DYLP: a dynamic lp code

SFU-CMPT TR 1998-23

Lou Hafer

December, 1998

Abstract

DYLP is a linear programming code implementing a dynamic simplex algorithm similar to that outlined by Padberg in [5]. The code attempts to maintain a reduced active constraint system by regularly purging loose constraints and variables with unfavourable reduced costs, and adding violated constraints and variables with favourable reduced costs. In abstract, the code alternates between primal and dual simplex algorithms, using dual simplex to reoptimise after updating the constraint set and primal simplex to reoptimise after updating the variable set.

1 Introduction

DYLP is a linear programming (LP) package designed to be used as the underlying LP code in a branch-and-cut integer linear programming (ILP) code. It emphasises convenience of use by the client, particularly with respect to fixing variables and adding and deleting constraints and variables. The target user population is ILP algorithm developers; as such, DYLP emphasises controllability and convenience over efficiency and is capable of producing copious amounts of output for use in debugging.

DYLP implements a dynamic simplex algorithm along the lines set out by Padberg in [5, §6.6]. The core idea is that, at any given time, many of the constraints of a LP problem are loose, and many nonbasic variables are unlikely to ever be considered for pivoting because their reduced costs are very unfavourable. A rough outline of the algorithm, neglecting issues of unboundedness and infeasibility, is as follows.

From the problem supplied by the client, DYLP chooses an initial subset of constraints and variables to become the active system. This system is solved to optimality with primal simplex. DYLP then enters a minor loop where it deactivates variables whose reduced costs are worse than a threshold, activates variables whose reduced costs are favourable, and reoptimises the system with primal simplex. This minor loop is repeated until there are no more variables suitable for entry. Next, DYLP deactivates any loose constraints, activates any constraints which are violated at the present solution point, and reoptimises with dual simplex. On regaining optimality, it returns to the deactivate/activate variable loop. When there are no variables with favourable reduced costs and no violated constraints among the inactive variables and constraints, the solution is optimal.

The primal simplex algorithm used by DYLP is a two-phase algorithm. Phase I uses a dynamically modified objective to attain a primal feasible solution. Both phase I and phase II use a projected steepest edge (PSE) pricing algorithm outlined by Forrest & Goldfarb ([2], algorithm 'dynamic'). There are two antidegeneracy methods. The first, referred to as 'anti-degeneracy lite', attempts to resolve ties among degenerate pivots by choosing the pivot in such a way as to make tight a hyperplane which has a desireable alignment. The second, applied when the first takes too long to resolve the degeneracy, is a perturbation algorithm which builds on a method described by Ryan & Osborne [7].

The dual simplex algorithm is simpler, providing only a a second phase with dual steepest edge (DSE) pricing ([2], algorithm 'steepest 1') and an implementation of anti-degeneracy lite in the dual space. In the context of DYLP, it is a subordinate algorithm, used for reoptimisation after adding constraints, and hence there is no motivation to provide a dual phase I.

The active and inactive constraint systems are maintained with the CONSYS subroutine library [4]. Basis factoring and pivoting are handled with a heavily modified version of the LA05 subroutine library [6], including modifications suggested by Suhl & Suhl [8] and Zlatev [9].

DYLP can be used as a standalone simplex LP code, with only a minimal shell to generate the constraint system. DYLP also provides hooks for column generation and constraint generation, between deactivation and activation for variables and constraints, respectively.

2 Notation

DYLP works naturally with the minimisation problem

$$\begin{aligned} \min\ & cx \\ & Ax \le b \\ & l \le x \le u \end{aligned} \tag{1}$$

Add slack variables s and partition $\begin{bmatrix} A & I \end{bmatrix}$ into basic and nonbasic portions as

$$\begin{bmatrix} B & N \end{bmatrix} = \left[\begin{array}{cc|cc} B^t & 0 & N^t & I^t \\ B^l & I^l & N^l & 0 \end{array}\right] \tag{2}$$

with corresponding partitions $\begin{bmatrix} x^B & s^B & x^N & s^N \end{bmatrix}^T$ for x, s, and $\begin{bmatrix} b^t & b^l \end{bmatrix}^T$ for b. The objective c is augmented with 0's in the columns corresponding to the slack variables, and partitioned as $\begin{bmatrix} c^B & 0 & c^N & 0 \end{bmatrix}$. The basis inverse will be

$$B^{-1} = \begin{bmatrix} (B^t)^{-1} & 0 \\ -B^l(B^t)^{-1} & I^l \end{bmatrix}. \tag{3}$$

We then have

$$\begin{aligned} \begin{bmatrix} x^B \\ s^B \end{bmatrix} &= B^{-1}b - B^{-1}N \begin{bmatrix} x^N \\ s^N \end{bmatrix} \\ &= \begin{bmatrix} (B^t)^{-1}b^t \\ b^l - B^l(B^t)^{-1}b^t \end{bmatrix} - \begin{bmatrix} (B^t)^{-1}N^t & (B^t)^{-1} \\ N^l - B^l(B^t)^{-1}N^t & -B^l(B^t)^{-1} \end{bmatrix} \begin{bmatrix} x^N \\ s^N \end{bmatrix} \end{aligned} \tag{4}$$

and

$$\begin{aligned} z &= \begin{bmatrix} c^B & 0 \end{bmatrix} \begin{bmatrix} x^B & s^B \end{bmatrix}^T + \begin{bmatrix} c^N & 0 \end{bmatrix} \begin{bmatrix} x^N & s^N \end{bmatrix}^T \\ &= \begin{bmatrix} c^B & 0 \end{bmatrix} B^{-1}b + \left(\begin{bmatrix} c^N & 0 \end{bmatrix} - \begin{bmatrix} c^B & 0 \end{bmatrix} B^{-1}N\right) \begin{bmatrix} x^N & s^N \end{bmatrix}^T \\ &= c^B(B^t)^{-1}b^t + \begin{bmatrix} c^N - c^B(B^t)^{-1}N^t & -c^B(B^t)^{-1} \end{bmatrix} \begin{bmatrix} x^N & s^N \end{bmatrix}^T \end{aligned} \tag{5}$$

The quantities $\begin{bmatrix} x^B & s^B \end{bmatrix}^T = \overline{b} = B^{-1}b$ are the values of the basic variables, the quantities $y = \begin{bmatrix} c^B & 0 \end{bmatrix} B^{-1}$ are the dual variables, and the quantities $\overline{c} = \left(\begin{bmatrix} c^N & 0 \end{bmatrix} - \begin{bmatrix} c^B & 0 \end{bmatrix} B^{-1}N\right)$ are the reduced costs. A row or column of $B^{-1}N$ (as appropriate to the context) will be denoted $\overline{a}_k$ (the single subscript distinguishes it from an individual element $\overline{a}_{ij}$). A row or column of B^{-1} (as appropriate to the context) will be denoted β_k. When discussing pivot selection calculations, Δ_j will be the change in nonbasic variable x_j or s_j.

