A Brief History of Computational Linear Programming

Geraldo Veiga

\( \mathcal{R}^n \) Ciência e Tecnologia

1\textsuperscript{st} Brazilian Workshop on Interior Point Methods
April 27-28, 2015 - Campinas, Brazil
Outline

1. Pioneers
2. The Simplex Method
   - Early development
   - Solving Larger Scale Problems
   - Doubting the Simplex Method
   - The Simplex method comes back
3. The Ellipsoid Method
4. Interior Point Methods
   - A Personal Reminiscence
   - Dual-Affine Scaling Algorithm
   - Primal-Dual Algorithm with infeasibilities
5. Parallelization of an Interior Point Method
   - Direct Factorization Methods
   - Parallelization strategies
   - Experimenting with MUMPS
   - Case study
   - Conclusion and future work
6. Some ideas for future development
7. References
Pioneers I

- Fourier [1826] studies the properties of system of linear inequalities, more complex than system of equations.

- De la Vallée-Poussin [1911] develops an iterative procedure for linear minimax estimation which can be adjusted to solve linear optimization problems [Farebrother, 2006].

- As early as 1930, A.N. Tolstoı described a number of solution approaches for transportation problems [Schrijver, 2012].

- Kantorovich [1939] proposes rudimentary algorithm for linear programming applied to production planning.
Pioneers II

- These contributions only come to attention after independent development of linear programming theory and the Simplex Method.

- Other contributions to optimization by mathematicians in the USSR also went unrecognized elsewhere victim of government personal and ideological obstacles imposed to international scientific interchange [Polyak, 2014]
Pioneers III

- A quote from Kantorovich betrays an attempt of making dual prices more palatable to Marxist orthodoxy [Todd, 2002]

  “I want to emphasize again that the greater part of the problems of which I shall speak, relating to the organization and planning of production, are connected specifically with the Soviet system of economy and in the majority of cases do not arise in the economy of a capitalist society.”

  [Kantorovich, 1939]
The Simplex Method I

- George Dantzig proposes the Simplex Method in 1947 [Dantzig, 2002]
- Early works by Leontief, von Neumann and Koopsman directly influenced the theoretical development of linear programming [Dantzig, 2002]
- From Dantzig’s point of view: Not just a qualitative tool in the analysis of economic phenomena, but a method to compute actual answers [Bixby, 2012]
- Unfortunately, not all economists are keen of numbers, the 1975 Nobel Prize in Economics was awarded to Kantorovich and Koopsman, ignoring Dantzig’s contribution [Nobelprize.org, 2015]
The Simplex Method II

- First application to the solution of a non-trivial LP: $21 \times 17$ instance of Stigler Diet Problem (computation time was 120 man-days!) [Dantzig, 1963]
- Orchard-Hays (1954) produces first successful LP software
- Sparse matrix representation and product-form of the inverse
- Largest problem solved: $26 \times 71$ solved in 8 hours [Bixby, 2002]
The Simplex Method I

- Designed “to be computable”, developed side-by-side with digital computers [Dantzig, 2002]
The Simplex Method II

- Familiar examples extracted from the original articles:

2. BLOCK TRIANGULARITY (GENERAL CASE)

By “block” triangular we mean that if one partitions the matrix of coefficients of the technology matrix into submatrices, the submatrices (or blocks) considered as elements form a triangular system,

\[
\begin{bmatrix}
A_{11} \\
A_{21} & A_{22} \\
\cdots & \cdots & \cdots \\
A_{r1} & A_{r2} & \cdots & A_{rr}
\end{bmatrix}
\]

\(^2\) A term suggested by Walter Jacobs.
The Simplex Method III

\[
\begin{array}{cccc}
  x_1 & x_2 & \cdots & x_n \\
  c_1 & c_2 & \cdots & c_n \\
  A_1 & A_2 & \cdots & A_n \\
  B_1 & & & b \\
  B_2 & & & b_1 \\
      & \vdots & \cdots & \vdots \\
\end{array}
\]
The Simplex Method IV

For example, von Neumann, [9], in considering a constantly expanding economy, developed a linear dynamic model whose matrix of coefficients may be written in the form (5)

$$
\begin{bmatrix}
A & -B & A \\
-B & A & \\
-B & & \\
& -B & \\
& & -B & A
\end{bmatrix},
$$

where $A$ is the submatrix of coefficients of activities initiated in period $t$, and $B$ is the submatrix of output coefficients of these activities in the following period.

