfies all constraints, the problem is called *feasible* and x^* is a *feasible solution*, otherwise the problem is called *infeasible*. If the *objective function* $f(x) = c^T x + x^T D x$ is bounded from below on the set of feasible solutions x^*, the problem is called *bounded*, otherwise *unbounded*. If the problem is both feasible and bounded, the objective function assumes a unique minimal value for some (not necessarily unique) optimal feasible solution x^*. Solving the quadratic program means finding such an optimal solution x^* (if it exists).

[3]i.e. $x^T D x \geq 0$ holds for all x.

The scenario we exclusively treat in this paper is the one where $\min(n, m)$ is small. In the geometric applications, this value is closely related to the dimension of the space the problem lives in. The value $\max(n, m)$, on the other hand, may be very large—it ususally comes from the number of objects (points, halfspaces etc.) that define the problem. Moreover, we assume no sparsity conditions—both matrices A and D may be dense. Namely, the geometric optimization problems we want to handle typically exhibit a dense structure; when we deal with point sets in Euclidean space, for example, the concept of sparsity can even be meaningless. Many properties of point sets are translation invariant, in which case coordinates with value zero play no special role at all.

Let us briefly discuss why 'standard' solvers (addressing the case where both m, n may be large and the problem is sparse), cannot efficiently be used in our scenario.

First of all, if n is large and D is dense, the problem cannot even be entered into solvers which require the $\Theta(n^2)$ nonzero entries of D to be explicitly given. This situation for example occurs in connection with the smallest enclosing ball problem. There is an alternative formulation in which D is sparse, but in return, the constraint matrix A will get larger. Our solver can handle matrices D which are implicitly given, and if m is small, only few entries will ever be evaluated.

Furthermore, the performance of standard solvers in practice crucially depends on the number of nonzero entries in the problem description, rather than on n and m (which is exactly what one wants for large, sparse problems). In our scenario, however, this is bad news, because in the dense case, no advantage is taken from the fact that one of the parameters is small.

3. THE QP SIMPLEX METHOD

In this section, we describe our generalization of the simplex method to QP. We will not give an in-depth treatment, but rather point out the main differences to the usual simplex method for LP (referred to as *LP simplex* in the following). Details appear in the full version. For the LP simplex, the reader may consult [2] and [4].

For the description, we consider QP in *standard form*, given as

$$\begin{array}{lll} \text{(QP)} & \text{minimize} & c^T x + x^T D x \\ & \text{subject to} & Ax = b, \\ & & x \geq 0, \end{array} \tag{2}$$

where the number of variables n is at least as large as the number of equality constraints m. As in the LP simplex, explicit bounds $\ell \leq x \leq u$ as in (1) can smoothly be integrated, and if inequality constraints occur, they can be turned into equalities by introducing slack variables. This means, if m is small in (1), we may without loss of generality assume that the QP is in standard form, having only few equality constraints.

If $m \gg n$ in (1), the problem contains many inequality constraints in all interesting cases; however, turning them into equalities generates a large number of slack variables, so that it is no longer true that the resulting standard form QP has few equality constraints. The crucial observation here is that the former inequalities can be treated implicitly; intuitively, we 'trade them in' for the new slack variables, moving the problem's complexity from the constraints to the variables, after which we are basically back to a standard from QP with $m \leq n$. Details are omitted here. In the following, we assume that we are given a problem in the form of (2).

Basic solutions.

As the LP simplex method, the QP simplex iterates through a sequence of *basic solutions*, always improving the objective function value. A basic solution is characterized by a subset B of the variables (a *QP-basis*) and numerical values x_B^* for the variables in B; variables not in B will always have value zero. Unlike in the LP-simplex, the basis B may have more than m variables, and in this case their values are not uniquely determined. Instead, x_B^* will be the optimal solution to the unconstrained subproblem

$$\begin{array}{lll} \text{(UQP)} & \text{minimize} & c_B^T x_B + x_B^T D_B x_B \\ & \text{subject to} & A_B x_B = b. \end{array} \tag{3}$$

The set B defines a basic solution if and only if this problem has a unique optimal solution $x_B^* > 0$ which is also feasible for the original problem (2). Here, c_B, D_B and A_B are the entries of c, D and A relevant for the variables in B. In case of LP (i.e. $D = 0$), this specializes to the usual notion of basic feasible solutions (in the nondegenerate case).

