

1 Solves with Sequences of B and B^T

Each iteration, simplex methods and the reduced-gradient method need to solve systems of the form $Bx = v$ and $B^T y = w$ (at least one of each), where B is the current square, nonsingular basis. Large problems may require tens of thousands of iterations. It is therefore normal to use *direct* methods rather than *iterative* methods as linear-system solvers. Typically this means sparse LU factorization methods.

1.1 Basis Updating

In general it is too expensive to compute brand new LU factors for every B . Instead, we take advantage of the fact that each B is almost the same as the one before. (One column of B has been replaced by a column of A .) In some way we *update* the current factors of B . Sometimes we keep the existing factors (unaltered) and accumulate updating information on the side. Other methods modify the existing factors themselves.

Suppose a column \bar{a} replaces the p th column of B to give the next basis \bar{B} . Then

$$\bar{B} = B + (\bar{a} - a)e_p^T, \quad (1)$$

where e_p is the p th column of the identity. Given some kind of factorization of B , we need to obtain a factorization of \bar{B} . We study several methods that have varying degrees of stability, efficiency, and ease of implementation.

The software that computes and updates factors of B is the “engine” of simplex-based solvers. It is called the Basis Factorization Package (BFP).

2 PFI and EFI

For many years, large-scale implementations of the simplex method used what was called “the product form of inverse”, in which the solution of $Bx = b$ was thought of (and implemented) as a product of the form

$$x = E_l \dots E_2 E_1 b,$$

where each matrix E_j was the identity except for one (somewhat sparse) column v_j . After a basis change, the solution of $\bar{B}\bar{x} = \bar{b}$ would be computed as

$$\bar{x} = E_{l+1} E_l \dots E_2 E_1 \bar{b},$$

and similarly for systems involving B^T and \bar{B}^T . The method allowed efficient use of auxiliary storage, with the growing list of vectors $\{v_1, v_2, \dots\}$ being accessed sequentially (either forwards or backwards).

Eventually the terms arising from an initial basis B_0 were constructed from LU factors of B_0 , leading to the name “elimination form of inverse” [14]. However, it is wiser to eliminate the word “inverse” from all discussions. We describe the updating methods in terms of *factorizations* of B_0 , B , and \bar{B} . (The above matrices E_j are inverses of the triangular matrices T_j discussed next.)

3 The Product-Form Update

We may factorize \bar{B} in (1) as $\bar{B} = BT$, where

$$Bv = \bar{a}, \quad T = I + (v - e_p)e_p^T.$$

Note that T is the identity matrix with the p th column replaced by v . Most importantly, T is a *permuted triangle* and it is easy to solve a system $Ty = d$. After k updates we have

$$B_k = B_0 T_1 T_2 \dots T_k,$$

where B_0 may be treated as a “black box”, and the factors T_j may be stored sequentially as a sequence of sparse vectors $\{v_j\}$.

Since B_k contains factorizations of all previous bases, it is clear that if any previous basis is ill-conditioned then the updated factorization involves ill-conditioned factors, even if B_k itself becomes well-conditioned. The PF update does not have the ability to “recover” if iterations pass through one or more unfriendly bases.

A practical safeguard is to test each update matrix T . If the diagonal element v_p is small, the update should be abandoned because T will be ill-conditioned. For example,

$$\text{if } |v_p|/\|v\|_\infty \geq 10^{-5} \text{ then Update else Factorize } \bar{B}.$$

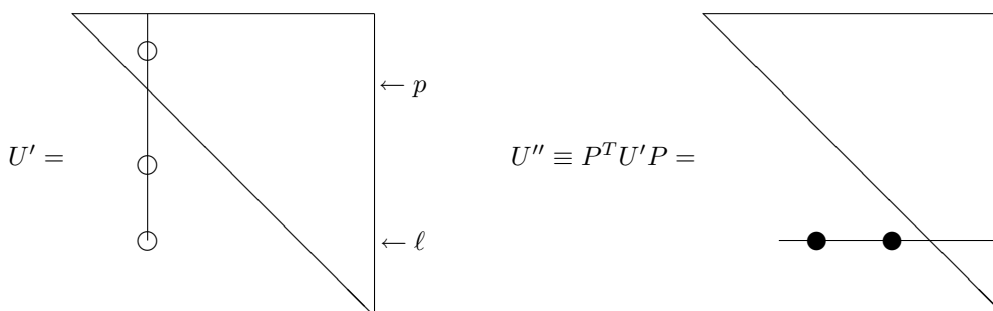
Such a test is “prudent”, but we cannot say that it makes the PF update a stable method. A tolerance nearer 10^{-2} or 10^{-1} would help, but then the update would be rejected rather often.