The dual problem is formed by first converting (1) to max $-cx$, giving

$$\begin{aligned} \min\ & yb \\ & yA \ge -c \\ & y \ge 0 \end{aligned} \tag{6}$$

Add surplus variables σ and partition $\begin{bmatrix} A & -I \end{bmatrix}^T$ into basic and nonbasic portions as

$$\begin{bmatrix} \mathcal{B} \\ \mathcal{N} \end{bmatrix} = \left[\begin{array}{cc} 0 & -I^{\mathcal{B}} \\ B^t & N^t \\ \hline -I^{\mathcal{N}} & 0 \\ B^l & N^l \end{array}\right] \tag{7}$$

with corresponding partitions $\begin{bmatrix}\sigma^{\mathcal{B}} & y^{\mathcal{B}} & \sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}$ for y, σ, and $\begin{bmatrix}c^B & c^N\end{bmatrix}$ for c. The right-hand side b is augmented with 0's in the rows corresponding to the surplus variables and partitioned as $\begin{bmatrix}0 & b^t & 0 & b^l\end{bmatrix}^T$. The basis inverse will be

$$\mathcal{B}^{-1} = \begin{bmatrix}(B^t)^{-1}N^t & (B^t)^{-1} \\ -I^{\mathcal{B}} & 0\end{bmatrix}. \tag{8}$$

We then have

$$\begin{aligned}\begin{bmatrix}\sigma^{\mathcal{B}} & y^{\mathcal{B}}\end{bmatrix} &= (-c)\mathcal{B}^{-1} - \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\mathcal{N}\mathcal{B}^{-1} \\ &= \begin{bmatrix}c^N - c^B(B^t)^{-1}N^t & -c^B(B^t)^{-1}\end{bmatrix} - \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\begin{bmatrix}-(B^t)^{-1}N^t & -(B^t)^{-1} \\ B^l(B^t)^{-1}N^t - N^l & B^l(B^t)^{-1}\end{bmatrix}\end{aligned} \tag{9}$$

and

$$\begin{aligned}z &= \begin{bmatrix}\sigma^{\mathcal{B}} & y^{\mathcal{B}}\end{bmatrix}\begin{bmatrix}0 & b^t\end{bmatrix}^T + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\begin{bmatrix}0 & b^l\end{bmatrix}^T \\ &= (-c)\mathcal{B}^{-1}b^{\mathcal{B}} + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}(b^{\mathcal{N}} - \mathcal{N}\mathcal{B}^{-1}b^{\mathcal{B}}) \\ &= -c^B(B^t)^{-1}b^t + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\begin{bmatrix}(B^t)^{-1}b^t \\ b^l - B^l(B^t)^{-1}b^t\end{bmatrix}\end{aligned} \tag{10}$$

When discussing pivot selection calculations, δ_j will be the change in nonbasic dual variable y_j or σ_j.

3 Updating Formulas for Variables

For purposes of the updating formulas, the distinction between original variables x and slack variables s is not important. For simplicity, x_k is used to represent original variables and slack variables in this section. In the same vein, c^B and c^N will denote $\begin{bmatrix}c^B & 0\end{bmatrix}$ and $\begin{bmatrix}c^N & 0\end{bmatrix}$, respectively.

3.1 Primal Variable Updates

Updating the primal variables is straightforward and follows directly from (4).

Both primal and dual pivots calculate the change in the entering primal variable, Δ_j. The entering variable x_j is set to $u_j + \Delta_j$ or $l_j + \Delta_j$, for x_j entering from its upper or lower bound, respectively. The leaving variable x_i is set to u_i or l_i, for x_i leaving at its upper or lower bound, respectively. The remaining basic variables x_k, $k \neq i$, are updated according to the formula

$$x_k = \overline{b}_k - \overline{a}_{kj}\Delta_j. \tag{11}$$

3.2 Dual Variable Updates

Updating the dual variables is simple in the final implementation, but a little work is necessary to derive the updating formula. The difficulty lies in the fact that the dual variables of interest are $y = \begin{bmatrix}y^{\mathcal{B}} & y^{\mathcal{N}}\end{bmatrix}$, *i.e.*, a mixture of basic and nonbasic dual variables. Direct application of (9) is not possible.

Assume that the leaving variable x_i occupies row k in the basis B. Let $e_k \in R^m$ be the row or column vector (as appropriate) with a 1 in position k and 0's in all other positions.

The new basis B' can be expressed as $B' = B - a_i e_k + a_j e_k$. The new vector of basic costs, $(c')^B$, can be expressed as $(c')^B = c^B - [0 \ldots c_i \ldots 0] + [0 \ldots c_j \ldots 0]$, where c_i and c_j occur in the k^{th} position.

Since $B'(B')^{-1} = I = B(B')^{-1} - a_i(\beta')_k + a_j(\beta')_k$, we have $B(B')^{-1} = I + a_i(\beta')_k - a_j(\beta')_k$.

We can proceed to derive the update formulas for y as follows:

$$\begin{aligned}
y' &= (c')^B(B')^{-1} \\
&= c^B B^{-1} B(B')^{-1} - c_i(\beta')_k + c_j(\beta')_k \\
&= y(I + a_i(\beta')_k - a_j(\beta')_k) - c_i(\beta')_k + c_j(\beta')_k \\
&= y + (c_j - y a_j)(\beta')_k - (c_i - y a_i)(\beta')_k.
\end{aligned}$$

Recognising that $\overline{c}_j = c_j - y a_j$ is the reduced cost of x_j before the basis change, and noting that $\overline{c}_i = c_i - y a_i = 0$ since x_i was basic, we have

$$y' = y + \overline{c}_j(\beta')_k. \tag{12}$$

As a further observation, note that $(\beta')_k = \beta_k / \overline{a}_{kj}$, so we can update y using a row of B^{-1} as

$$y' = y + \overline{c}_j \beta_k / \overline{a}_{kj}. \tag{13}$$

4 Pricing Algorithms

4.1 Projected Steepest Edge Pricing

The primal simplex algorithm in DYLP uses projected steepest edge (PSE) pricing; the algorithm used is described as dynamic projected steepest edge ('dynamic') in Forrest and Goldfarb [2].

To understand the operation of projected steepest edge (PSE) pricing, it will be helpful to start with the definition of a direction of motion. The values of the basic and nonbasic variables can be expressed as

$$\begin{bmatrix} x^B \\ x^N \end{bmatrix} = \begin{bmatrix} b \\ l/u \end{bmatrix} - \begin{bmatrix} B^{-1}A^N \\ -I \end{bmatrix} \Delta \tag{14}$$

where l/u is intended to indicate use of the lower or upper bound as appropriate for the particular nonbasic variable. When a given nonbasic variable x_j is moved by an amount Δ_j, the values of x will change as

$$-\begin{bmatrix} B^{-1}a_j \\ -e_j \end{bmatrix} \Delta_j = -\begin{bmatrix} \overline{a}_j \\ -e_j \end{bmatrix} \Delta_j = \eta_j \Delta_j \tag{15}$$

The vector η_j is the direction of motion as x_j is changed; alternatively, it is the edge of the polyhedron which is traversed as x_j is changed. Let $\gamma_j = \|\eta_j\|$ be the norm of η_j.