- Extensions to quadratic programming and linear complementarity.
- *Large scale* not in today’s sense. Then, $1000 \times 2000$ would be the limit of tractability with methods that used special matrix structures. Still, the methods survive, updated to advances in computer technology, especially parallel architectures.
The Simplex Method V

- The common belief is that the Simplex Method is Exponential behavior in theory and almost linear in practice

- Sparse LU representation of the basis with Bartel-Golub/Forrest-Tomlin/Fletcher-Matthews update. [Forrest and Tomlin, 1972]
Doubting the Simplex Method I

- Even Dantzig had his doubts [Todd, 2002]:

  "Luckily the particular geometry used in my thesis was the one associated with the columns of the matrix instead of its rows. This column geometry gave me the insight which led me to believe that the simplex method would be an efficient solution technique. I earlier had rejected the method when I viewed it in the row geometry because running around the outside edges seemed so unpromising." [Dantzig, 1991]

- Finite behavior was enough in early analysis of the algorithm

- However ... It was not even finite!
Doubting the Simplex Method II

- Cycling is possible in degenerate LPs.
- But it got fixed with Bland’s pivoting Rule [Bland, 1977]
- Theory of computational complexity gets developed
- Is linear programming in $P$?
- Klee and Minty [1972] shows the simplex method with a common pivot rule is of exponential complexity
The Simplex method comes back I

- Theory counters with bounds on the expected number of pivot steps [Borgwardt, 1982]
- The work of Karmarkar had stimulated a rebirth of interest in LP, both on the theoretical and computation sides [Bixby, 2012]
- Computational studies on problems with numbers of variables ranging up to the millions also reaffirm confidence
- More recent linear algebra improvements such as Markowitz threshold and sparse partial pivoting [Bixby, 2002]
- Modern implementation (XMP, OSL, CPLEX, Gurobi, Mosek, Xpress) with power preprocessors,
The Simplex method comes back II

- The Simplex method is also naturally suited for mixed integer problem in Branch-and-bound and Branch-and-cut algorithms [Bixby, 2002]
- Remains a primary computational tool in linear and mixed-integer programming (MIP)
- Parallel implementations of the simplex method usually must exploit special structures
- A general approach hindered by the changing sparse pattern of the basic matrix
The Ellipsoid Method I

- The Ellipsoid Method [Khachiyan, 1979]
- Revolutionary for complexity theory without computational impact
- However brings back the idea of solving linear programs with traditionally non-linear techniques
Karmarkar’s algorithm [Karmarkar, 1984]
- Projective algorithm with a potential function sets a lower complexity for linear programming: $O(n^{3.5}L)$
- Claims of great performance gains for a dual-affine scaling variant [Adler et al., 1989a]
- Similar algorithm had gone unnoticed by LP researchers [Dikin, 1967]

Primal-Dual/Path Following methods
- New wave of interest in linear programming reintroduces path-following methods developed in the nonlinear context: Logarithm Barrier Function [Fiacco and McCormick, 1990] and Method of Centers [Huard, 1967]
- Central trajectory methods with lower complexity $O(n^3L)$
- Primal/Dual infeasible methods become standard for implementation, included in leading LP software.
- [Shanno, 2012]
A Personal Reminiscence I

- Initial contact with Karmarkar article
- Narendra Karmarkar was a recent PhD graduate from UC Berkeley, Computer Science department, working under Richard Karp
- Talk by Narendra Karmarkar at Evans Hall where he claimed a modified algorithm was "hundreds of times faster" than the Simplex Method
- Even featured on the first page of Sunday NY Times in November 18, 1984! [Gleick, 1984]
- "Breakthrough in Problems Solving" was the headline (N.B. AT&T was known for its ability to place Bell Labs stories in the science section of the NY Times)
- But making it first page news was unheard
A Personal Reminiscence II

- In Berkeley’s IEOR computer lab, I built a simple implementation in APL of the original projective algorithm confirmed the small number of iterations
- Linear algebra infrastructure under APL did not allow for a serious performance analysis
- Ilan Adler arranged with Narendra Karmakar to collaborate in a serious implementation to vouch for the speed claims
- Mauricio Resende and myself ended up leading the effort along with other students in Ilan Adler’s graduate seminar [Adler et al., 1989b]
Contrary to other recollections [Gill et al., 2008], we were never told or felt that any of the ideas in the algorithms were proprietary by AT&T. There was however a self imposed restriction to avoid any possible copyright infringement. Then, we never saw nor asked for code that Karmarkar might have written himself.

