Hyper-sparsity in the revised simplex method and how to exploit it

Julian Hall

Ken McKinnon

School of Mathematics

University of Edinburgh

9^{th} October 2003

Overview

- Theory of linear programming and the simplex method
- The revised simplex method
- What is hyper-sparsity and how can it be exploited?
- Results, applications, conclusions and extensions

Solving linear programming (LP) problems

$$\begin{array}{rl} \text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m. \end{array}$$

Geometric view:

- The **feasible region** is the surface of a polyhedron in $I\!R^n$.
- **Results:**
  - At any vertex the m equations and $n - m$ bounds are satisfied exactly.
  - There is an optimal solution at a vertex of the feasible region.

Algebraic view

- For the system of equations in an LP problem,

$$A\boldsymbol{x} = \boldsymbol{b},$$

  if the variables are partitioned into index sets
  - $\mathcal{B}$ of m variables $\boldsymbol{x}_B$
  - $\mathcal{N}$ of $n - m$ variables $\boldsymbol{x}_N$,

  and the columns of A as matrices B and N (such that B is nonsingular) then

$$B\boldsymbol{x}_B + N\boldsymbol{x}_N = \boldsymbol{b} \quad \Rightarrow \quad \boldsymbol{x}_B = \hat{\boldsymbol{b}} - \hat{N}\boldsymbol{x}_N \quad \text{where} \quad \hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b} \quad \text{and} \quad \hat{N} = B^{-1}N.$$

- **Results:**
  - When $\boldsymbol{x}_N = \boldsymbol{0}$, such a partition corresponds to a vertex of the problem.
  - The columns of $\begin{bmatrix} \hat{N} \\ I \end{bmatrix}$ are the directions of the edges leading from the vertex.

Conditions for optimality

For a partition $\{\mathcal{B}, \mathcal{N}\}$ of the variables into

- **basic** variables $\boldsymbol{x}_B$ and $n - m$ **nonbasic** variables $\boldsymbol{x}_N$,

with **basis matrix** B, the corresponding components of $\boldsymbol{c}$ are

- the basic costs $\boldsymbol{c}_B$ and non-basic costs $\boldsymbol{c}_N$.

Eliminating the basic variables from the objective yields the **reduced problem**

$$\begin{array}{lrl} \text{minimize} & f \;=\; \hat{\boldsymbol{c}}_N^T \boldsymbol{x}_N \;+\; \hat{f} & \\ \text{subject to} & \hat{N}\,\boldsymbol{x}_N \;+\; \boldsymbol{x}_B \;=\; \hat{\boldsymbol{b}} & \\ & \boldsymbol{x}_N \geq \boldsymbol{0} \quad \boldsymbol{x}_B \geq \boldsymbol{0}, & \end{array}$$

where $\hat{\boldsymbol{c}}_N$ is the vector of **reduced costs** given by

$$\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T \hat{N} \quad \text{and} \quad \hat{f} = \boldsymbol{c}_B^T \hat{\boldsymbol{b}}.$$

Sufficient conditions for optimality at a vertex are

$$\hat{\boldsymbol{c}}_N \geq \boldsymbol{0}.$$

The simplex method

- Algorithm developed by Dantzig (1947)
- Standard simplex method (1948)
  - Maintains tableau of reduced equations
- Revised simplex method (1953)
  - Computes tableau rows and columns as required
  - Updates reduced costs and RHS.

The standard simplex method (1948)

	$\mathcal{N}$	RHS
$\mathcal{B}$	$\hat{N}$	$\hat{\boldsymbol{b}}$
0	$\hat{\boldsymbol{c}}_N^T$	$-\hat{f}$

In each iteration:

- Select the column q' of a nonbasic variable $q \in \mathcal{N}$ to be increased from zero.
- Find the row p of the first basic variable $p' \in \mathcal{B}$ to be zeroed.
- Exchange indices p' and q between sets $\mathcal{B}$ and $\mathcal{N}$.
- Update tableau corresponding to this **basis change**.
  - Scale row p by $\hat{N}_{pq'}$.
  - Eliminate $\hat{N}_{iq'}$, $i = 0, \ldots, m, i \neq p$.