4 The Bartels-Golub-Reid Update

The need for a stable update was emphasized by Bartels and Golub [1], and an efficient (but dense) implementation was proposed in [2]. The key idea was to update LU factors of B by replacing a column of U and restoring U to triangular form using stabilized elimination matrices. The updates to L may be kept in product form, but U must be updated explicitly. A satisfactory sparse implementation, LA05, was developed by Reid [15], and a similar implementation is included in LUSOL [8, 18]. LA05 was influenced by Forrest and Tomlin [6] and Saunders [17]. Reid later updated LA05 as LA15 [16].

To illustrate, suppose $B = LU$ and note that $\bar{B} = LU'$, where U' is identical to U except for its p th column \bar{u} , which solves the system $L\bar{u} = \bar{a}$. If U' is already triangular, we are lucky—the new factors are on hand. More generally, the last nonzero element of \bar{u} will be in row ℓ with $\ell > p$, and to restore triangularity we need to find an LU factorization of U' .

As a first step, let P be a cyclic permutation that moves the p th column and row of U' to position ℓ and shifts the intervening columns and rows forward. We see that U' is upper triangular except for one *column*, while the permuted matrix U'' is upper triangular except for one *row*, as shown in the following picture [17]:



We refer to the exceptional column and row as *spikes*. The circles remind us that the column and row spikes are *sparse*. We may be lucky again and find that the row spike has no subdiagonal nonzeros (U'' will be already triangular), but in general we need to find a factorization $U'' = \tilde{L}\tilde{U}$.

Ideally the existing diagonals in rows $p:\ell-1$ will be large enough to serve as pivots, but to ensure stability we must allow *row interchanges*. The factor \tilde{L} is constructed as a product

of *stabilized elementary transformations* whose essential part is one of the 2×2 matrices

$$M = \begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix} \quad \text{or} \quad \widetilde{M} = \begin{pmatrix} & 1 \\ 1 & \mu \end{pmatrix}$$

as principal submatrices of $m \times m$ identities, where the choice is made to keep μ bounded ($|\mu| \leq \text{UpdateTol} = 10, 5, \text{ or } 2$, say) while optimizing sparsity to some extent. The updated factorization is then $\bar{B} = \bar{L}\bar{U}$ with $\bar{L} = L\widetilde{L}$ (in product form) and $\bar{U} = \widetilde{U}$ (explicit).

The row operations involved in factorizing U'' require a fairly complex data structure. In LA05 and LUSOL, the nonzeros in U are stored *row-wise* and initially in natural order (so that solving a system with U or U^T involves a sweep through contiguous memory). During updates, several rows may be modified when the column spike \bar{u} is inserted, and further rows may be modified during the factorization of U'' (depending how often \widetilde{M} is chosen). Modifications are made in place when possible. If too many nonzeros are generated in a particular row, the row must be moved to the beginning of free storage. Periodic garbage collection reclaims the unused storage.

In the context of Bartels-Golub-Reid (BGR) updates, the initial LU factorization should ensure that L is well-conditioned. With `UpdateTol` in the range $[1, 10]$, the modified L factors tend to be well-conditioned also. Although the storage of U becomes increasingly fragmented, the benefit is that hundreds of BGR updates can be carried out safely (especially if `UpdateTol` is 5 or 2, say). The decision to refactorize rather than update may be based on storage concerns rather than loss of precision.

5 The Forrest-Tomlin Update

The update proposed by Forrest and Tomlin in 1972 [6] followed Bartels and Golub in updating L in product form and U explicitly, but with emphasis on dealing efficiently with *sparse* LU factors. Algebraically, the FT update is equivalent to the BGR update with the restriction that $\ell = m$ and row interchanges are not allowed in the factorization $U'' = \widetilde{L}\widetilde{U}$.