For pricing, it can be immediately seen that $c\eta_j = c_j - c^B \overline{a}_j$ is the reduced cost $\overline{c}_j$. Dantzig pricing chooses an entering variable x_j such that $\overline{c}_j$ has appropriate sign and the largest magnitude over all reduced costs, but it can be misled by differences in scaling from one column to the next. Steepest edge (SE) pricing scales $\overline{c}_j$ by γ_j, choosing an entering variable x_j with $\overline{c}_j$ of appropriate sign and the largest $\left|\frac{c\eta_j}{\|\eta_j\|}\right|$, effectively calculating the change in objective value over a unit vector in the direction of motion. This gives a uniform pricing comparison, using the slope of the edge.

Projected steepest edge (PSE) pricing uses 'projected' column norms which are calculated using a vector $\tilde{\eta}_j$ which contains only the components of η_j included in a reference frame. Initially, this reference frame contains only the nonbasic variables, so that $\tilde{\gamma}_j = 1$ for all $x_j \in x^N$. In order to avoid calculating $\tilde{\gamma}_j$ from scratch each time a column must be priced, the norms are iteratively updated.

To derive the update formulas for $\tilde{\gamma}_j$, it is useful to start with the update formulas for the full vector η_j. As mentioned in §3.2, for x_i leaving basis position k and x_j entering, $B(B')^{-1} = I + a_i(\beta')_k - a_j(\beta')_k$. Taking this one step further, $(B')^{-1} = B^{-1} + \overline{a}_i(\beta')_k - \overline{a}_j(\beta')_k$. Then for an arbitrary column a_p,

$$\begin{aligned}(B')^{-1}a_p &= B^{-1}a_p + \overline{a}_i(\beta')_k a_p - \overline{a}_j(\beta')_k a_p \\ \overline{a}'_p &= \overline{a}_p + e_k(\frac{\overline{a}_{kp}}{\overline{a}_{kj}}) - \overline{a}_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})\end{aligned} \tag{16}$$

(recalling that $(\beta')_k = \beta_k/\overline{a}_{kj}$).

To see that (16) amounts to $\eta'_p = \eta_p - \eta_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})$, it's helpful to expand the vectors:

$$\overline{a}'_p = \begin{bmatrix}\overline{a}_{1p}\\ \vdots \\ \overline{a}_{kp} \\ \vdots \\ \overline{a}_{mp}\end{bmatrix} + \begin{bmatrix}0\\ \vdots \\ 1 \\ \vdots \\ 0\end{bmatrix}\frac{\overline{a}_{kp}}{\overline{a}_{kj}} - \begin{bmatrix}\overline{a}_{1j}\\ \vdots \\ \overline{a}_{kj} \\ \vdots \\ \overline{a}_{mj}\end{bmatrix}\frac{\overline{a}_{kp}}{\overline{a}_{kj}}. \tag{17}$$

With a little thought, it can be seen that the middle term represents one half of the permutation which moves x_j into the basic partition of η'_j. (The other half moves x_i into the nonbasic partition). When updating η_i, the update formula can be collapsed to $\eta'_i = -\eta_j/\overline{a}_{kj}$, since $\overline{a}_{ki} = 1$. Summarising, the update formulas for the edge directions η_j are

$$\begin{aligned}&\eta'_p = \eta_p - \eta_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}}), \qquad p \neq i \\ &\eta'_i = -\eta_j/\overline{a}_{kj}.\end{aligned} \tag{18}$$

In fact, the code actually stores and updates γ_j^2. With (18) in hand, derivation of the update formulas are straightforward:

$$\begin{aligned}
(\gamma_p')^2 &= \eta_p' \cdot \eta_p' \\
&= (\eta_p - \eta_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})) \cdot (\eta_p - \eta_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})) \\
&= \eta_p \cdot \eta_p - 2(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})\eta_j \cdot \eta_p + (\frac{\overline{a}_{kp}}{\overline{a}_{kj}})^2 \eta_j \cdot \eta_j \\
&= \gamma_p^2 - 2(\frac{\overline{a}_{kp}}{\overline{a}_{kj}}) \begin{bmatrix} \overline{a}_j^T & e_j^T \end{bmatrix} \begin{bmatrix} \overline{a}_p \\ e_p \end{bmatrix} + (\frac{\overline{a}_{kp}}{\overline{a}_{kj}})^2 \gamma_j^2 \\
&= \gamma_p^2 - 2(\frac{\overline{a}_{kp}}{\overline{a}_{kj}})(\overline{a}_j^T B^{-1}) a_p + (\frac{\overline{a}_{kp}}{\overline{a}_{kj}})^2 \gamma_j^2
\end{aligned} \tag{19}$$

$$\begin{aligned}
(\gamma_i')^2 &= \eta_i' \cdot \eta_i' \\
&= \eta_j / \overline{a}_{kj} \cdot \eta_j / \overline{a}_{kj} \\
&= \gamma_j^2 / \overline{a}_{kj}^2
\end{aligned} \tag{20}$$

Equations (18) can be used directly to update the $\tilde{\eta}_j$. To adapt (19) and (20) for the $\tilde{\gamma}_j$, a little algebra should serve to see that it's sufficient to substitute $\tilde{a}_j$ in (19), as well as using $\tilde{\gamma}_p$ and $\tilde{\gamma}_j$.

It is straightforward to observe that when equations (18) are premultiplied by c, they can be used to update the reduced costs as

$$\begin{aligned}
\overline{c}_p' &= \overline{c}_p - \overline{c}_j(\frac{\overline{a}_{kp}}{\overline{a}_{kj}}) \qquad p \neq i \\
\overline{c}_i' &= -\overline{c}_j / \overline{a}_{kj}.
\end{aligned} \tag{21}$$

4.2 Dual Steepest Edge Pricing

The dual simplex in DYLP uses dual steepest edge (DSE) pricing; the algorithm used is described as dual algorithm 1 ('steepest 1') in Forrest and Goldfarb [2].

The values $\overline{b} = B^{-1}b$ are the reduced costs of the nonbasic dual variables. Analogous to Dantzig pricing in the primal case, one can choose a entering dual variable y_i such that $\overline{b}_i$ has appropriate sign and the largest magnitude over all reduced costs, but there is the same problem with scaling. The version of dual steepest edge (DSE) pricing implemented in DYLP scales $\overline{b}_i$ by $\rho_i = \|\beta_i\|$, choosing a leaving variable x_i with $\overline{b}_i$ of appropriate sign and the largest $\left|\frac{\beta_i b}{\|\beta_i\|}\right|$, effectively calculating the change in the dual objective value over a unit vector in the dual direction of motion in the space of the dual variables. This gives a uniform pricing comparison, using the slope of the dual edge.

In the next few paragraphs, an alternative motivation of the algorithm is presented which (perhaps) clarifies the relationship between dual algorithm 1 and dual algorithm 2 in that paper[1].