Disadvantages:

- Prohibitively expensive since the matrix $\hat{N}$ is assumed to be full.
  - Storage requirement: mn memory locations.
  - Computation requirement: mn floating point operations per iteration
- Numerically unstable.

Revised simplex method

`CHUZC:` Scan the reduced costs $\hat{\boldsymbol{c}}_N$ for a good candidate q to enter the basis.

`FTRAN:` Form the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.

`CHUZR:` Scan the ratios $\hat{b}_i/\hat{a}_{iq}$ for the row p of a good candidate to leave the basis. Let $\alpha = \hat{b}_p/\hat{a}_{pq}$.
Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$.

`BTRAN:` Form $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$.

`PRICE:` Form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$.
Update reduced costs $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \hat{c}_q\hat{\boldsymbol{a}}_p^T$.

`If` (growth in factors) `then`

    `INVERT:` Form a factored representation of B^{-1}.

`else`

    `UPDATE:` Update the factored representation of B^{-1} corresponding to the basis change.

`end if`

Factored representation of B^{-1}: INVERT

- INVERT determines sparsity-preserving LU decomposition of basis matrix B_0.
- Nontrivial columns in decomposition give

$$B_0^{-1} = \prod_{k=k_I}^{1} E_k \quad \text{where} \quad E_k = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & \boldsymbol{\eta}_k & & & \\ & & 1 & & & & \\ & & & \eta_k & & & \\ & & & & 1 & & \\ & & & \boldsymbol{\eta}_k & & \ddots & \\ & & & & & & 1 \end{bmatrix} \leftarrow \text{row } p_k$$

Factored representation of B^{-1}: UPDATE

- Basis change results in column p of B being replaced by $\boldsymbol{a}_q$:

$$\begin{aligned} B &:= B - (B\boldsymbol{e}_p - \boldsymbol{a}_q)\boldsymbol{e}_p^T \\ &= B(I - (\boldsymbol{e}_p - \hat{\boldsymbol{a}}_q)\boldsymbol{e}_p^T) \end{aligned}$$

- Product of terms $I - (\boldsymbol{e}_p - \hat{\boldsymbol{a}}_q)\boldsymbol{e}_p^T$ over the sequence of k_U iterations following INVERT is the matrix E_U for which

$$E_U^{-1} = \prod_{k=k_I+k_U}^{k_I+1} E_k$$

- Hence

$$B^{-1} = E_U^{-1} B_0^{-1} = \prod_{k=k_I+k_U}^{1} E_k$$

- The set of data $\{p_k, \eta_k, \boldsymbol{\eta}_k\}_{k=1}^{k_I+k_U}$ is referred to as the **eta file**.
- Eventually more efficient to perform INVERT than UPDATE

Sparse FTRAN and BTRAN

- In FTRAN, $B\boldsymbol{x} = \boldsymbol{r}$ may be solved by transforming $\boldsymbol{r}$ into $\boldsymbol{x}$ as follows.

  `do` $k = 1,\ k_I + k_U$

  $\quad r_{p_k} := r_{p_k}/\eta_k$

  $\quad \boldsymbol{r} := \boldsymbol{r} - r_{p_k}\boldsymbol{\eta}_k$

  `end do`

- In BTRAN, $B^T\boldsymbol{x} = \boldsymbol{r}$ may be solved by transforming $\boldsymbol{r}$ into $\boldsymbol{x}$ as follows.

  `do` $k = k_I + k_U,\ 1,\ -1$

  $\quad r_{p_k} := (r_{p_k} - \boldsymbol{r}^T\boldsymbol{\eta}_k)/\eta_k$

  `end do`

What is hyper-sparsity?

