A Sparse QP-Solver Implementation in CGAL

Yves Brise, 20090904
Problem

\[
\begin{align*}
\text{min} & \quad c^T x + x^T D x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

\[
\begin{align*}
c, x & \in \mathbb{R}^n \\
b & \in \mathbb{R}^m \\
D & \in \mathbb{R}^{n \times n} \\
A & \in \mathbb{R}^{m \times n}
\end{align*}
\]
Problem

\[ \begin{align*}
\min & \quad c^T x + x^T Dx \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*} \]

\[ c, x \in \mathbb{R}^n \]
\[ b \in \mathbb{R}^m \]
\[ D \in \mathbb{R}^{n \times n} \]
\[ A \in \mathbb{R}^{m \times n} \]

Feasible region like LP, but quadratic objective function
Applications in CG

Smallest Enclosing Ball

Polytope Distance

Smallest Enclosing Annulus
Status Quo

Current implementation optimized for dense systems and $\min\{m, n\}$ small ($< 300$).
Status Quo

Current implementation optimized for dense systems and $\min\{m, n\}$ small ($< 300$).

Simplex-like exact algorithm.
Interior point codes can handle much larger $m$ and $n$, but they may misclassify certain instances.
Status Quo

Current implementation optimized for dense systems and $\min\{m, n\}$ small ($< 300$).

Simplex-like exact algorithm.
Interior point codes can handle much larger $m$ and $n$, but they may misclassify certain instances.

Other Applications

Portfolio analysis, structural analysis, VLSI design, image restoration, signal processing, data fitting, economic power dispatch, flow analysis, agronomy, combinatorial optimization, fuzzy control systems, ... 

> 400 application papers
Other Applications

Portfolio analysis, structural analysis, VLSI design, image restoration, signal processing, data fitting, economic power dispatch, flow analysis, agronomy, combinatorial optimization, fuzzy control systems, . . .

> 400 application papers

Applications often result in sparse systems, i.e., $A$ and $D$ have a lot of zero entries.

But: $m$ and $n$ can both be large.
Goal I (Code)

Restructure current implementation to allow sparse input.

Get rid of some outdated routines.

Provide the first exact QP-solver for arbitrary inputs.
Interfaces

typedef CGAL::Quadratic_program_from_iterators
   <int**, int*, ..., bool*, int*, int**, int*> Program;
Program qp (...);

typedef CGAL::Quadratic_program<int> Program;
Program qp (...);
qp.set_a(0, 0, 1);
...

typedef CGAL::Quadratic_program_from_mps<int> Program;
std::ifstream in ("qp.mps");
Program qp(in);

CGAL::solve_quadratic_program(qp, int);

From iterators
Programmatically
From file
template <typename ET>
class QP_vector {
    private:
        typedef std::map<int,ET> Vector;
    public:
        ...
}

template <typename ET>
class QP_matrix {
    private:
        typedef std::map<int,QP_vector<ET>> Matrix;
    public:
        ...
}
Goal II (Theory)

Develop update scheme for sparse basis matrix in the case of QP.

Develop efficient pivot rule.
Simplex-like Algorithm

1. Find basic feasible solution.

2. Check optimality of solution.

3. Pivot step
   a) Pricing (variable entering the basis)
   b) Ratio test (variables leaving the basis)
   c) Update (recompute basis matrix)

4. Go back to 2.
Step 3c Detail

$B$ is the active set of variables

$$M_B := \begin{pmatrix} 0 & A_B \\ A_B^T & 2D_B \end{pmatrix} \quad \text{Basis matrix}$$
Step 3c Detail

\( B \) is the active set of variables

\[ M_B := \begin{pmatrix} 0 & A_B \\ A_B^T & 2D_B \end{pmatrix} \]  

Basis matrix

\( B' \) is the new active set after 3a & 3b

\( M_B \) is kept as inverse \( (M_B^{-1}) \)

Problems:
1. Expensive computation
2. Loss of sparsity
Step 3c Detail

\(M_B\) is kept as inverse \((M_B^{-1})\)

Problems:
1. Expensive computation
2. Loss of sparsity

If we keep \(M_B\) as \(LU\)-factorization, sparsity is retained.
Step 3c Detail

$M_B$ is kept as inverse ($M_B^{-1}$)  

Problems:
1. Expensive computation
2. Loss of sparsity

If we keep $M_B$ as $LU$-factorization, sparsity is retained.

Case study done by Robleda (Master thesis, 2008) shows that using the $LU$-factorization on sparse instances can significantly speed up the computation.
Step 3c Detail

$M_B$ is kept as inverse ($M_B^{-1}$) Problems:
1. Expensive computation
2. Loss of sparsity

If we keep $M_B$ as $LU$-factorization, sparsity is retained.

Case study done by Robleda (Master thesis, 2008) shows that using the $LU$-factorization on sparse instances can significantly speed up the computation. 

even without efficient update!
Goal III (Benchmarking)

How does the sparse version do on dense inputs?

How big is the penalty of treating special cases separately (LP, non-negative)?

Compare with other solvers.
template <typename Program, typename ET, 
    typename Is_linear, typename Is_nonnegative >
Quadratic_program_solution<ET> solve_program
   (const Program &p, const ET&, Is_linear, Is_nonnegative, 
    const Quadratic_program_options& options)

Distinctions between QP/LP and simple/general bounds.
Progress

Project started: June 2009

Progress to date:
Preliminary version of sparse matrix representation.
Internal integration into existing code.

Project end: 2011

Thank you! Email: ybrise@inf.ethz.ch