Note that $\ell = m$ means P moves the column spike to the end of U'' , where it is no longer a spike. Also, “no row interchanges” means the row spike is simply deleted from the existing U . Hence, the FT update can be implemented efficiently with a *column-oriented* data structure for U .

To derive the FT update directly, let δ be the p th diagonal of U and consider the system $U^T r = \delta e_p$. Then U and r have the following structure:

$$U = \begin{pmatrix} U_1 & u & U_2 \\ & \delta & v^T \\ & & U_3 \end{pmatrix}, \quad r = \begin{pmatrix} 0 \\ 1 \\ s \end{pmatrix},$$

where $U_3^T s = -v$. We can now obtain a triangular factorization $U = R\widetilde{U}$ in which most of the p th row of U is eliminated:

$$R = \begin{pmatrix} I & & \\ & 1 & -s^T \\ & & I \end{pmatrix}, \quad \widetilde{U} = \begin{pmatrix} U_1 & u & U_2 \\ & \delta & \\ & & U_3 \end{pmatrix}.$$

We already have $L\bar{u} = \bar{a}$, and solving $LR\bar{u} = \bar{a}$ alters only the p th element of \bar{u} to give $\bar{\delta} = r^T \bar{u}$. With suitable permutations, \bar{u} and the modified factors follow:

$$\bar{u} = \begin{pmatrix} \bar{u}_1 \\ \bar{u}_p \\ \bar{u}_3 \end{pmatrix}, \quad \bar{L} = LR, \quad \bar{U} = \begin{pmatrix} U_1 & U_2 & \bar{u}_1 \\ & U_3 & \bar{u}_3 \\ & & \bar{\delta} \end{pmatrix}.$$

In sequence, the steps are

Solve $L\bar{u} = \bar{a}$.
 Find $\delta = U_{pp}$ and solve $U^T r = \delta e_p$.
 Set $\bar{\delta} = r^T \bar{u}$.
 Delete the p th column and row of U .
 Define $\bar{L} = LR$ and insert the new last column with $\bar{\delta}$ to define \bar{U} .

Bearing in mind that this is a restricted form of the BGR update, we know that the numerical stability of the FT update depends on the size of the elements of s in the triangular matrix R . (These are the multipliers μ from the equivalent sequence of 2×2 matrices M .) Thus, a practical safeguard could take the form

$$\text{if } \|s\|_\infty \leq 10^5 \text{ then Update else Factorize } \bar{B}.$$

As with the PF update, such a test is prudent but not completely reliable unless the tolerance is chosen much closer to 1. Nevertheless, the ease of implementation (and the existence of some sort of stability check) means that the FT update has been adopted in virtually *all* sparse implementations of the simplex method. Efficiency wins!

6 The Block-LU Update

As before, suppose B_0 is a given $m \times m$ starting basis. Let B be a later basis obtained by replacing various columns of B_0 one by one, using columns of the larger matrix A . In general, B is a matrix in which the columns of $V \equiv (v_1 \ v_2 \ \dots)$ have replaced columns p_1, p_2, \dots of B_0 . If a sparse LU factorization

$$B_0 = L_0 U_0 \tag{2}$$

is available, the solution of a system $Bx = b$ may be obtained from a larger system

$$\begin{pmatrix} B_0 & V \\ E^T & \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{3}$$

where the columns of E are columns p_1, p_2, \dots of the $m \times m$ identity. The larger system may be solved using the block-triangular factorization

$$\begin{pmatrix} B_0 & V \\ E^T & \end{pmatrix} = \begin{pmatrix} L_0 & \\ Z^T & I \end{pmatrix} \begin{pmatrix} U_0 & Y \\ & C \end{pmatrix}, \tag{4}$$

where we need $L_0 Y = V$, $U_0^T Z = E$, and $C = -Z^T Y (= -E^T B_0^{-1} V)$. Thus, Y and Z are solutions of sparse systems and may be stored column-wise as $m \times p$ sparse matrices, while C is $p \times p$ and may be treated as a dense matrix. After k updates, the dimension p satisfies $p \leq k$. In practice the same columns tend to come and go, and after 100 updates we might have $p \approx 80$. Note that C is the Schur complement of B_0 in the matrix on the left of (4), but it is more important to regard (4) as a block-LU factorization.