- Hyper-sparsity exists when the results of matrix-vector operations in the revised simplex method are sparse.
  - The pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$ is sparse.
  - The 'update $\boldsymbol{\pi}$' $\boldsymbol{\pi}^T = \boldsymbol{e}_p^T B^{-1}$ is sparse.
  - The pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$ is sparse.
- Exploit hyper-sparsity both when forming and using these vectors.
- Hyper-sparsity in general linear systems identified by Gilbert and Peierls (1988).

FTRAN with sparse RHS

- In general, when solving LP problems, $B\boldsymbol{x} = \boldsymbol{r}$ has a **sparse RHS** $\boldsymbol{a}_q$.
- $B\boldsymbol{x} = \boldsymbol{r}$ may be solved by transforming $\boldsymbol{r}$ into $\boldsymbol{x}$ as follows.

```
do k = 1, k_I + k_U
    if (r_{p_k} .ne. 0) then
        r_{p_k} := r_{p_k} / η_k
        r := r − r_{p_k} η_k
    end if
end do
```

- In presence of hyper-sparsity, the dominant cost is the test for zero.

Hyper-sparse FTRAN

- Split FTRAN into two operations

$$\tilde{\boldsymbol{a}}_q = B_0^{-1}\boldsymbol{a}_q \qquad \text{(I-FTRAN)}$$

  followed by

$$\hat{\boldsymbol{a}}_q = E_U^{-1}\tilde{\boldsymbol{a}}_q. \qquad \text{(U-FTRAN)}$$

- Most tests for zero occur in I-FTRAN
- Perform U-FTRAN using standard technique.

Hyper-sparse I-FTRAN

- During each INVERT:

  For each row $i = 1, \ldots, m$,

  - record the index k of the *first* eta for which $p_k = i$
  - record the index k of the *second* eta for which $p_k = i$

- During each I-FTRAN:

  Let $\mathcal{R}$ be the set of indices of nonzeros in $\boldsymbol{r}$.

  Let $\mathcal{E}$ be the set of etas with pivots corresponding to nonzeros in $\boldsymbol{r}$.

  1 Scan $\mathcal{E}$ to determine the index k of the next eta to be applied.

  If k is undefined then I-FTRAN is complete.

  Apply eta k:

  add to $\mathcal{R}$ the indices of any RHS entries where nonzeros are created.

  add to $\mathcal{E}$ etas with pivots where nonzeros are created.

  Go to 1.

Hyper-sparse CHUZR

Recall: Scan the ratios $\hat{b}_i/\hat{a}_{iq}$

- Following hyper-sparse FTRAN, the indices of the nonzeros in $\hat{a}_{iq}$ are known.

Hyper-sparse BTRAN with unit RHS

- Recall, $B^T\boldsymbol{x} = \boldsymbol{r}$ may be solved by transforming $\boldsymbol{r}$ into $\boldsymbol{x}$ as follows.

  `do` $k = k_I + k_U,\ 1,\ -1$
  $\quad r_{p_k} := (r_{p_k} - \boldsymbol{r}^T\boldsymbol{\eta}_k)/\eta_k$
  `end do`

- When $\boldsymbol{x}$ is sparse, most inner products involve only zero components of $\boldsymbol{r}$.
- No way to exploit hyper-sparsity properly with 'column-wise' eta file.
- Split BTRAN into two operations

$$\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1} \qquad \text{(U-BTRAN)}$$

  followed by

$$\boldsymbol{\pi}^T = \tilde{\boldsymbol{\pi}}^T B_0^{-1}. \qquad \text{(I-BTRAN)}$$

- For I-BTRAN
  - After INVERT: Form a 'row-wise' copy of the eta file.
  - Pass row-wise eta file to hyper-sparse forward solution code.