To see how DSE operates within the context of the revised primal simplex tableau, we can refer

[1]Those who have read [2] are warned that the author's notation is in no way compatible with that of Forrest and Goldfarb.

back to equations 9 and 10 from §2. We have

$$\begin{aligned}\begin{bmatrix}\sigma^{\mathcal{B}} & y^{\mathcal{B}}\end{bmatrix} &= (-c)\mathcal{B}^{-1} - \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix} \mathcal{N}\mathcal{B}^{-1} \\ &= \begin{bmatrix}c^N - c^B(B^t)^{-1}N^t & -c^B(B^t)^{-1}\end{bmatrix} - \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix} \begin{bmatrix}-(B^t)^{-1}N^t & -(B^t)^{-1} \\ B^l(B^t)^{-1}N^t - N^l & B^l(B^t)^{-1}\end{bmatrix}\end{aligned} \tag{9}$$

and

$$\begin{aligned} z &= \begin{bmatrix}\sigma^{\mathcal{B}} & y^{\mathcal{B}}\end{bmatrix}\begin{bmatrix}0 & b^t\end{bmatrix}^T + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\begin{bmatrix}0 & b^l\end{bmatrix}^T \\ &= (-c)\mathcal{B}^{-1}b^{\mathcal{B}} + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}(b^{\mathcal{N}} - \mathcal{N}\mathcal{B}^{-1}b^{\mathcal{B}}) \\ &= -c^B(B^t)^{-1}b^t + \begin{bmatrix}\sigma^{\mathcal{N}} & y^{\mathcal{N}}\end{bmatrix}\begin{bmatrix}(B^t)^{-1}b^t \\ b^l - B^l(B^t)^{-1}b^t\end{bmatrix}\end{aligned} \tag{10}$$

Recall that the values of the dual basic variables are the reduced costs of the primal problem, and the reduced costs of the dual variables are the values of the primal basic variables (*cf.* equations 4 and 5).

By analogy to the primal pivoting rules, for dual simplex we want to choose a nonbasic dual variable which will move us in a direction of steepest descent. If the nonbasic dual is to increase, its reduced cost must be less than 0 in order to see a reduction in the dual objective. This corresponds to the case of a primal variable which will be increased and driven out of the basis at its lower bound with a positive primal reduced cost. If the nonbasic dual is to decrease, its reduced cost must be greater than 0 in order to see a reduction in the dual objective. This corresponds to the case of a primal variable which will be decreased and driven out of the basis at its upper bound with a negative primal reduced cost.

The actual direction of motion in the full dual space (y and σ) would be specified by a row of

$$\mathcal{N}\mathcal{B}^{-1} = \begin{bmatrix}-(B^t)^{-1}N^t & -(B^t)^{-1} \\ B^l(B^t)^{-1}N^t - N^l & B^l(B^t)^{-1}\end{bmatrix},$$

a vector which is not readily available in the revised primal simplex. (Moreover, for the typical problem in which the number of variables greatly exceeds the number of constraints, the norm of this vector is expensive to calculate when initialising the pricing algorithm, and the updates are expensive. This is the algorithm which Forrest and Goldfarb describe as dual algorithm 2.)

However, one can make an argument that there's no need to consider the component of the direction of motion in the subspace of the dual surplus variables. (More positively, we can take the view that we're only interested in motion in the polyhedron $\{y \in R^m \mid yA \geq -c, y \geq 0\}$ defined by the dual variables.) Changes in the surplus variables cannot affect the objective directly, as they account for the 0's in the augmented and partitioned b vector. Algebraically, we can see that the dual basic portion of b, $\begin{bmatrix}0 & b^t\end{bmatrix}^T$, guarantees that there will never be any contribution from the columns of $\mathcal{N}\mathcal{B}^{-1}$ involving N. The component of motion in the space of the dual variables y is then simply the rows β_l of B^{-1}, which are easily available from the primal tableau. (The analogous action in the primal problem — ignore the component of η_j in the subspace of the primal slack variables — offers no computational advantage.)

Given a rationale for taking the rows β_l of B^{-1} as the edges of interest, what remains is to work out the details. Since we're aiming for a steepest edge algorithm, we'll be interested in iteratively updating $\|\beta_l\|^2 = \beta_l \cdot \beta_l$, the square of the norm of a row β_l. We can begin by deriving an update formula for the basis inverse, and from there restrict ourselves to a single row and its norm. Let x_i be the leaving variable and x_j be the entering variable, and assume x_i occupies row k of the basis

B before the update. The new basis B' can be expressed as $B' = B - a_i e_k + a_j e_k$. Premultiplying by B^{-1} and postmultiplying by $(B')^{-1}$, we have

$$\begin{aligned} B^{-1}B'(B')^{-1} &= B^{-1}B(B')^{-1} - B^{-1}a_i e_k(B')^{-1} + B^{-1}a_j e_k(B')^{-1} \\ B^{-1} &= (B')^{-1} - \overline{a}_i\beta'_k + \overline{a}_j\beta'_k \\ (B')^{-1} &= B^{-1} + \overline{a}_i\beta'_k - \overline{a}_j\beta'_k \end{aligned}$$

Since x_i was basic, $\overline{a}_i = e_k$. This gives

$$(B')^{-1} = B^{-1} + e_k\beta'_k - \overline{a}_j\beta'_k.$$

Premultiplying by e_l to obtain an update formula for row l, and recalling that $\beta'_k = \beta_k/\overline{a}_{kj}$, we have

$$\begin{aligned} \beta'_l &= \beta_l - \frac{\overline{a}_{lj}}{\overline{a}_{kj}}\beta_k \qquad l \neq k \\ \beta'_k &= \frac{1}{\overline{a}_{kj}}\beta_k \end{aligned} \tag{22}$$

The development of the update formulæ for $\rho_l = \|\beta_l\|^2$ is straightforward algebra; we have

$$\begin{aligned} \rho'_l &= \rho_l - 2\frac{\overline{a}_{lj}}{\overline{a}_{kj}}\beta_l \cdot \beta_k + (\frac{\overline{a}_{lj}}{\overline{a}_{kj}})^2\rho_k \qquad l \neq k \\ \rho'_k &= (\frac{1}{\overline{a}_{kj}})^2\rho_k \end{aligned} \tag{23}$$

Since the update will be performed for all rows in the basis, it's worth calculating the vector $\tau = B^{-1}\beta_k^T$ to obtain all the inner products $\beta_l \cdot \beta_k$ in one calculation.

5 Lightweight Anti-Degeneracy Measures Based on Hyperplane Alignment

Each constraint $a_k x \leq b_k$ defines an associated hyperplane at equality. In the absence of degeneracy, a simplex pivot consists moving away from one hyperplane along an edge until another hyperplane blocks further progress. The constraint which is becoming loose (the hyperplane being left) is said to be deactivated, and the constraint becoming tight (the hyperplane being approached) is said to be activated. The choice of entering variable x_j determines the constraint to be deactivated, and the choice of leaving variable x_i determines the constraint which will be activated.

Ideally, the choice of constraints is unique, but life is seldom ideal. Often the lack of uniqueness is due to degeneracy, in which one or more basic variables have the value 0. In this case the change of basis that occurs with the pivot will still result in the activation of one constraint and the deactivation of another, but it is not possible to move to a new extreme point.

