Sparsity Matters

Robert J. Vanderbei

Nov. 5, 2018

INFORMS Annual Meeting
Phoenix AZ

http://www.princeton.edu/∼rvdb

“The simplex method is 200 times faster...

**“The simplex method is 200 times faster...
than the simplex method.” – John Forrest**

More Generally...

Any optimization algorithm is 200 times faster...

More Generally...

Any optimization algorithm is 200 times faster...
than that same algorithm.

What Can Go Wrong?

- Sometimes the straightforward/default numerical *linear algebra* isn't the best way to solve a particular type of problem.

- Sometimes the obvious natural formulation of an optimization problem isn't very easy to solve but there is a *mathematically equivalent variant* that one can solve quickly.

- Sometimes the real-world problem to be solved isn't precisely defined and an *alternate formulation* might be vastly easier to solve.

I will discuss one example of each of these three scenarios.

Numerical Linear Algebra

Problems with Dense Columns

Consider this LP:

$$\begin{array}{lrcrcl} \text{maximize} & c^T x & + & c_0\, x_0 & & \\ \text{subject to} & Ax & + & a\, x_0 & = & b \\ & & & x, x_0 & \geq & 0 \end{array}$$

where x is a high-dimensional vector, x_0 is just a single scalar variable, A is a *very sparse matrix*, but a is a *dense vector*.

There are practical real-world problems with this structure. Also, the "Big-M" method for getting an initial starting point for interior-point methods involves such a structure.

The computationally intensive part of the original approach to finding the step directions in interior-point methods was to solve the so-called "normal equations" for the dual step direction Δy:

$$\begin{bmatrix} A & a \end{bmatrix} \begin{bmatrix} D^2 & 0 \\ 0 & d^2 \end{bmatrix} \begin{bmatrix} A^T \\ a^T \end{bmatrix} \Delta y = \ldots$$

where D is a diagonal matrix and d is a scalar. Multiplying out, we get:

$$(AD^2A^T + d^2\, aa^T)\, \Delta y = \ldots$$

The matrix AD^2A^T is *sparse* but aa^T is *dense*. Hence, the sum is dense and solving the system of equations as formulated is highly inefficient.

There are three ways to address this problem:

- Use the *Sherman-Morrison-Woodbury-AddYourNameHere* formula to express solutions to the dense system in terms of solutions to the system without the dense (rank one) term.

- Solve the *reduced KKT system* instead of the normal equations.
  (*Symmetric Indefinite Systems for Interior Point Methods*, Math. Prog., 58:1-32, 1993)

- Re-express the problem using several variables in place of the single variable x_0.
  (*Splitting Dense Columns in Sparse Linear Systems*, Linear Algebra and Applications, 152:107-117, 1991)

Splitting Dense Columns

A problem with these constraints...

$$\begin{bmatrix} A & a \end{bmatrix} \begin{bmatrix} x \\ x_0 \end{bmatrix} = b$$

can be reformulated in this equivalent but sparser fashion...

$$\left[\begin{array}{c|cccccc} & a_1 & & & & & \\ & & a_2 & & & & \\ A & & & a_3 & & & \\ & & & & \ddots & & \\ & & & & & a_{m-1} & \\ & & & & & & a_m \\ \hline & 1 & -1 & & & & \\ & & 1 & -1 & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & -1 & \\ & & & & & 1 & -1 \end{array}\right] \left[\begin{array}{c} x \\ \hline x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ \vdots \\ x_{0,m-1} \\ x_{0,m} \end{array}\right] = \left[\begin{array}{c} b \\ \\ \\ \\ \hline 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{array}\right]$$

For an optimizer based on solving the normal equations, this second form of the problem will solve *much faster* than the original form.

Of course, most/all modern interior-point method solvers solve the reduced KKT system to avoid this issue.

The *dense columns* example is a bit dated but it illustrates an important point:

The number of constraints, m, and the number of variables, n, often have little to do with how long it takes to solve a problem.

The *sparsity* of the constraint matrix plays a huge role.