Hyper-sparse U-BTRAN ($\tilde{\boldsymbol{\pi}}^T = \boldsymbol{e}_p^T E_U^{-1}$)

- Observe
  - RHS is $\boldsymbol{e}_p$, where p is the index of the pivotal row.
  - In U-BTRAN,

    $\boldsymbol{r} = \boldsymbol{e}_p$
    do $k = k_I + k_U,\ k_I + 1,\ -1$
    $r_{p_k} := (r_{p_k} - \boldsymbol{r}^T \boldsymbol{\eta}_k)/\eta_k$
    end do
    $\tilde{\boldsymbol{\pi}} = \boldsymbol{r}$
  - Nonzeros in $\tilde{\boldsymbol{\pi}}$ occur in set $\mathcal{P}$ of indices of rows which were pivotal since INVERT.
  - $|\mathcal{P}| \leq k_U \ll m$.
- Maintain dense representation $\mathtt{E_P}$ of entries in UPDATE etas for rows $p \in \mathcal{P}$.
- Determine nonzeros in $\tilde{\boldsymbol{\pi}}$ as the result of sequence of inner products between sparse vector $\boldsymbol{r}$ and dense columns of $\mathtt{E_P}$.

Hyper-sparse PRICE

Recall: $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}^T N$.

- Form $\hat{\boldsymbol{a}}_p^T$ as a linear combination of rows of N corresponding to nonzeros in $\boldsymbol{\pi}^T$.
- Maintain list of indices of nonzeros in $\hat{\boldsymbol{a}}_p^T$.

Requires row-wise representation of N to be maintained.

Hyper-sparse CHUZC

Recall: Reduced costs updated according to

$$\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \hat{c}_q \hat{\boldsymbol{a}}_p^T$$

- $\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T B^{-1} N$ is typically dense
- $\hat{\boldsymbol{a}}_p^T$ is sparse—few reduced costs change
- Initially: full CHUZC to form list of ($\leq s$) good candidates
- Each iteration:
  - Form list of ($\leq s$) good candidates from those with changed reduced cost
  - Merge lists to form new list of ($\leq s$) good candidates
- Periodically: full CHUZC to pick up good candidates which have never changed

Hyper-sparse (product-form) UPDATE

Recall: B^{-1} is represented as $B^{-1} = E_U^{-1} B_0^{-1}$, where B_0^{-1} is obtained by INVERT.

Eta file for E_U^{-1} requires packed form of $\boldsymbol{\eta} = \begin{bmatrix} \hat{a}_{1q} \\ \vdots \\ \hat{a}_{p-1\,q} \\ -1/\hat{a}_{pq} \\ \hat{a}_{p+1\,q} \\ \vdots \\ \hat{a}_{nq} \end{bmatrix}$

This is formed at almost no cost during hyper-sparse CHUZR.

Hyper-sparse (Tomlin) INVERT

- Triangularise B_0
- Factorise 'bump' using fixed column order
  - Columns of the active submatrix computed as required
  - Column k requires the solution of $(\prod_{i=1}^{k-1} L_i)\hat{\boldsymbol{a}}_k = \boldsymbol{a}_k$
  - Apply techniques for hyper-sparse FTRAN