The following lemma gives a bound for the maximum size of a QP-basis.

LEMMA 3.1. *Every QP-basis B satisfies*

$$|B| \leq m + \mathit{rank}(D).$$

Again, if $D = 0$, we recover the bound for LP-bases. The lemma is a crucial ingredient of our method. Well-known methods to generalize the simplex algorithm to QP integrate D explicitly into the constraint matrix [16]. This is appropriate for large sparse problems and corresponding solvers, because it does not increase the number of nonzero entries in the problem description. In our scenario, however, this method is a killer, because it results in a problem with many variables *and* many constraints. The lemma limits the influence of D by considering its rank rather than its size; as we see below, this rank is small in our applications.

Ratio Test.

The process of going from one basic solution to the next is called a *pivot step*. In the LP simplex, the ratio test is the part of the pivot step that decides which variable has to *leave* the basis if another variable *enters* it. Unless the problem is degenerate, there is a unique choice for the leaving variable. In the QP simplex, it can happen that a variable enters the basis, but there is no leaving variable, so that the basis gets larger. This is the case if the objective function

reaches a local minimum (while the entering variable is increased), before some other (basic) variable goes down to zero. Moreover, it can be the case that even if some leaving variable is found, the solution at that point is not basic. In this case, the pivot step continues, and more variables may leave the basis, until another basic solution is discovered.

The other part of the pivot step, the *pricing*, decides which variable is chosen as the entering variable, if there is a choice. The QP pricing works in the same way as the LP pricing. As in [4], we combine exact and floating-point arithmetic, and we use *partial* pricing, a highly efficient strategy to reduce the cost of a single iteration if $n \gg m$. (A theoretical analysis of partial pricing has recently been done [7].)

4. THE IMPLEMENTATION

The QP solver is carefully designed and implemented in C++. We followed the generic programming paradigm, as it is realized in the STL [11], and the design goals of CGAL [3]. The most important ones, among others, were flexibility for the user and efficiency of the code. In the sequel we describe how these at first sight conflicting design goals can be achieved both at the same time.

Some calculations in the pivot step are only needed for the QP case and can be omitted if $D = 0$. We introduced a compile time tag indicating a linear respectively quadratic objective function. Thus, the compiler only generates the code needed for the specific type of problem, resulting in almost no overhead compared to a stand-alone implementation for LP (this is also underpinned by the test results).

Different geometric optimization problems come with different representations. We allow the user to choose his favorite format by only asking for *iterators* to access the problem. This also avoids copying overhead and, more important, gives the possibility of representing the $n \times n$ matrix D implicitly (which is indeed done in the tests for polytope distance and smallest enclosing ball).

The efficiency of the algorithm heavily depends on the underlying arithmetic. The user can specify the optimization problem with a fast (and possibly inexact) number type, which is used for the pricing, while providing an exact number type for the internal representation of the basis inverse and the current solution. This combination of inexact and exact arithmetic results in a fast *and* correct implementation.

The whole pricing step is encapsulated in a class. It can be replaced with other pricing strategies by deriving a new class from a given base class and specializing some virtual functions (this feature was used to perform the tests involving partial and full pricing and different arithmetic, see the next section).

5. TEST RESULTS

We have tested our algorithm on three different problems, with various settings. All these problems can be formulated as standard form QP (2) with few constraints.

The first one is the polytope distance problem, where the input polytopes are specified by points. The case where they are specified by halfspaces results in a QP (1) with $m \gg n$; the routines needed to handle this case exist as prototypes but have not reached a stable state yet; therefore we do not include test results for this scenario.

For the smallest enclosing ball problem (our major test problem), we have performed extensive tests, and compared the results to the ones obtained using other codes (including the QP-solver of CPLEX). We will argue that our code is a substantial new contribution.