Example:

I recently solved a generalized network flow problem using AMPL/LOQO.
It has $195{,}503$ variables and $123{,}571$ constraints.
The problem solved in 3.9 seconds on my laptop computer.

For comparison, I solved a dense problem with $1/100$-th as many variables and $1/100$-th as many constraints.
It took 59 seconds to solve.

The *dense columns* example is a bit dated but it illustrates an important point:

The number of constraints, m, and the number of variables, n, often have little to do with how long it takes to solve a problem.

The *sparsity* of the constraint matrix plays a huge role.

Example:

I recently solved a generalized network flow problem using AMPL/LOQO.
It has $195{,}503$ variables and $123{,}571$ constraints.
The problem solved in 3.9 seconds on my laptop computer.

For comparison, I solved a dense problem with $1/100$-th as many variables and $1/100$-th as many constraints.
It took 59 seconds to solve.

SPARSITY MATTERS

Mathematically Equivalent Problems

Markowitz Model for Portfolio Selection

$$\begin{array}{lrcl} \text{minimize} & \mu\, x^T \Sigma x - r^T x & & \\ \text{subject to} & e^T x & = & 1 \\ & x & \geq & 0 \end{array}$$

Here, r denotes the vector of expected returns, Σ denotes the matrix of covariances of the returns, e is the vector of all ones and μ is the *risk aversion* parameter.

Equivalent Reformulation

The matrix Σ is positive semidefinite and therefore can be expressed (and probably was defined) as a product $\Sigma = U^T U$.

$$\begin{array}{lrcrcl} \text{minimize} & -\mu\, y^T y & - & r^T x & & \\ \text{subject to} & y & - & Ux & = & 0 \\ & & & e^T x & = & 1 \\ & & & x & \geq & 0 \end{array}$$

Example. A problem with $10{,}000$ market assets and a covariance matrix based on data from 100 time periods in the past, the first problem has $10{,}000$ variables and just one constraint. It solves in 1480 seconds. The second formulation has $10{,}100$ variables and 101 constraints. It solves in 10.3 seconds—a speedup by a factor of 144.

Alternate Formulations

Basis Pursuit Variant of Compressed Sensing

Aims to recover a sparse signal, $x^* = (x_1^*, \ldots, x_n^*)^T$, from a small number of measurements: $y = Ax^*$.

We assume n is large, x^* is sparse, and the number of measurements m is small.

The *basis pursuit problem* is to "recover" x^* by solving by solving

$$(\mathrm{P}_1) \qquad \min_x \|x\|_1 \text{ subject to } Ax = y,$$

where

$$\|x\|_1 = \sum_j |x_j|.$$

Kronecker Compressed Sensing

Unlike the vector compressed sensing problem, Kronecker compressed sensing is used for sensing multidimensional signals (e.g., matrices or tensors).

For example, given a sparse matrix signal $X^* \in \mathbb{R}^{n_1 \times n_2}$, we can use two sensing matrices $A \in \mathbb{R}^{m_1 \times n_1}$ and $B \in \mathbb{R}^{m_2 \times n_2}$ and try to recover X^* from knowledge of $Y = AX^*B^T$ by solving the *Kronecker compressed sensing* problem:

$$(\mathrm{P}_2) \qquad \min \|X\|_1 \text{ subject to } AXB^T = Y.$$

When the signal is multidimensional, Kronecker compressed sensing is more natural than classical vector compressed sensing.

Kroneckerizing Vector Problems

Sometimes, even when facing vector signals, it is smart to use Kronecker compressed sensing due to its added computational efficiency.

More specifically, even though the target signal is a vector $x^* \in \mathbb{R}^n$, if we assume that n can be factored into $n_1 \times n_2$, then we can first reshape x^0 into a matrix $X^* \in \mathbb{R}^{n_1 \times n_2}$.

We then multiply the matrix signal X^* on both the left and the right by sensing matrices A and B to get a compressed matrix signal Y.

We can solve this Kronecker compressed sensing problem much more efficiently than the corresponding vector compressed sensing problem.