This section describes a suite of measures based on hyperplane alignment which can be used to guide the selection of the leaving variable, and thus the hyperplane to be activated, in the presence of degeneracy.

5.1 Activation of Constraints

In both the primal and dual simplex algorithms, the constraint which is activated by a pivot depends on the leaving variable and its direction of motion. Before discussing the types of alignment calculations, it will be useful to discuss the activation of constraints. Knowing the type of constraint ('≤' or '≥') is necessary because it determines the direction of the normal with respect to the feasible region.

DYLP assumes that the majority of explicit constraints of the primal problem are of the form $a_k x \leq b_k$. It also understands range constraints of the form $\check{b}_k \leq a_k x \leq b_k$. These are implemented by placing an upper bound on the associated slack variable s_k, but for purposes of determining the constraint to be activated we need to recognise that there are really two constraints, $a_k x \geq \check{b}_k$ and $a_k x \leq b_k$.

Bounded variables are handled implicitly by the primal simplex algorithm. When a bounded variable becomes nonbasic at its lower bound, the constraint $x_k \geq l_k$ is activated; when it becomes nonbasic at its upper bound, the constraint $x_k \leq u_k$ is activated.

A final complication is introduced in phase I of the primal simplex, where it's possible to approach a constraint from the 'wrong' side in the process of finding a primal feasible basic solution. For example, if a slack variable $s_k < 0$ will increase and leave the basis at 0, the constraint which is becoming tight is actually $a_k x \geq b_k$.

Turning to the dual problem, the question of what constraint is being activated is substantially obscured by the mechanics of running the dual simplex algorithm from the primal data structures. A much clearer picture can be obtained by expanding the primal system to include explicit upper and lower bound constraints and examining the resulting dual constraints ([2, §3.4], or see [3] for an extended development). Briefly, let y be the dual variables associated with the original explicit constraints $a_k x \leq b_k$ (the architectural constraints), $\check{y}$ be the dual variables associated with the lower bound constraints, and $\hat{y}$ be the dual variables associated with the upper bound constraints. A superscript $\underline{N}$ will represent the set of primal variables at their lower bound, $\overline{N}$ the set of primal variables at their upper bound, and B the set of basic primal variables. The set of dual constraints can then be written as

$$
\begin{aligned}
yB - \check{y}^B I + \hat{y}^B I &= c^B \\
y\underline{N} - \check{y}^{\underline{N}} I + \hat{y}^{\underline{N}} I &= c^{\underline{N}} \\
y\overline{N} - \check{y}^{\overline{N}} I + \hat{y}^{\overline{N}} I &= c^{\overline{N}}
\end{aligned}
$$

where the first term in each dual constraint comes from the primal architectural constraints, the second term from the lower bound constraints, and the third term from the upper bound constraints. The variables $\check{y}^B$, $\hat{y}^B$, $\hat{y}^{\underline{N}}$, and $\check{y}^{\overline{N}}$ are dual nonbasic and therefore have the value zero. (They are associated with primal bound constraints which are not tight.) We can rewrite the dual constraints as

$$
\begin{aligned}
yB &= c^B \\
y\underline{N} - \check{y}^{\underline{N}} I &= c^{\underline{N}} \\
y\overline{N} + \hat{y}^{\overline{N}} I &= c^{\overline{N}}
\end{aligned}
\tag{24}
$$

We can then interpret the constraints $y\underline{N} - \check{y}^{\underline{N}} I = c^{\underline{N}}$ as $y\underline{N} \geq c^{\underline{N}}$, with $\check{y}^{\underline{N}}$ acting as the surplus variables. Similarly, the constraints $y\overline{N} + \hat{y}^{\overline{N}} I = c^{\overline{N}}$ can be interpreted as $y\overline{N} \leq c^{\overline{N}}$, with $\hat{y}^{\overline{N}}$ acting as the slack variables.

With this interpretation in hand, it's easy to determine the hyperplane that's activated by a pivot. When a dual variable $\check{y}^{\underline{N}}_k$ is driven out of the basis at 0 (x_k enters rising from its lower bound), the constraint $ya_k \geq c_k$ becomes tight. When a dual variable $\hat{y}_k^{\bar{N}}$ is driven out of the basis at 0 (x_k enters decreasing from its upper bound), the constraint $ya_k \leq c_k$ becomes tight. This interpretation is uniform for the original primal variables as well as the primal slack variables.

For the most common case of a primal constraint $a_i x \leq b_i$, with associated slack s_i, $0 \leq s_i \leq \infty$, the dual constraint reduces to $y_i \geq 0$, and this is handled as an implicit bound by the dual simplex algorithm implemented in DYLP. (Range constraints complicate the interpretation, but not the mechanics, of the implementation. Again, see [3] for a detailed explanation.)

In the sections which follow, the alignment calculations are developed in terms of the most common constraint form ($a_k x \leq b_k$ in the case of the primal simplex, and $ya_k \geq c_k$ in the case of the dual simplex). Accommodating the different constraint types described in this section is simply a matter of correcting the sign of the calculation as needed to account for the direction of the constraint normal.

5.2 Alignment With Respect to the Objective Function

The primal objective used in DYLP is $\min cx$. We need to move in the direction $-c$ until we reach an extreme point of the polytope where the cone formed by the normals of the active constraints includes $-c$.

If the goal is to travel in the direction $-c$, one approach would be to leave each vertex by moving along the edge which most nearly points in the direction $-c$. The edges traversed by the simplex algorithm are simply the intersections of active hyperplanes. If we're trying to construct an edge with which we can leave a degenerate vertex, we could choose to activate a hyperplane $a_k x = b_k$ such that $-c$ most nearly lies in the hyperplane, on the theory that its intersection with other active hyperplanes at the vertex is more likely to produce an edge with the desired orientation. This is the 'Aligned' strategy, because we want the hyperplanes most closely aligned with the normal of the objective.

Going to the other extreme, at the optimal vertex it must be true that the active hyperplanes block further motion in the direction $-c$, and $-c$ must lie within the cone of normals of the active hyperplanes. One can make the argument that a good choice of hyperplane would the one that most nearly blocks motion in the direction $-c$, as it's likely to be active at the optimal vertex. This is called the 'Perpendicular' strategy, because we want the hyperplanes which are most nearly perpendicular to the normal of the objective.

For constraints $a_k x \leq b_k$ the normal points out of the feasible region. Let the alignment of the normal a_k with $-c$ be calculated as $\dfrac{a_i \cdot c}{\|a_i\|\|c\|}$. Then for the Perpendicular strategy, we want to select the hyperplane $a_i x = b_i$ such that $i = \arg\max\limits_k \dfrac{a_k \cdot c}{\|a_k\|}$ over all constraints $a_k x \leq b_k$ in the degenerate set.