Finally, the smallest enclosing annulus problem is mainly selected as a test problem, because it is an LP problem, and benchmarks have already been obtained for an exact LP solver on that problem. It turns out that although we solve LP as a special case of QP, our method is faster than the dedicated LP code described in [4]. This is due to the fact that great care has been taken to ensure runtime efficiency (see previous section).

All tables show the performance of different codes on data of different dimensions, for different numbers of points, and different arithmetics (averaged over several runs in each case). For smallest enclosing ball, the codes that were used are the first author's `double`-only code `Miniball`, the algorithms of LEDA [10] and CGAL [14] (in the latter case also the dedicated 2-dimensional version), our new QP based method, and finally the QP-solver of CPLEX. For the smallest enclosing annulus, the exact LP solver of the first author was compared with our new method.

Most tests have been performed with pseudo-random input, i.e. the coordinates were chosen as the lower-order 24 bits of the pseudo-random numbers generated using the generator `random`. This generator is not a linear congruential generator, and therefore not subject to 'nonrandom' effects as described in [6], as far as we know. For smallest enclosing ball and annulus, we further have tested with degenerate inputs, sets of points lying exactly or almost on a sphere.

The three types of arithmetics are `double` (floating-point only), `filtered` (a standard floating-point filter approach in case of LEDA, and our hybrid scheme in case of the QP method), as well as `exact`, denoting full multiple precision number arithmetic.

In case of the QP solver, two pricing strategies are available, full and partial. Partial pricing is almost always faster than full pricing, which is therefore not tabulated for all tests (see [4] for an explanation of the pricing strategies).

All test results appear in tables at the end of the paper (see Section 7).

5.1 Polytope Distance

For point sets $P = \{p_1, \ldots, p_r\}$, $Q = \{q_1, \ldots, q_s\}$, $r+s = n$, the polytope distance problem is to minimize $\|p - q\|$ such that $p = \sum_{i=1}^{r} \lambda_i p_i$, $q = \sum_{i=1}^{s} \mu_i q_i$, where the λ_i, μ_i are in $[0, 1]$ and sum up to one. As a quadratic program, this can

be written as

$$\begin{array}{lll} \text{(PD)} & \text{minimize} & x^T C^T C x^T \\ & \text{subject to} & \sum_{i=1}^{r} x_i = 1, \\ & & \sum_{i=r+1}^{s} x_i = 1, \\ & & x \geq 0, \end{array} \tag{4}$$

where $C = (p_1, \ldots, p_r, -q_1, \ldots, -q_s)$. Here, $D = C^T C$ is an $n \times n$-matrix, but its rank is only d. Hence, by Lemma 3.1, the QP simplex will trace only bases of size $d+2$ at most. This corresponds to the fact that the closest features of two d-polytopes are always determined by at most $d+2$ points. (If the polytopes have positive distance, $d+1$ points suffice.)

Table 1 shows the results of our solver (the best available setting: partial filtered pricing) for polytope distance problems with various random point sets in various dimensions. The main 'message' of the table is that the solver is able to handle *large* instances *fast* and *exact*. A more detailed picture is obtained from the results for smallest enclosing balls, where we compete against other codes.

5.2 Smallest Enclosing Ball

If $P = \{p_1, \ldots, p_n\}$, the problem is to find a point p such that $max_{i=1}^{n} \|p_i - p\|$ is minimized. The point p is then the center of the smallest enclosing ball. The following lemma shows that this problem can be written in the form of QP. Its proof is omitted here—it follows from the Karush-Kuhn-Tucker optimality conditions for convex optimization problems [12].

LEMMA 5.1. *For an n-point set $P = \{p_1, \ldots, p_n\}$, define the $d \times n$-matrix $C := (p_1, \ldots, p_n)$, consider the quadratic programming problem*

$$\begin{array}{lll} \textit{(MB')} & \textit{minimize} & x^T C^T C x - \sum_{i=1}^{n} p_i^T p_i x_i \\ & \textit{subject to} & \sum_{i=1}^{n} x_i = 1, \\ & & x \geq 0. \end{array} \tag{5}$$

and let $x_1^*, \ldots, x_n^*$ be its optimal solution. Then the point

$$p^* = \sum_{i=1}^{n} p_i x_i^*,$$

is the center of the smallest enclosing ball of P.