Vectorizing the Kroneckerization

Standard LP solvers expect to see a vector of variables, not a matrix.

The naive vectorization goes like this...

Let $x = \mathrm{vec}(X)$ and $y = \mathrm{vec}(Y)$, where, as usual, the $\mathrm{vec}(\cdot)$ operator takes a matrix and concatenates its elements column-by-column to build one large column-vector containing all the elements of the matrix.

In terms of x and y, problem (P_2) can be rewritten as an equivalent *vector compressed sensing* problem:

$$\min \|x\|_1 \quad \text{subject to} \quad Ux = y,$$

where U is given by the $(m_1 m_2) \times (n_1 n_2)$ Kronecker product of A and B:

$$U = B \otimes A = \begin{bmatrix} Ab_{11} & \cdots & Ab_{1n_2} \\ \vdots & \ddots & \vdots \\ Ab_{m_2 1} & \cdots & Ab_{m_2 n_2} \end{bmatrix}.$$

The matrix U is fully dense. *This is bad.*

Sparsifying the Constraint Matrix

The key to an *efficient* algorithm for solving the linear programming problem associated with the Kronecker sensing problem lies in noting that the dense matrix U can be factored into a product of two very sparse matrices:

$$U = \begin{bmatrix} Ab_{11} & \cdots & Ab_{1n_2} \\ \vdots & \ddots & \vdots \\ Ab_{m_2 1} & \cdots & Ab_{m_2 n_2} \end{bmatrix} = \begin{bmatrix} A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{bmatrix} \begin{bmatrix} b_{11}I & b_{12}I & \cdots & b_{1n_2}I \\ b_{21}I & b_{22}I & \cdots & b_{2n_2}I \\ \vdots & \vdots & \ddots & \vdots \\ b_{m_2 1}I & b_{m_2 1}I & \cdots & b_{m_2 n_2}I \end{bmatrix} =: VW,$$

where I denotes an $n_1 \times n_1$ identity matrix and 0 denotes an $m_1 \times n_1$ zero matrix.

The constraints on the problem are

$$Ux = y.$$

Exploiting the Sparsification

The matrix U is usually completely dense.

But, it is a product of two very sparse matrices: V and W.

Hence, introducing some new variables, call them z, we can rewrite the constraints like this:

$$\begin{array}{rcll} z & - & Wx & = 0 \\ Vz & & & = y. \end{array}$$

And, as usual, we can split x into a difference between their positive and negative parts to convert the problem to a linear program:

$$\begin{array}{lrcll} \min\limits_{x^+,x^-} & & & \mu\, \mathbf{1}^T(x^+ + x^-) & \\ \text{subject to} & z & - & W(x^+ - x^-) & = 0 \\ & Vz & & & = y \\ & & & x^+, x^- & \geq 0. \end{array}$$

This formulation has more variables and more constraints.

But, the constraint matrix is *very sparse*.

For linear programming, sparsity of the constraint matrix is the key to algorithm efficiency.

Numerical Results

For the *vector* sensor, we generated random problems using $m = 1{,}122 = 33 \times 34$ and $n = 20{,}022 = 141 \times 142$.

We varied the number of nonzeros k in signal x^0 from 2 to 150.

We solved the straightforward linear programming formulations of these instances using an interior-point solver called LOQO and a simplex-method solver called SIMPO.

We followed a similar plan for the *Kronecker* sensor.