For the Aligned strategy, the criteria is a bit more subtle. If $a_k \cdot -c = 0$, $-c$ lies in the hyperplane $a_k x = b_k$. Selecting the hyperplane i such that $i = \arg\min\limits_k \left|\dfrac{a_k \cdot c}{\|a_k\|}\right|$ is not quite sufficient. Where possible, DYLP attempts to choose hyperplanes which are tilted in the direction of the objective, so as to bound the problem. The preferred hyperplane is $a_i x = b_i$ such that $i = \arg\min\limits_{\{k \mid a_k \cdot c \geq 0\}} \dfrac{a_k \cdot c}{\|a_k\|}$ over the constraints in the degenerate set. If $a_k \cdot c < 0$ for all k, the preferred hyperplane is chosen as $i = \arg\max\limits_k \dfrac{a_k \cdot c}{\|a_k\|}$.

The dual objective used in DYLP is $\min yb$, but we must be careful here to to include the effect of the bounds on the primal variables. The objective is properly stated as $\min \begin{bmatrix} y & \check{y} & \hat{y} \end{bmatrix} \begin{bmatrix} b & -l & u \end{bmatrix}^T$, and we will need to include the coefficients of $\check{y}$ and $\hat{y}$ in the constraint normals. (In the primal we could ignore this effect, because the objective coefficients associated with the slack variables are uniformly zero.)

For dual constraints $ya_k \geq c_k$, the normal $\begin{bmatrix} a_k & -e_k & 0 \end{bmatrix}$ will point into the feasible region and DYLP calculates the alignment of $\begin{bmatrix} -b & l & -u \end{bmatrix}$ with the hyperplane as $\dfrac{b \cdot a_k + l_k}{(\|a_k\| + 1)\|\begin{bmatrix} b & -l & u \end{bmatrix}\|}$, so that a positive result identifies a constraint which blocks motion in the direction of the objective. For a constraint $ya_k \leq c_k$, the calculation is $\dfrac{(-b) \cdot a_k - u_k}{(\|a_k\| + 1)\|\begin{bmatrix} b & -l & u \end{bmatrix}\|}$. Selection of a specific leaving variable $\check{y}_k^N$ or $\hat{y}_k^{\overline{N}}$ is done using the same criteria outlined for the Perpendicular and Aligned cases in the primal problem.

5.3 Alignment With Respect to the Direction of Motion

The selection of an entering variable specifies the desired direction of motion for the pivot. At a degenerate vertex, we cannot move in the desired direction because the set of active hyperplanes does not contain this edge. Intuitively, activating a hyperplane which is closely aligned with the desired direction of motion might increase the chance of being able to move in that direction.

For the primal simplex, the direction of motion derived in §4.1 is $\eta_j = \begin{bmatrix} -B^{-1}a_j & -e_j \end{bmatrix}^T$. The normal of a constraint $a_k x \leq b_k$ points out of the feasible region. The alignment of η_j and the normal a_k is calculated as $\dfrac{a_k \cdot \eta_j}{\|a_k\|\|\eta_j\|}$, so that a positive value indentifies a hyperplane which blocks motion in the direction η_j.

It's important to note that normal a_k in this calculation is that of the inequality — the coefficient associated with the slack s_k is *not* included. This means that $a_k \cdot \eta_j \equiv -\overline{a}_{kj}$. For a bound constraint, the relation is obvious by inspection. If, for example, the constraint is $x_k \leq u_k$, the normal is e_k, and $e_k \cdot -\overline{a}_j = -\overline{a}_{kj}$. For an architectural constraint, it's necessary to look at the calculation in a way that separates the contributions of the architectural and slack variables, and basic and nonbasic variables. We are interested in the structure of the product $\begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$ for loose constraints which will be activated by pivoting the associated slack variable out of the basis. Breaking up the matrices as detailed in §2, we have

$$\begin{aligned}
\begin{bmatrix} B^l & I & N^l & 0 \end{bmatrix} \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} &= \begin{bmatrix} B^l & I & N^l & 0 \end{bmatrix} \begin{bmatrix} -\begin{bmatrix} (B^t)^{-1} & 0 \\ -B^l(B^t)^{-1} & I \end{bmatrix} \begin{bmatrix} N^t \\ N^l \end{bmatrix} \\ \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} B^l & I & N^l & 0 \end{bmatrix} \begin{bmatrix} -(B^t)^{-1}N^t & -(B^t)^{-1} \\ B^l(B^t)^{-1}N^t - N^l & B^l(B^t)^{-1} \\ I & 0 \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} -B^l(B^t)^{-1}N^t + B^l(B^t)^{-1}N^t - N^l + N^l & -B^l(B^t)^{-1} + B^l(B^t)^{-1} \end{bmatrix}
\end{aligned}$$

Removing the contribution due to the basic slack variables, we have $\begin{bmatrix} -B^l(B^t)^{-1}N^t + N^l & -B^l(B^t)^{-1} \end{bmatrix}$. Because the leaving variable for the pivot is a slack, the pivot element $\overline{a}_{kj}$ will be drawn from the component $\begin{bmatrix} B^l(B^t)^{-1}N^t - N^l & B^l(B^t)^{-1} \end{bmatrix}$ in $-B^{-1}N$, and the equivalence is verified.

To finish the alignment calculation for the purposes of selecting a leaving variable, all that is needed is to perform the normalisation by $\|a_k\|\|\eta_j\|$, and since $\|\eta_j\|$ is constant during the selection of the leaving variable, we need only divide by $\|a_k\|$ for comparison purposes. The selection of a leaving variable using the Aligned strategy is as outlined in the previous section.

Given that $a_k \cdot \eta_j \equiv -\overline{a}_{kj}$, it's worth taking a moment to consider a common tie-breaking rule for selecting the leaving variable — pick the variable with the largest $|\overline{a}_{kj}|$, to maintain numerical stability. In fact, this amounts to selecting a hyperplane to activate using an unnormalised variation of the Perpendicular strategy. The obvious corollary is that using the Aligned strategy presents a potential danger to numerical stability by deliberately choosing small pivots.

For the dual simplex, the direction of motion ζ_i is more complicated. Fortunately, we need only consider the portion of ζ_i in the space of the dual variables y. As derived in §4.2, this is simply row β_i of B^{-1}. For the dual constraints $ya_k \geq c_k$, the normal points into the feasible region. To maintain the convention that the alignment calculation should produce a positive result if the constraint blocks motion, the alignment calculation used by DYLP is $-\dfrac{\zeta_i \cdot a_k}{\|\zeta_i\|\|a_k\|}$. Given that we're only interested in the portion of $\zeta_i \cdot a_k$ contributed by the dual variables y, it's immediately apparent that the alignment calculation can be reduced to $-\dfrac{\overline{a}_{ik}}{\|a_k\|}$ for purposes of selecting the leaving dual variable. The final selection of a leaving dual variable using the Aligned or Perpendicular strategy proceeds as outlined in the previous section.

6 The LP Basis

As mentioned in the introduction, a heavily modified version of the LA05 subroutine library is used to factor and pivot the LP basis. An LU factorisation is maintained as U and $L^{-1} = H_1 H_2 \ldots H_p$. The initial set of eta matrices H_k is derived when B is factored, and an additional H matrix is added with each pivot. LA05 provides three basic services:

- Initial factoring of the basis into U and L^{-1} (yla05a).
- Change of basis corresponding to a pivot with x_i leaving the basis and x_j entering the basis, $i \neq j$ (yla05c).
- Calculation of $\overline{a}_j = B^{-1} a_j$ and $y = c^B B^{-1}$ (yla05b).