6.1 Solving with B and B^T

From (2)–(4), we can solve $Bx = b$ by the following sequence:

$$\begin{aligned} &\text{Solve } L_0 w = b \\ &\text{Solve } Cz = -Z^T w \\ &\text{Solve } U_0 y = w - Yz \\ &\text{Extract } x \text{ from the appropriate parts of } \begin{pmatrix} y \\ z \end{pmatrix}. \end{aligned} \tag{5}$$

Similarly, we can solve $B^T y = c$ as follows:

$$\begin{aligned} &\text{Permute } \begin{pmatrix} c \\ 0 \end{pmatrix} \text{ to } \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \text{ to match rows of } B^T \\ &\text{Solve } U_0^T w = c_1 \\ &\text{Solve } C^T z = c_2 - Y^T w \\ &\text{Solve } L_0^T y = w - Zz. \end{aligned} \tag{6}$$

6.2 Updating Y , Z , C

Suppose the basis matrices B arise when column a_s of A replaces column a_r for a sequence of sensible pairs (s, r) . There are four updates to deal with:

1. Usually a_s is a new column of A (not in B_0 or V) and a_r is column of B_0 . Then Y , Z , C expand.
2. Sometimes a_s is a new column and a_r is a column of V . All dimensions remain the same.
3. Sometimes a_s is a column of B_0 that was replaced earlier, and a_r is another column of B_0 . All dimensions remain the same.
4. Sometimes a_s is a column of B_0 that was replaced earlier, and a_r is a column of V . Then Y , Z , C shrink.

6.3 Origins

The block-LU (BLU) update above was first described in [7] (where it was called the Schur-complement update). It was motivated by the work of Bisschop and Meeraus [3, 4], who used the related block factorization

$$\begin{pmatrix} B_0 & V \\ E^T & \end{pmatrix} = \begin{pmatrix} B_0 & \\ E^T & I \end{pmatrix} \begin{pmatrix} I & Y \\ & C \end{pmatrix}, \quad B_0 Y = V \quad (7)$$

without storing Y (at the expense of additional solves with B_0 and B_0^T). The Schur complement C is the same as before. In [3, 4], C^{-1} was updated explicitly, but it is more stable to update a factorization of C .

In [5], this approach was implemented on a Cray Y-MP and tested on a set of 30 moderate-sized LP problems. Each column of Y was stored in sparse format if less than 40% dense; otherwise in dense format. Dense orthogonal factors $QC = R$ were maintained by our QRMOD code using sequences of plane rotations. On most problems, the BLU updates were found to be faster than our LUSOL implementation of the Bartels-Golub-Reid update. The vectors in Y were often rather dense (as for the Product-Form update), but it did save time to store them and avoid double solves with B_0 and B_0^T .

Our Fortran 77 code LUMOD [18] maintains a dense factorization $LC = U$ when rows and columns of C are added, deleted or replaced, as needed in section 6.2. Note that L is *square* and well-conditioned, and U is upper triangular. LUMOD stores both factors row-wise in 1D arrays. It was designed for future implementations of the BLU update.

6.4 QPBLU

Block-LU updates of (symmetric) KKT systems have been employed in a Fortran 95 QP solver QPBLU implemented by Hanh Huynh for her thesis [9]. QPBLU uses an inertia-controlling active-set method to solve convex QP problems of the form

$$\begin{aligned} \min_x \quad & c^T x + \frac{1}{2} x^T H x \\ & Ax = b, \quad l \leq x \leq u, \end{aligned}$$

where H is diagonal and semidefinite. The KKT matrix for some current active set is factorized as

$$K_0 = \begin{pmatrix} H_0 & A_0^T \\ A_0 & \end{pmatrix} = L_0 U_0$$

and BLU updates are then applied as rows and columns are symmetrically added and deleted (to accommodate changes to A_0 and H_0):

$$\begin{pmatrix} K_0 & V \\ V^T & \end{pmatrix} = \begin{pmatrix} L_0 & \\ Z^T & I \end{pmatrix} \begin{pmatrix} U_0 & Y \\ & C \end{pmatrix}. \quad (8)$$