Speedup of EMSOL for problems exhibiting hyper-sparsity

	speedup in total solution time and computational components							
Problem	Solution	CHUZC	I-FTRAN	CHUZR	I-BTRAN	U-BTRAN	PRICE	INVERT
80BAU3B	3.34	3.05	5.13	1.93	3.51	6.72	6.06	1.34
FIT2P	1.75	18.91	1.30	0.93	12.22	3.59	13.47	0.87
GREENBEA	2.71	1.33	1.30	1.13	3.60	19.87	3.45	2.83
GREENBEB	2.44	1.39	1.35	1.21	3.69	21.88	3.44	2.78
STOCFOR3	1.85	4.47	1.14	0.96	7.26	56.99	7.61	3.16
WOODW	3.40	1.82	1.70	1.30	4.17	11.72	5.14	1.53
DCP1	3.25	1.58	2.36	1.19	3.87	6.25	6.71	1.70
DCP2	5.32	1.60	8.24	2.51	6.21	13.99	6.20	8.63
CRE-A	3.05	2.54	4.00	1.72	3.64	6.50	4.48	1.14
CRE-C	2.89	2.88	4.67	1.97	3.58	6.53	4.97	1.08
KEN-11	22.84	19.36	98.04	13.93	27.22	9.90	66.36	1.02
KEN-13	12.12	6.31	104.09	7.31	12.87	9.19	17.60	0.94
KEN-18	15.27	6.63	263.94	15.27	13.91	13.07	19.92	1.01
PDS-06	17.48	15.25	24.07	3.57	21.58	35.76	28.18	1.02
PDS-10	10.36	10.67	11.24	1.85	16.60	49.99	17.55	0.96
PDS-20	10.35	8.58	5.96	1.68	14.33	189.19	15.40	1.44
Mean	5.21	4.38	7.03	2.28	7.64	15.44	9.71	1.55

Comparison of EMSOL with CPLEX and SOPLEX

Problem	m	EMSOL Solution time CPU (s)	CPLEX 7.0 (Devex) Solution time CPU (s)	CPLEX 7.0 (Devex) EMSOL solution speedup	SOPLEX 1.2 (Devex) Solution time CPU (s)	SOPLEX 1.2 (Devex) EMSOL solution speedup
80BAU3B	2262	9.43	14.39	1.53	20.50	2.17
FIT2P	3000	67.34	42.08	0.62	1058.79	15.72
GREENBEA	2392	24.63	21.18	0.86	150.53	6.11
GREENBEB	2392	19.55	18.31	0.94	76.36	3.91
STOCFOR3	16675	201.21	41.15	0.20	154.11	0.77
WOODW	1098	3.09	3.99	1.29	21.41	6.93
DCP1	4950	14.64	16.30	1.11	330.35	22.56
DCP2	32388	483.64	1412.89	2.92	1507.40	3.12
CRE-A	3516	4.49	6.29	1.40	5.07	1.13
CRE-C	3068	3.50	5.91	1.69	3.39	0.97
KEN-11	14694	9.19	13.57	1.48	46.99	5.11
KEN-13	28632	100.77	91.41	0.91	417.26	4.14
KEN-18	105127	1421.50	786.08	0.55	5167.60	3.64
PDS-06	9881	8.20	5.49	0.67	44.22	5.39
PDS-10	16558	83.83	26.92	0.32	158.79	1.89
PDS-20	33874	1059.19	238.48	0.23	1754.17	1.66
Mean				0.84		3.47

How much of all this is new?

Very little is published on the (detailed) computational techniques which make commercial codes run fast.

- Storing row-wise copy of L to speed up BTRAN suggested by Reid.
- CPLEX now known to exploit hyper-sparsity using Gilbert-Peierls algorithm.

Applications

- When solving single hyper-sparse LP problems, revised simplex method is often better than an interior point method.
- Revised simplex method is preferable when a family of problems is to be solved
  - (Mixed) Integer programming (MILP) by branch-and-bound/cut/price
  - Successive linear programming (SLP)
  - Decomposition for stochastic and decentralised planning problems (LP).
- Our simplex techniques have been sold to software companies operating in the following areas
  - Petroleum engineering (SLP)
  - Animal feed formulation (LP/MILP)
  - Chemical engineering (LP)

Conclusions and extensions

- Considerable improvement in efficiency shown for a significant class of problems.
- Speedup often limited by computational cost of the relatively small number of iterations where hyper-sparsity is not exhibited.
  - Can this number of iterations be reduced by modified column and row selection strategies?
- Heuristic approach to hyper-sparse I-FTRAN and I-BTRAN may be inefficient for very large problems.
  - May need Gilbert-Peierls.
- Parallelisation of the revised simplex method.