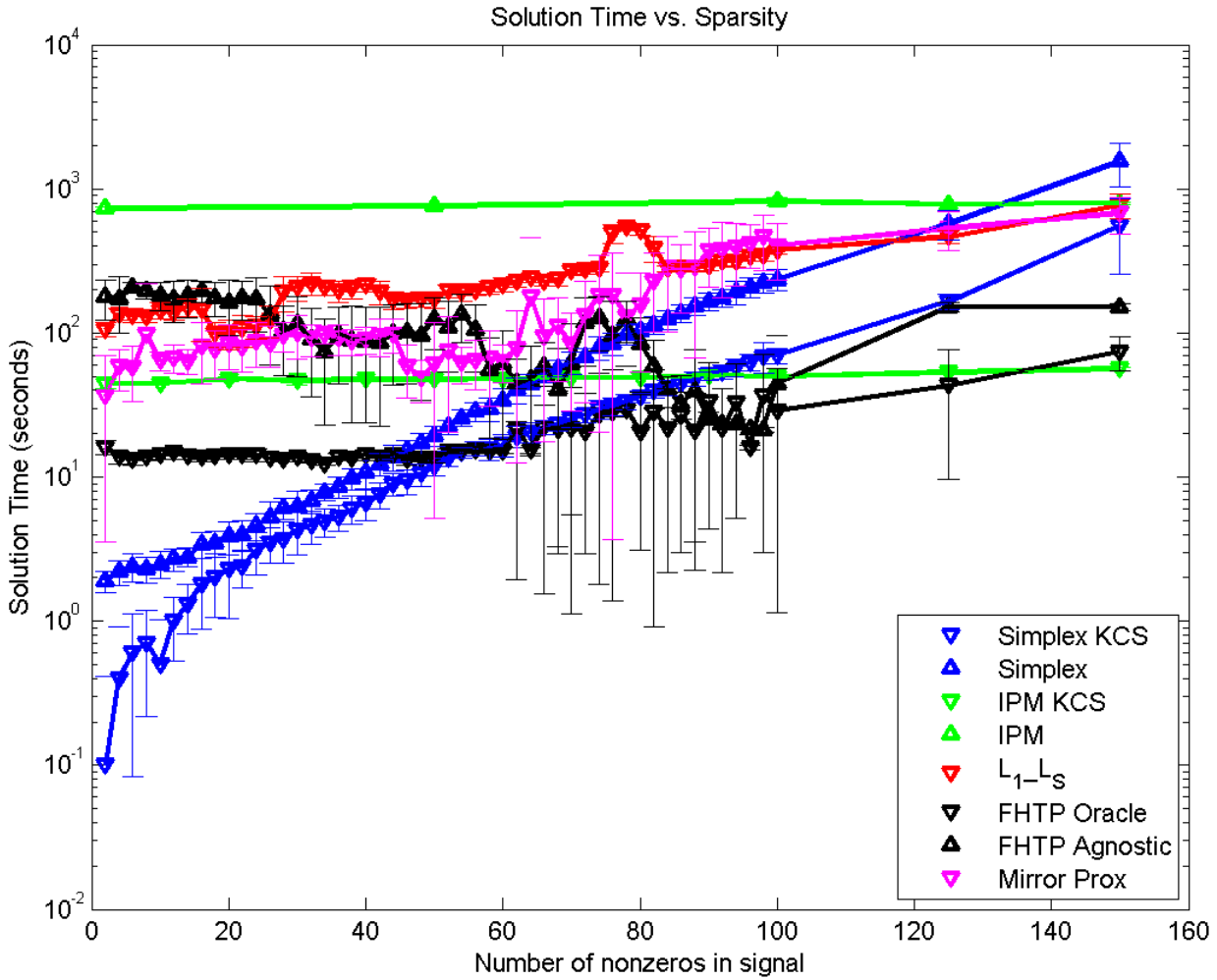
For these problems, we used $m_1 = 33$, $m_2 = 34$, $n_1 = 141$, $n_2 = 142$, and various values of k.

Again, the straightforward linear programming problems were solved by LOQO and SIMPO.