Sparse factors $K_0 = L_0U_0$ are obtained from the linear system packages LUSOL, MA57, PARDISO, SuperLU, or UMFPACK (where all except PARDISO can treat L_0 and U_0 separately). An F95 version of LUMOD is used to maintain dense factors $LC = U$ for the BLU updates. Further BLU updates are used to accommodate QP Hessians of the form

$$H_\ell = R_\ell^T H_0 R_\ell, \quad \text{where} \quad R_\ell = (I + u_1 v_1^T)(I + u_2 v_2^T) \dots (I + u_\ell v_\ell^T)$$

for a sequence of vector pairs $(u_1, v_1), (u_2, v_2), \dots$, as required by the nonlinear optimizer SNOPT for its sequence of convex QP subproblems. QPBLU was designed to complement SQOPT for subproblems involving thousands of degrees of freedom (where A_0 has many more columns than rows).

Note that MA57 produces symmetric factors $K_0 = L_0 D_0 L_0^T$, where D_0 is symmetric indefinite and block-diagonal with blocks of order 1 or 2. To take advantage of these symmetric factors, we could replace (8) by

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} L_0 & \\ Y^T & I \end{pmatrix} \begin{pmatrix} D_0 & \\ & C \end{pmatrix} \begin{pmatrix} L_0^T & Y \\ & I \end{pmatrix}, \quad (9)$$

where $L_0 W = V$, $D_0 Y = W$, $C = D - Y^T W$, and D allows for more general updates.

6.5 QPBLUR

On some of the test problems reported in [9], QPBLU encountered KKT systems that were essentially singular. Since it is difficult to implement a ‘‘KKT repair’’ procedure for each linear system package, we considered *primal and dual regularization* of the QP problem. A new solver QPBLUR was developed by Christopher Maes for strictly convex QP problems of the form

$$\begin{aligned} \min_{x, y} \quad & c^T x + \frac{1}{2} x^T H x + \frac{1}{2} \delta \|x\|_2^2 + \frac{1}{2} \mu \|y\|_2^2 \\ & Ax + \mu y = b, \quad l \leq x \leq u, \end{aligned}$$

where δ and μ are small scalars of order 10^{-6} to 10^{-12} . Given any x that satisfies the bounds $l \leq x \leq u$, we can choose y to satisfy the constraints $Ax + \mu y = b$. Thus, a single-phase active-set algorithm becomes possible. Search directions are computed from KKT systems of the form

$$\begin{pmatrix} H_M + \delta I & A_M^T \\ A_M & -\mu I \end{pmatrix} \begin{pmatrix} \Delta x_M \\ -\Delta y \end{pmatrix} = \begin{pmatrix} -g_M + A_M^T y \\ 0 \end{pmatrix}, \quad (10)$$

where the subscript M refers to the variables that are moving. A major benefit is that the KKT system is nonsingular for *any active set*, and the BLU updates from QPBLU can still be used. For safety, a sequence of problems are solved with δ and μ reducing in stages from 10^{-6} to 10^{-12} . Preliminary results with a MATLAB implementation of QPBLUR were encouraging [12].

6.6 QPBLUR + BCL

Defining $r = \mu y$ and $\rho = 1/\mu$ above, we see that QPBLUR is really minimizing a quadratic penalty function subject to bounds:

$$\begin{aligned} \min_{x, r} \quad & c^T x + \frac{1}{2} x^T H x + \frac{1}{2} \delta \|x\|_2^2 + \frac{1}{2} \rho \|r\|_2^2 \\ & Ax + r = b, \quad l \leq x \leq u. \end{aligned}$$

We know that ρ must be very large to make $r = b - Ax$ small. Thus Chris Maes developed a ‘‘bound-constrained augmented Lagrangian’’ (BCL) version of QPBLUR for his thesis [11]. This involves a sequence of strictly convex QP problems of the form

$$\begin{aligned} \min_{x, r} \quad & c^T x + \frac{1}{2} x^T H x + \hat{y}^T r + \frac{1}{2} \delta \|x\|_2^2 + \frac{1}{2} \rho \|r\|_2^2 \\ & Ax + r = b, \quad l \leq x \leq u, \end{aligned}$$

where \hat{y} is an estimate of the dual variables, and the Lagrangian term $\hat{y}^T r$ has been added. After each problem is solved, if $\|r\|$ is sufficiently small, \hat{y} is updated. Otherwise, \hat{y} is retained and ρ is increased. This BCL approach is embodied in the LANCELOT solver [10] for general NLP problems. The benefit is that ρ might not need to be very large to achieve acceptably small $\|r\|$, and fewer QP problems (with changing \hat{y} or ρ) should suffice to solve the original QP accurately.