In retrospect, I feel that his comments in the early talks on the speed of the algorithms were based on simple prototypes that were not ready for sharing.

The ill feelings in initial presentations by Karmarkar [Shanno, 2012] were caused more from problem of personal style than company policy.
However, a very legitimate complain comes later with AT&T trying to enforce patents on the Karmarkar’s algorithm and the affine scaling interior point method [Karmarkar, 1988, Vanderbei, 1988, 1989]

Eventually AT&T’s Korbx system, an attempt to commercialize interior point methods, failed, making this discussion mute.
A Personal Reminiscence

- Library of Alexandria - Interior Point Wing
A Personal Reminiscence

- Christiano Lyra knocks at my door selects a pile of papers and runs to Krishna Copy Center
A Personal Reminiscence

- Back at Unicamp brilliant student Aurélio Oliveira reads the whole lot, writes dissertation, articles etc ... Ans begets this workshop.
Dual Affine Algorithm

- **c**, \(x\) \(n\)-vectors; \(A\) \(m\times n\) matrix; \(b\), \(y\) \(m\)-vectors
  \[
  \max \left\{ b^\top y \mid A^\top y \leq c \right\}
  \]

- Add slack variables
  \[
  \max \left\{ b^\top y \mid A^\top y + v = c, \ v \geq 0 \right\}
  \]

- Scaling transformation
  \[
  \hat{v} = D_v^{-1} v \quad \text{where} \quad D_v = \text{diag}(v_1^k, \ldots, v_m^k)
  \]

- Projected gradient as search direction
  \[
  h_y = (AD_v^{-2}A^\top)^{-1}b \quad \text{and} \quad h_v = -A^\top h_y
  \]
Affine-Dual Algorithm II

procedure dualAffine (A, b, c, y₀, stopping criterion, γ)

k := 0;

do stopping criterion not satisfied →

vᵏ := c − Aᵀyᵏ;
Dᵥ := diag(v₁ᵏ, ..., vₘᵏ);
hy := (ADᵥ⁻²Aᵀ)⁻¹b;
hᵥ := −Aᵀhy;
if hᵥ ≥ 0 → return fi;

α := γ × min{−vᵢᵏ/(hᵥ)i | (hᵥ)i < 0, i = 1, ..., m};
yᵏ₊₁ := yᵏ + αhy;
k := k + 1;

od

end dualAffine
Primal-Dual Algorithm with infeasibilities I

Formulation:
- Upper bounds for a subset of variables
- $c, x, s, z$ are $n$-vectors
- $u_b, x_b, s_b, w_b$ $n_b$-vectors – $x_n, s_n$ $n_n$-vectors
- $A$ $m \times n$ matrix – $b, y$ $m$-vectors

Add slack variables

$$\begin{align*}
\min \ & \{c^T x | Ax = b, x_b + s_b = u_b, x \geq 0, s_b \geq 0\} \\
\max \ & \{b^T y - u_b^T w_b | A_b^T y - w_b + z_b = c_b, \\\n& A_n^T y + z_n = c_n, w_b \geq 0, z \geq 0\}
\end{align*}$$
Primal-Dual Algorithm with infeasibilities II

- $X = \text{diag}(x)$, $S = \text{diag}(s)$, $W = \text{diag}(w)$, $Z = \text{diag}(z)$
- $\mu$ Central trajectory parameter
- Karush-Kuhn-Tucker conditions:

\[
\begin{align*}
Ax &= b \\
x_b + s_b &= u_b \\
A_b^T y - w_b + z_b &= c_b \\
A_n^T y + z_n &= c_n \\
XZe &= \mu e \\
S_b We &= \mu e \\
x, s_b, w_b, z &> 0
\end{align*}
\]
Primal-Dual Algorithm with infeasibilities III