As before, the matrix C has rank at most d, so Lemma 3.1 shows that the optimal basis has size at most $d+1$, corresponding to the fact that $d+1$ points suffice to determine the smallest enclosing ball. Because the matrix $D = C^T C$ is a dense $n \times n$-matrix (where n will range up to 1,000,000 below), we cannot feed the problem into CPLEX in the present formulation. However, at the cost of adding a few additional constraints and variables, the following equivalent formulation is obtained.

$$\begin{array}{lll} \text{(MB'')} & \text{minimize} & y^T y - \sum_{i=1}^{n} p_i^T p_i x_i \\ & \text{subject to} & y = Cx, \\ & & \sum_{i=1}^{n} x_i = 1, \\ & & x \geq 0. \end{array} \tag{6}$$

In all tests with CPLEX below, we use formulation (6).

While all methods that are considered can handle large point sets in dimension 2, there are quite some differences in efficiency between floating-point, filtered and exact versions of the same code. Usually, our exact QP solver is only slightly slower than the `double` versions of the other codes, but dramatically faster than their exact implementations. The LEDA version is still comparable, because it applies a filtered approach itself. In cases the table entry is blank, we did not wait for the result, the time here is at least one hour (Table 2).

As degenerate inputs for $d = 2$, we have chosen two sets of points lying exactly on a circle. The small set has coordinates such that the squares still fit into a `double` value, while the large one has not. The only `double` implementations that can reasonably handle those sets are the `Miniball` routine, and CGAL's dedicated 2-dimensional code. Again, the exact QP solver is faster than LEDA's filtered approach, and much faster than all exact versions. An interesting phenomena occurs when one slightly perturbs the points, so that they are no longer cocircular. The QP solver becomes much faster because the perturbation already suffices to make the built-in error bounds work effectively: to verify optimality in the last iteration, no exact checks are necessary anymore, while they extensively happen for the non-perturbed input (Table 3).

In $d = 3$, we have chosen a point set almost on a sphere, obtained by tiling the sphere according to longitude and latitude values. The only `double` code still able to handle this problem is `Miniball`; our filtered approach, however, is only by a factor of 6 slower, while the exact versions are out of the game (Table 4).

We also have results for higher dimensions (Tables 5, 6, and 7). These show how the missing 'curse of dimensionality' in the QP solver compensates for the effect of exact arithmetic, so that our exact solver is already faster than the inexact solver `Miniball` in dimension 20, for 10,000 points. (For $n = 100{,}000$ we are not far off, but for 1,000,000 points we reach the machine's memory limit). Note that the table entry is blank for most pure `double` versions in higher dimensions, which means that the results were too inaccurate.

Probably most interesting is that problems up to dimension 100 can routinely be handled, and even for 10,000 points in $d = 200$, we only need about seven minutes (Table 8). It should be noted that these results hold for random points, where we observed that the number of points that determine the final ball is quite small (much smaller than the dimension itself). In this situation, the QP-bases the algorithm needs to handle are relatively small, which makes the exact arithmetic fast.

Finally, we have performed comparisons with the QP solver of CPLEX (an interior point code), showing that such codes are not competitive in our scenario (Table 9). It is interesting to note that the performance of the CPLEX solver mainly depends on the product of n and d (which is closely related to the number of nonzeroes in the problem description), while we pay a penalty for larger values of d. However, the results show that our exact method is still superior to CPLEX for dimensions below 30, which is the range of di-

mensions our code is made for. It is clear that CPLEX's method will become superior as d goes higher up.

5.3 Smallest Enclosing Annulus

The problem can be formulated as a linear program in $2n$ variables and $d+2$ constraints, where it nicely fits into our scenario [4].

We have tested our QP solver against the exact solver described in [4] which employs basically the same combination of exact and floating-point arithmetic (denoted as LP simplex in the tables). However, as the results show, our solver is even faster than the dedicated LP solver. This is due to the fact that we invested much effort in optimizing the code.