An F95 implementation of QPBLUR has been developed and incorporated into SNOPT as an alternative to SQOPT (for solving the sequence of QP problems arising from the major iterations of SNOPT's SQP method). Warm-starts are possible with \hat{y} and the active set from the previous major iteration (even though A changes when the constraints are nonlinear). KKT repair is not needed because the KKT systems are always nonsingular. As expected, QPBLUR gives much-improved efficiency on problems that have thousands of degrees of freedom at a solution [13, 11].

References

- [1] R. H. Bartels and G. H. Golub. The simplex method of linear programming using the LU decomposition. *Commun. ACM*, 12:266–268, 1969.
- [2] R. H. Bartels, J. Stoer, and Ch. Zenger. A realization of the simplex method based on triangular decompositions. In J. H. Wilkinson and C. Reinsch, editors, *Handbook for Automatic Computation, Volume II*, pages 219–239. Springer Verlag, Berlin, Heidelberg, New York, 1971.
- [3] J. Bisschop and A. Meeraus. Matrix augmentation and partitioning in the updating of the basis inverse. *Math. Program.*, 13:241–254, 1977.
- [4] J. Bisschop and A. Meeraus. Matrix augmentation and structure preservation in linearly constrained control problems. *Math. Program.*, 18:7–15, 1980.
- [5] S. K. Eldersveld and M. A. Saunders. A block-LU update for large-scale linear programming. *SIAM J. Matrix Anal. Appl.*, 13:191–201, 1992.
- [6] J. J. H. Forrest and J. A. Tomlin. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Math. Program.*, 2:263–278, 1972.
- [7] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Sparse matrix methods in optimization. *SIAM J. Sci. and Statist. Comput.*, 5:562–589, 1984.
- [8] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88/89:239–270, 1987.
- [9] H. M. Huynh. *A Large-scale Quadratic Programming Solver Based On Block-LU Updates of the KKT System*. PhD thesis, SCCM, Stanford University, 2008.
- [10] LANCELOT optimization software. <http://www.numerical.rl.ac.uk/lancelot/blurb.html>.
- [11] C. M. Maes. *A Regularized Active-Set Method for Sparse Convex Quadratic Programming*. PhD thesis, ICME, Stanford University, Nov 2010.
- [12] C. M. Maes and M. A. Saunders. An active-set convex QP solver based on regularized KKT systems. Presented at BIRS Workshop 09w5101, Advances and Perspectives on Numerical Methods for Saddle Point Problems, Banff, Alberta, Canada, Apr 12–17, 2009. <http://www.stanford.edu/group/SOL/talks.html>.
- [13] C. M. Maes and M. A. Saunders. QPBLUR: An active-set convex QP solver based on regularized KKT systems. Presented at RTRA STAE Workshop, Advanced Methods and Perspectives in Nonlinear Optimisation and Control, Toulouse, France, Feb 3–5, 2010. <http://www.stanford.edu/group/SOL/talks.html>.
- [14] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.
- [15] J. K. Reid. A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Math. Program.*, 24:55–69, 1982.
- [16] J. K. Reid. LA15 basis factorization package (thread-safe version of LA05). HSL 2007 Catalogue, <http://www.hsl.rl.ac.uk/specs/la15.pdf>, 2007.
- [17] M. A. Saunders. The complexity of LU updating. In R. S. Anderssen and R. P. Brent, editors, *The Complexity of Computational Problem Solving*, pages 214–230. University of Queensland Press, 1976. <http://www.stanford.edu/group/SOL/classics.html>.
- [18] SOL downloadable software. <http://www.stanford.edu/group/SOL/software.html>.