- System of equations with primal and dual infeasibilities

\[
A \Delta x^k = -(Ax^k - b) = r_p^k \\
\Delta x_b^k + \Delta s_b^k = -(x_b^k + s_b^k - u_b) = r_u^k \\
A_b^T \Delta y^k - \Delta w_b^k + \Delta z_b^k = -(A_b^T y^k + z_b^k - w_b^k - c_b) = (r_d^k)_b = 0 \\
A_n^T \Delta y^k + \Delta z_n^k = -(A_n^T y^k + z_n^k - c_n) = (r_d^k)_n \\
Z^k \Delta x^k + X^k \Delta z^k = -(X^k Z^k e - \mu_k e) = r_{xz}^k \\
W_b^k \Delta s_b^k + S_b^k \Delta w_b^k = -(W_b^k S_b^k e - \mu_k e) = (r_{sw}^k)_b
\]
Primal-Dual Algorithm with infeasibilities IV

- Normal Equations

\[ A\Theta^k A^\top \Delta y^k = \bar{b} \]

where

\[
\Theta^k = \begin{bmatrix}
(Z^k_b (X^k_b)^{-1} + W^k_b (S^k_b)^{-1})^{-1} & 0 \\
0 & (Z^k_n)^{-1} X^k_n 
\end{bmatrix}
\]

\[
\bar{b} = r^k_p + A_b \Theta^k_b ((r^k_d)_b + (S^k_b)^{-1}(r^k_{sw} - W_b r^k_u) - (X^k_b)^{-1} r^k_{xz})
+ A_n \Theta^k_n ((r^k_d)_n - (X^k_n)^{-1} r^k_{xz})
\]
Primal-Dual Algorithm with infeasibilities V

- Other search direction computed without substantial computational effort

\[
\Delta x_b^k = \Theta_b A_b^T \Delta y^k - \Theta_b ((r_d^k)_b + \\
(S_b^k)^{-1}(r_{sw}^k - W_b r_u^k) - (X_b^k)^{-1}(r_{xz}^k)_b)
\]

\[
\Delta x_n^k = \Theta_n A_n^T \Delta y^k - \Theta_n ((r_d^k)_n - (X_n^k)^{-1}(r_{xz}^k)_n)
\]

\[
\Delta s_b^k = r_u^k - \Delta x_b^k
\]

\[
\Delta z^k = (X^k)^{-1}(r_{xz} - Z^k \Delta x^k)
\]

\[
\Delta w_b^k = A_b^T \Delta y^k + \Delta z_b^k
\]
Parallelization opportunities in Interior Point

Direct Factorization I

- Main computational step common to all variants is the solutions of a system of normal equations

\[ A\Theta^k A^\top \Delta y^k = \bar{b} \]

- Examining an implementation in Matlab/Octave, potentially computationally expensive steps:

- Computing system matrix

\[ B = A * \text{sparse}(\text{diag}(d)) * A' ; \]

- Custom parallel sparse linear algebra
Parallelization opportunities in Interior Point
Direct Factorization II

Example: BandM from the Netlib collection
Parallelization opportunities in Interior Point
Direct Factorization III

- Order for sparsity
  \[ \text{ordering} = \text{symamd}(B); \]

- Reordering for sparsity: Matrix $AA^T$ and Cholesky factors without ordering [Adler et al., 1989a]
Parallelization opportunities in Interior Point
Direct Factorization IV

- Matrix $AA^T$ and Cholesky factors after minimum degree ordering

Figure 6. Nonzero pattern of $AA^T$ after ordering (minimum degree ordering heuristic).

Figure 7. Nonzero pattern of $LU$ factors after ordering (minimum degree ordering heuristic).
Parallelization opportunities in Interior Point

Direct Factorization V

- Reordering rows of $A$ to avoid \textit{fill-in}
- Optimal ordering is \textit{NP-Complete} [Yannakakis, 1981]
- Linear solvers compute the ordering during the \textit{Analyse} step, based solely on the matrix sparsity pattern
- Performed only once in interior point algorithms, sparsity pattern are identical for all iterations
- Parallel/Distributed MPI based implementations available: ParMETIS
Parallelization opportunities in Interior Point
Direct Factorization VI

- Direct Cholesky factorization
  \[ R = \text{chol}(B(\text{ordering, ordering})) ; \]

- Repeated at every iteration, consumes most of the computational effort

- For larger problems: Main parallelization target

- Chart displaying the portion of the algorithm running time for Netlib problems, suggesting an increase with size

- Available Parallel/Distributed implementations: MUMPS [Amestoy et al., 2000] for distributed memory architectures and PARDISO for shared memory
Parallelization opportunities in Interior Point
Direct Factorization VII