When factoring the basis, LA05 attempts to strike a balance between maintaining sparsity and numeric stability by using a sliding threshold for pivot acceptance. Initially, partial threshold pivot selection (PTPS) is used, with a relatively low threshold. If numerical problems are detected, the threshold is gradually increased, and full threshold pivot selection (FTPS) is initiated.

For individual pivots, the simplex rules for selecting the entering and leaving variable are the primary criteria for pivot selection, but acceptance of the pivot is determined by the same pivot threshold used during factoring, with the exception that the user can adjust the threshold through an independent multiplier.

Knowledge of the structure and operation of the LA05 subroutines is confined to a set of interface subroutines in the file dy_basis.c. The majority of these are straightforward interface functions whose sole purpose is to hide the LA05 structures and to mediate between LA05 and the remainder of the code.

6.1 Pivoting

To be judged numerically stable, a prospective pivot coefficient $\overline{a}_{ij}$ must exceed the product of the LA05 stability multiplier (la05d_.stable), the DYLP pivot selection multiplier (dy_tols.pivot), and the maximum element in the transformed column $\overline{a}_j = B^{-1}a_j$ (primal simplex) or row $\overline{a}_i = \beta_i N$ (dual simplex). Standard defaults are 10^{-1} for the LA05 stability multiplier and 10^{-4} for the DYLP pivot selection multiplier, so that the pivot coefficient is required to satisfy $\overline{a}_{ij} > 10^{-5} \max_k \overline{a}_{kj}$ (primal simplex) or $\overline{a}_{ij} > 10^{-5} \max_k \overline{a}_{ik}$ (dual simplex). The routine dy_chkpiv is supplied to perform this test, and is used as a qualification test by the routines which select the leaving primal variable in primal simplex and the entering primal variable in dual simplex. If $\overline{a}_{ij}$ is rejected as numerically unstable, the pivot attempt is aborted and the entering variable x_j is entered on a rejected pivot list which prevents further consideration until the basis is refactored following one or more successful pivots using some other entering variable.

If the primal or dual pivoting rules dictate the selection of a pivot coefficient $\overline{a}_{ij} <$ dy_tols.zero $*$ dy_tols.bogus, DYLP will abort the pivot attempt and request that the basis be refactored. Immediately following a refactor, $\overline{a}_{ij}$ will be accepted as a pivot coefficient as long as it exceeds dy_tols.zero and satisfies the numerical stability test.

In cases where the primal or dual simplex is unable to select an entering or leaving variable, respectively, but there are variables on the pivot reject list, the pivot selection multiplier is reduced by a factor of 10 and the scan is repeated after refactoring and clearing the pivot reject list. This relaxation is implemented at the level of the outer (preoptimality) pivoting loops in the primal and dual simplex. Reduction of the pivot selection multiplier continues until it reaches the zero tolerance, at which point the simplex routines will return optimal, infeasible, or unbounded, as appropriate for the currrent solution.

The net effect of this mechanism is that DYLP will, with great reluctance and considerable fuss, consider pivoting on coefficients $\overline{a}_{ij}$ which are less than wonderful from the viewpoint of numerical stability but which appear necessary to make progress toward optimality.

yla05c can still fail if it runs out of space for coefficients or if the pivot results in a singular basis. The implementation of LA05 requires that the basis be reloaded and factored to recover from the latter two errors; this is orchestrated by dy_duenna.

Recovery from rejection of an unsuitable pivot is simpler (the relevant variable is flagged as ineligible) and is again handled by dy_duenna. Once a variable has been flagged, it remains ineligible until the basis is next factored, at which time all flagged variables are again made eligible for pivoting.

Note that yla05c expects to be supplied with $L^{-1}a_j$ as a hidden parameter[2]. If yla05b has been used for an intervening calculation (ftran (premultiplication by B^{-1}) or btran (postmultiplication by B^{-1})), yla05_ftran must be called specifically for this purpose, prior to attempting the pivot.

6.2 Factoring

For factoring the basis, the routine dy_factor provides significant error recovery functions on top of the basic abilities of LA05. The call structure is shown in Figure 1.

A singular basis can occur because of a simplex pivot attempt or as the result of a change in

[2]The expectation of the writers seems to have been that this would be available as a side effect of calculating $B^{-1}a_j$ when pricing the entering column. This is the case for simple Dantzig pricing, but no longer holds for the steepest edge pricing algorithms used in DYLP.

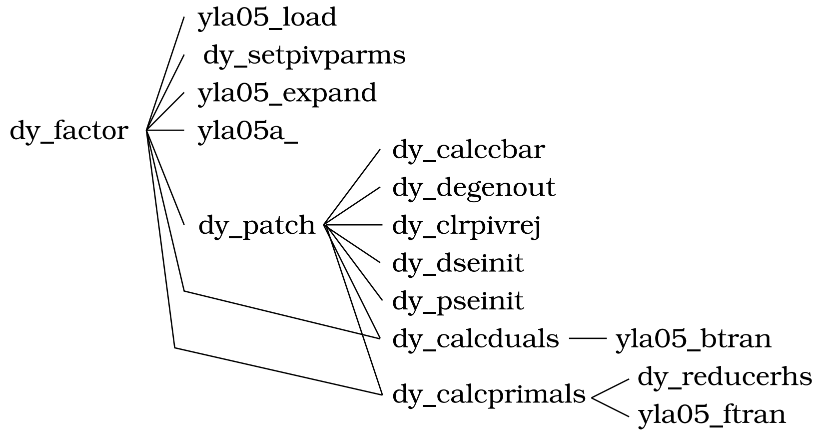

Figure 1: Call Graph for dy_factor

the coefficients of the basis because the client has fixed variables and then requested a warm or hot start. yla05a can repair a singular basis, using yla05s to substitute columns associated with slack variables for the set of columns identified as singular. In the larger context of DYLP, this is the least of the work. dy_factor will call dy_patch to adjust the DYLP data structures as necessary to reflect the exchange of variables between the basic and nonbasic partitions. Depending on the phase, this can include updating the structures which maintain the basis, recalculating the primal (dy_calcprimals) and dual (dy_calcduals) variables, recalculating the reduced costs (dy_calccbar), resetting the DSE or PSE norms (dy_dseinit and dy_pseinit, respectively), clearing the list of variables marked ineligible for pivoting (dy_clrpivrej), and backing out a perturbed subproblem (dy_degenout).

yla05a will abort an attempt to factor the basis if the current pivot selection parameters give rise to numerical instability (detected as excessive growth in the magnitude of the coefficients of the factored basis). dy_pivot will make repeated tries to factor the basis, tightening the pivot selection parameters before each attempt. It will admit failure only if the numerical instability remains when each pivot is chosen as the maximum coefficient remaining in the unpivoted portion of the basis. Tightening of the pivot selection parameters is done according to a schedule kept in a static data structure dy_basis.c:pivtols. (This schedule is also followed when the tolerances are loosened following a successful accuracy check, as described in §7.)

yla05a can also fail to factor the basis if there is insufficient space to store the coefficients. While DYLP uses a generous initial allocation of space for the basis factors, computational experience shows that it can be insufficient if pivot selection is tightened to the point where it's dominated by consideration of numerical stability. dy_factor attempts to recover by calling yla05_expand and yla05_load to expand and reload the LA05 data structures, increasing the space by 50% each time. It gives up only when it cannot allocate more space.