As before, the filtered approach is much faster than the exact approach, and comparable to pure-floating point solutions. Unlike in the case of smallest enclosing ball, `double` versions are still able to compute the correct result in higher dimensions, which indicates that true QP is more challenging for the numerics than LP (Tables 10, 13, and 14).

In case of degenerate input (points on a circle), we observe the same phenomenon as with smallest enclosing balls: as soon as we slightly perturb the points, the filtered approach gets much faster, because less exact computations are necessary (Tables 11 and 12).

6. REFERENCES

[1] P. K. Agarwal, B. Aronov, S. Har-Peled, and M. Sharir. Approximation and exact algorithms for minimum-width annuli and shells. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 380–389, 1999.

[2] V. Chvátal. *Linear Programming*. W. H. Freeman, New York, NY, 1983.

[3] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, a computational geometry algorithms library. *Software – Practice and Experience*, 2000. to appear.

[4] B. Gärtner. Exact arithmetic at low cost – a case study in linear programming. *Computational Geometry - Theory and Applications*, 13:121–139, 1999.

[5] B. Gärtner. Fast and robust smallest enclosing balls. In *Proc. 7th Annu. ACM Symp. on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 325–338, 1999.

[6] B. Gärtner. Pitfalls in computing with pseudorandom determinants. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, 2000.

[7] B. Gärtner and E. Welzl. Random sampling in geometric optimization: new insights and applications. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, 2000.

[8] R. Kumar and D. Sivakumar. Roundness estimation via random sampling. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 603–612, 1999.

[9] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[10] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. The LEDA User Manual, 1999. Version 4.0.

[11] D. Musser and A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley, 1996.

[12] A. L. Peressini, F. E. Sullivan, and J. J. Uhl. *The Mathematics of Nonlinear Programming*. Undergraduate Texts in Mathematics. Springer-Verlag, 1988.

[13] 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*, 61(2):233–249, 1993.

[14] The CGAL Consortium. The CGAL Reference Manual, 2000. Version 2.1.

[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 Comput. Sci.*, pages 359–370. Springer-Verlag, 1991.

[16] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27:382–398, 1959.

[17] P. Wolfe. Finding the nearest point in a polytope. *Math. Programming*, 11:128–149, 1976.

7. TABLES OF RESULTS

This section contains the tables with performance evaluation results that were discussed in Section 5. With one exception, runtimes have been measured on a SUN Ultra-250 workstation, using the GNU `g++` compiler, version 2.95.2, at optimization level `-O3`.

Because the most recent version 6.5.3 of CPLEX was available to us only on a SUN Ultra-60 workstation, the results in Table 9 refer to that machine.

7.1 Polytope Distance

d	random points		
	10,000	100,000	1,000,000
2	82 ms	797 ms	8.3 s
3	90 ms	942 ms	8.8 s
5	127 ms	1.3 s	12.8 s
10	221 ms	2.3 s	21.6 s
15	416 ms	3.8 s	35.5 s
20	670 ms	4.7 s	1:01 min
30	1.1 s	9.9 s	1:23 min
50	2.2 s	22.1 s	3:30 min
100	7.0 s	57.8 s	9:59 min

Table 1: Runtimes on random polytope-distance problems, QP solver with partial filtered pricing

7.2 Smallest Enclosing Ball

Algorithm ($d = 2$)		**random points**		
		10,000	100,000	1,000,000
Miniball	(double)	9 ms	102 ms	1.3 s
LEDA Min_circle	(double)	29 ms	451 ms	5.8 s
CGAL Min_sphere	(double)	26 ms	344 ms	3.5 s
CGAL Min_circle	(double)	92 ms	982 ms	9.5 s
QP Solver	(partial,double)	28 ms	323 ms	3.3 s
QP Solver	(full,double)	59 ms	673 ms	7.0 s
LEDA Min_circle	(filtered)	103 ms	1.13 s	12.6 s
QP Solver (partial,filtered)		64 ms	661 ms	6.6 s
QP Solver	(full,filtered)	89 ms	1.01 s	10.3 s
QP Solver	(partial,exact)	3.6 s	34.6 s	5:45 min
QP Solver	(full,exact)	6.9 s	1:07 min	
CGAL Min_sphere	(exact)	6.8 s	1:02 min	
CGAL Min_circle	(exact)	11.1 s	1:51 min	