- Triangular Solution for rhs

\[
dy(\text{ordering}) = R\backslash(R'\backslash\overline{b}(\text{ordering}));
\]

- General sparse linear algebra parallelization

- In distributed implementations, parallelization implied by \textit{Factorization} step
Parallelization opportunities in Interior Point
Direct Factorization VIII
Parallelization strategies

- **MPI: Message Passing/Distributed Memory**
  - Standard for high-performance computing
  - Processors operate with private memory spaces, sharing results of only through point-to-point or collective communication
  - Goals are high performance, scalability and portability
  - Bindings for Fortran and C/C++
  - Target architectures are both high performance computer clusters tightly linked with fast switched interconnects and grids of loosely-coupled systems

- **Shared memory multiprocessing**
  - Multiple computing threads operate in shared memory space
  - Programming standards: OpenMP and Pthreads (Posix threads)
  - Suited for multi-core processor architectures

- **Hybrid model of parallel programming use multi-core MPI nodes executing shared memory threads**
Experimenting with MUMPS

- Multifrontal Massively Parallel Solver MUMPS [Amestoy et al., 2000] for distributed memory architectures
- Multifrontal methods first build an assembly tree
- At each node, a dense submatrix (frontal matrix) is assembled using data from the original matrix and from the children of the node
- Main source of parallelism consists in simultaneously assigning same level frontal matrices onto separate processors
- MUMPS uses standard linear algebra libraries BLAS, BLACS, ScaLAPACK
- BLAS functions can use shared memory parallelism, depending on implementation
- Experiments with Netlib collection unsuccessful due to small size, but suggest better performance as problems grow
Multifrontal assembly trees for two orderings
Experiments with Netlib problems

Solution times
(reference = mumps-1proc)
Power system expansion planning model

- Linear relaxation of mixed integer planning model for the expansion of a combined hydro and thermal power system
- Formulated with Optgen© modeling tool, developed by PSR
- Problem instance generated with Brazilian system of 280 hydro and 120 thermal plants
- LP size: 840285 columns, 598066 rows and 2462627 nonzeros entries
- Interior Point linear system: 360455 rows and 27390204 nonzeros
Case Study Experiment

- Experiment solves one typical system from an interior point iteration
- SGI Altix ICE 8200 with 64 quad-Core Intel Xeon CPU and 512 Gbytes of distributed RAM, using a Infiniband interconnect
- Software infrastructure: MUMPS 4.8.3 with BLAS, BLACS, ScaLAPACK provided by Intel MKL 10.1
- MUMPS is successful in low-scale parallelization
- Times for the Analyze stage comparable
- Total computation is dominated by matrix-matrix multiplication
- Shared memory parallelism using OpenMP in the BLAS and Lapack routines has little effect in this architecture
MUMPS Speedup

Factorization/Solution Time

[seconds]

#processes

[seconds]

[speed-up]
Conclusion and future work

- This experiment with parallelization was class project at COPPE-UFRJ by Luiz Carlos da Costa, Jr. and Fernanda Thomé [Maculan et al., 2008]
- Large-scale problems using implementations with direct factorization can profit from parallelization, but less than expected
- Parallelization still an art form: No assurance of performance, too dependent on the infrastructure and algorithms
- MUMPS and other MPI-based tools are designed for high performance clusters
- Multi-core workstations are a better suited for shared memory parallelization
- Other sources of parallelism must be addressed
- Experiments with iterative methods for solving interior point linear systems
Some ideas for future development

- Identifying an optimal basis
  - Most modern optimization software packages include an Interior Point implementation. However, they still rely on a cumbersome *crossover* step to produce a Simplex-like optimal solution.
  - In network flow problems it is possible to guess and test the optimal basis. More interestingly, the guessed basis is used to build a preconditioner in iterative solutions of the main system of equations Resende and Veiga [1993].
  - Can this be generalized for LP and separating preconditioners?

- Parallel implementations
  - Continued increase in computer processing power depends on multicore and distributed architectures.
  - Successful parallelization using direct factorization only for special structure problems [Gondzio and Grothey, 2006].
  - Using pure iterative methods would be instantly parallelizable.
  - Design of preconditioners must also take parallel architectures into consideration. Usually, they must yield systems easy to solve (diagonal, triangular) and parallelize.


References II


References VI
