Abstract

The goal of this project is to improve an existing quadratic programming solver (written in C++ at ETH) w.r.t. handling of sparse input. If successful, this improvement would significantly extend the applicability of the solver in practice. The sparsity handling can be well-encapsulated both in theory and in the code, so in order to work on it, no prior knowledge of quadratic programming is necessary, and no deeper understanding of the whole C++ code must be acquired. A solid background in C++, however, would be an advantage.

Starting from Spring 2007, the Computational Geometry Algorithms Library CGAL (www.cgal.org) will contain an exact solver for convex quadratic programs, developed at ETH. Given a convex quadratic function $f$ in $n$ variables, and $m$ linear (in)equalities, the solver outputs an exact rational representation (in terms of multiprecision numbers) of an $n$-vector that minimizes $f$ subject to the given (in)equalities. This in particular works for linear programs.

The solver is based on a generalization of the simplex method to quadratic programming [1] (see also Sven Schönherr’s PhD thesis Quadratic programming in geometric optimization : theory, implementation, and applications, Department of Computer Science, ETH Zürich, 2002, available at http://e-collection.ethbib.ethz.ch/show?type=diss&nr=14738)

The applications originally intended are geometric optimization problems like computing the distance between two convex polytopes. In these applications, one of the two parameters $n$ and $m$ is usually small. But as it turns out, many quadratic programs that occur in other fields do not have $\min(n,m)$ small, and for such problems, the current solver is not efficient. An important reason is that linear equation systems over $\min(n,m)$ variables have to be solved (exactly), and currently this is done by explicitly maintaining the inverse of the system’s matrix $A$. Often, $A$ is sparse (contains many zeroes), but since $A^{-1}$ will typically not be sparse, the code is slow even for sparse inputs.

One approach to dealing with sparse inputs is to replace the exact inverse $A^{-1}$ by a suitable factorization of $A$; there are libraries available for such factorizations, even if we insist on exact computations with multiprecision numbers. During the project, you will therefore identify suitable libraries and try them out. This requires to identify a clean interface between the solver and the linear algebra. A second step (if time permits) would be to fine-tune the implementation. For example, the next matrix $A'$ in the algorithm differs from $A$ by only one or two rows and columns, so that it might pay off to update the factorization instead of recomputing it. While standard techniques are available for that in linear programming, the quadratic case is much less understood. This second part could therefore also be quite interesting, theorywise.

References