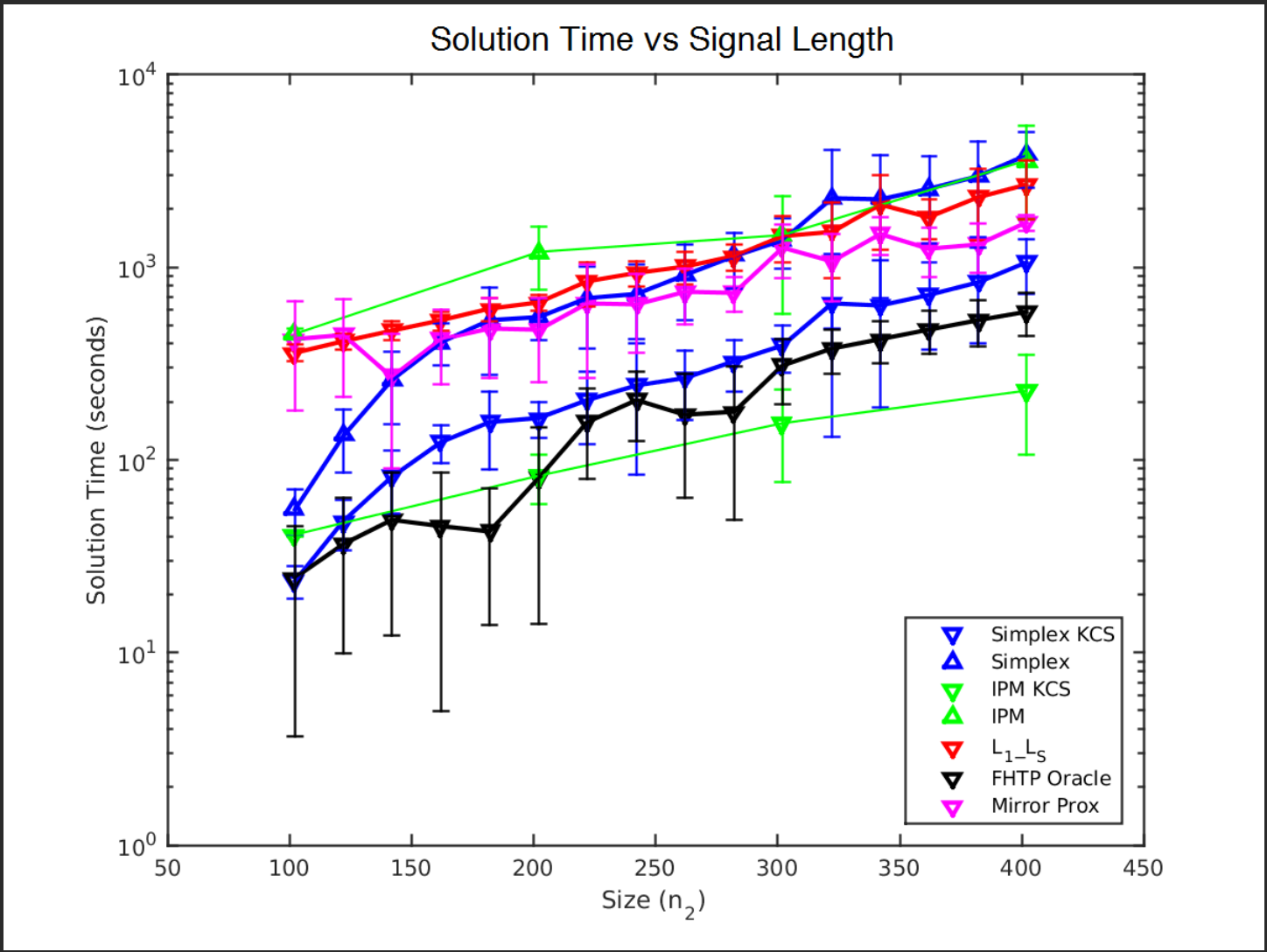
For the Kronecker sensing problems, the matrices A and B were generated so that their elements are independent standard Gaussian random variables.

For the vector sensing problems, the corresponding matrix $\mathbf{U}$ was used.

We also ran some publicly-available state-of-the-art codes: *L_1_L_s*, *FHTP*, and *Mirror Prox*.



$m = 1{,}222$, $n = 20{,}022$. Error bars represent one standard deviation.



$m = 1{,}122,\ k = 100,\ n = 141 \times n_2.$

Conclusion

SPARSITY MATTERS!!!

Thank You!

And Let The Fun Begin...