Table 2: Runtimes on random miniball problems, $d = 2$

Algorithm ($d = 2$)		**points on circle**		
		6,144	13,824	6.144 (perturbed)
Miniball	(double)	<1 ms	10 ms	10 ms
CGAL Min_circle	(double)	20 ms		60 ms
LEDA Min_circle	(filtered)	2.56 s	6.4 s	6.5 s
QP Solver (partial,filtered)		830 ms	5.1 s	60 ms
QP Solver	(full,filtered)	870 ms	9.8 s	110 ms
QP Solver	(partial,exact)	840 ms	14.6 s	2.7 s
QP Solver	(full,exact)	1.7 s	48.1 s	8.8 s
CGAL Min_sphere	(exact)	860 ms	2.0 s	7.0 s
CGAL Min_circle	(exact)	1.8 s	4.0 s	6.9 s

Table 3: Runtimes on degenerate miniball problems, $d = 2$

Algorithm ($d = 3$)		**points on sphere**
		10,000 (perturbed)
Miniball	(double)	40 ms
QP Solver (partial,filtered)		250 ms
QP Solver	(full,filtered)	260 ms
QP Solver	(partial,exact)	22.4 s
QP Solver	(full,exact)	23.6 s
CGAL Min_sphere	(exact)	48.3 s

Table 4: Runtimes on degenerate miniball problem, $d = 3$

Algorithm		**random points** (10,000)					
d		2	3	5	10	15	20
Miniball	(double)	9 ms	15 ms	34 ms	118 ms	341 ms	2.1 s
CGAL Min_sphere	(double)	26 ms	37 ms				
QP Solver	(partial,double)	28 ms	35 ms	54 ms	128 ms		
QP Solver (partial,filtered)		64 ms	84 ms	153 ms	443 ms	926 ms	1.7 s
QP Solver	(partial,exact)	3.6 s	4.4 s	7.1 s	14.8 s	23.3 s	34.5 s
CGAL Min_sphere	(exact)	6.8 s	12.4 s	32.0 s	2:19 min	7:25 min	

Table 5: Runtimes on random miniball problems, $n = 10{,}000$

Algorithm / d	random points (100,000)					
	2	3	5	10	15	20
Miniball (double)	102 ms	206 ms	502 ms	1.6 s	3.1 s	7.9 s
CGAL Min_sphere (double)	344 ms	516 ms	1.1 s			
QP Solver (partial,double)	323 ms	370 ms				
QP Solver (partial,filtered)	661 ms	814 ms	1.3 s	2.9 s	5.3 s	12.0 s
QP Solver (partial,exact)	34.6 s	42.0 s	1:07 min	2:07 min	3:10 min	4:35 min
CGAL Min_sphere (exact)	1:02 min	2:05 min	5:45 min			

Table 6: Runtimes on random miniball problems, $n = 100{,}000$

Algorithm / d	random points (1,000,000)					
	2	3	5	10	15	20
Miniball (double)	1.3 s	2.2 s	5.0 s	18.6 s	30.0 s	59.5 s
CGAL Min_sphere (double)	3.5 s	5.4 s	9.8 s	31.7 s		
QP Solver (partial,double)	3.3 s	3.8 s				
QP Solver (partial,filtered)	6.6 s	7.9 s	11.5 s	28.3 s	47.1 s	3:55 min
QP Solver (partial,exact)	5:45 min					

Table 7: Runtimes on random miniball problems, $n = 1{,}000{,}000$

d	random points	
	10,000	100,000
30	6.7 s	26.3 s
50	20.2 s	1:49 min
100	50.7 s	3:08 min
200	6:51 min	
300	20:41 min	

Table 8: Runtimes on random miniball problems, QP solver with partial filtered pricing

Algorithm / d / n	random points	
	3	30
	100,000	10,000
CPLEX (double)	12.8 s	10.2 s
QP Solver (partial,filtered)	671 ms	6.8 s

Table 9: Runtimes on two random miniball problems, compared to CPLEX's QP solver

7.3 Smallest Enclosing Annulus

Algorithm	random points ($d = 2$)	
	10,000	100,000
QP Solver (partial,double)	15 ms	232 ms
QP Solver (full,double)	65 ms	935 ms
LEDA Min_annulus (double)	4.5 s	57.6 s
QP Solver (partial,filtered)	42 ms	388 ms
QP Solver (full,filtered)	84 ms	1.1 s
LP Simplex (partial,filtered)	493 ms	4.9 s
LEDA Min_annulus (filtered)	39.3 s	7:50 min
QP Solver (partial,exact)	8.1 s	1:40 min
QP Solver (full,exact)	27.9 s	4:46 min

Table 10: Runtimes on random annulus problems, $d = 2$

Algorithm	points on circle ($d = 2$)		
	6,144	13,824	6.144 (perturbed)
QP Solver (partial,filtered)	1.8 s	4.3 s	50 ms
QP Solver (full,filtered)	1.8 s	4.3 s	80 ms
LP Simplex (partial,filtered)	1.8 s	3.5 s	320 ms
LEDA Min_annulus (filtered)	5.7 s	13.1 s	
QP Solver (partial,exact)	1.8 s	4.5 s	6.9 s
QP Solver (full,exact)	8.4 s	27.3 s	29.2 s

Table 11: Runtimes on degenerate annulus problems, $d = 2$

Algorithm	points on sphere ($d = 3$)
	10,000 (perturbed)
QP Solver (partial,double)	80 ms
QP Solver (full,double)	120 ms
QP Solver (partial,filtered)	180 ms
QP Solver (full,filtered)	210 ms
LP Simplex (partial,filtered)	630 ms
QP Solver (partial,exact)	36.7 s
QP Solver (full,exact)	59.7 s

Table 12: Runtimes on degenerate annulus problem, $d = 3$

Algorithm / d	random points (10,000)						
	2	3	5	10	15	20	30
QP Solver (partial,double)	15 ms	31 ms	41 ms	131 ms	289 ms	677 ms	
QP Solver (full,double)	65 ms	156 ms	275 ms	1.1 s	2.1 s	4.1 s	
QP Solver (partial,filtered)	42 ms	90 ms	206 ms	1.6 s	8.4 s	28.7 s	3:10 min
QP Solver (full,filtered)	84 ms	187 ms	368 ms	2.0 s	7.6 s	23.7 s	
LP Simplex (partial,filtered)	493 ms	551 ms	722 ms	2.6 s	9.8 s	33.2 s	3:35 min
QP Solver (partial,exact)	8.1 s	11.9 s	20.8 s	1:23 min	4:25 min	10:38 min	
QP Solver (full,exact)	27.9 s	54.0 s	1:55 min	8:59 min	27:31 min		

Table 13: Runtimes on random annulus problems, $n = 10{,}000$

Algorithm / d	random points (100,000)						
	2	3	5	10	15	20	30
QP Solver (partial,double)	232 ms	238 ms	335 ms	685 ms	1.5 s	3.6 s	
QP Solver (full,double)	935 ms	1.5 s	3.4 s	11.1 s			
QP Solver (partial,filtered)	388 ms	431 ms	704 ms	2.6 s	10.8 s	36.8 s	3:52 min
QP Solver (full,filtered)	1.1 s	1.7 s	3.6 s	12.5 s			
LP Simplex (partial,filtered)	4.9 s	4.9 s	5.3 ms	8.0 s	17.8 s	47.7 s	4:23 min
QP Solver (partial,exact)	1:40 min	1:50 min	3:01 min	8:12 min			
QP Solver (full,exact)	4:46 min	11:41 min	22:57 min				

Table 14: Runtimes on random annulus problems, $n = 100{,}000$