7 Accuracy Checks and Maintenance

Primal and dual accuracy checks, primal and dual feasibility checks, and factoring of the basis can be requested through the routine dy_accchk; each action can be requested separately. The call structure is shown in Figure 2.

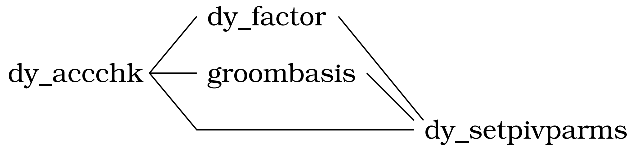

Figure 2: Call Graph for dy_accchk

DYLP refactors the basis and performs accuracy checks at regular intervals, based on a count of pivots which actually change the basis. By default, primal and dual accuracy checks are performed at twice this frequency. During phase II of the primal and dual simplex algorithms, the appropriate feasibility check is performed following each accuracy check. dy_duenna tracks the pivot count and requests checks and factoring at the scheduled intervals.

dy_accchk uses dy_factor to factor the basis and recalculate the primal and dual variables. When the basis has been factored and has passed the accuracy checks, the routine groombasis checks that the status of the basic variables matches their values and makes any necessary adjustments.

Failure of an accuracy check will cause the basis to be refactored. Failure of an accuracy check immediately after refactoring will cause the current pivot selection tolerances to be tightened by one increment before another attempt is made. dy_accchk will repeat this cycle until the accuracy checks are satisfied or there's no more room to tighten the pivot selection parameters. On the other hand, each time that an accuracy check is passed without refactoring the basis, the current pivot selection tolerances are loosened by one increment, to a floor given by the minimum pivot selection tolerance.

The minimum pivot selection tolerance is reset to the loosest possible setting at the start of each simplex phase. If groombasis detects and corrects major status errors (indicating that an unacceptable amount of inaccuracy accumulated since the basis was last factored), it will raise the minimum pivot selection tolerance. This provides long-term control over the pivot selection tolerance and works to minimise accumulated inaccuracy.

The primal accuracy check is $Bx^B = b - Nx^N$. Comparisons are made against the scaled tolerance $\|b\|_1$dy_tols.pchk. To pass the primal accuracy check, it must be that

$$\|(b - Nx^N) - Bx^B\|_1 \leq \|b\|_1 \text{dy\_tols.pchk}$$

The dual accuracy check is $yB = c^B$. Comparisons are made against the scaled tolerance $\|c\|_1$dy_tols.dchk. To pass the dual accuracy check, it must be that

$$\|c^B - yB\|_1 \leq \|c\|_1 \text{dy\_tols.dchk}$$

The primal feasibility check is $l \leq x \leq u$. For each variable, it must be true that $x_j \geq l_j -$ dy_tols.pfeas$(1 + |l_j|)$ and $x_j \leq u_j +$ dy_tols.pfeas$(1 + |u_j|)$. In the implementation, only the basic variables are actually tested; nonbasic variables are assumed to be within bound as an invariant property of the simplex algorithm. dy_tols.pfeas is scaled from dy_tols.zero as

$$\text{dy\_tols.pfeas} = \min(1, \log\left(\frac{\|x_B\|_1}{\sqrt{m}}\right))(\text{dy\_tols.zero}).$$

The dual feasibility check is $\overline{c} = c^N - yN$ of appropriate sign. For each variable, it must be true that $\overline{c}_j \leq$ dy_tols.dfeas for x_j nonbasic at u_j and $\overline{c}_j \geq -$dy_tols.dfeas for x_j nonbasic at l_j. dy_tols.dfeas is scaled from dy_tols.cost as

$$\text{dy\_tols.dfeas} = \min(1, \log\left(\frac{\|y_k\|_1}{\sqrt{m}}\right))(\text{dy\_tols.cost}).$$

8 Dynamic Simplex

Figure 3 gives the overall flow of the dynamic simplex algorithm implemented in DYLP. The actual implementation is written as a finite state machine, with 8 states: primal simplex (dy_primal), dual simplex (dy_dual), deactivate variables (dy_purgevars), generate variables (user supplied), activate variables (dy_addvars, dy_dualaddvars), deactivate constraints (dy_purgecons), generate constraints (user supplied), and activate constraints (dy_addcons, dy_addbndcons). State transitions are controlled by previous state, type of simplex in use, and the outcome of actions in a state.

DYLP provides a cold, warm, and hot start capability. For a cold start, DYLP selects a set of constraints and variables to be the initial active constraint system (dy_coldstart, and see §11.1) and then crashes a basis (dy_crash). For a warm start, DYLP expects that the caller will supply a basis but assumes that the active constraint system and other data structures need to be built to this specification (dy_warmstart). For a hot start, DYLP assumes that its internal data structures are valid except for possible modifications to variable bounds, objective coefficients, or right-hand-side coefficients. It will incorporate these modifications (dy_hotstart) and continue with simplex iterations. DYLP will default to attempting a hot start unless specifically requested to perform a warm or cold start. For all three start types, DYLP will check the constraint system for primal and dual feasibility, choosing primal simplex unless the constraint system is already dual feasible.

When an optimal solution to the active constraint system has been obtained by primal or dual simplex, DYLP will attempt to deactivate variables with unfavourable reduced costs, then activate variables with favourable reduced costs. dy_purgevars scans the active variables for variables with unfavourable reduced costs and deactivates any whose reduced cost exceeds a user-specified threshold. dy_addvars scans the inactive variables looking for variables with favourable reduced costs and activates any that it finds. If variables are activated, phase II of the primal simplex is used to reoptimise. This cycle repeats until no variables can be activated; at this point the solution is optimal for the active constraints, over all variables.

When no variables can be activated, DYLP deactivates any loose constraints (dy_purgecons) and activates any violated constraints (dy_addcons). To avoid cycling (repeatedly deactivating and then activating the same constraint), constraint deactivation is skipped unless there has been an improvement in the objective function over the previous constraint deactivation phase. This guarantees that the simplex will not return to the previous extreme point.

If no inactive constraints are violated, the solution is optimal over all variables and satisfies all constraints and is therefore optimal for the original constraint system.

If violated constraints are found, they are activated (*vid.* §11.2) and dual simplex (dy_dual) is used to reoptimise. Dual feasibility is retained because no new variables are activated — the system is increased only by the coefficients of already active variables in the violated constraints. Once optimality is regained, DYLP returns to attempting to add variables, using primal simplex to reoptimise. Use of the dual simplex can be forbidden by the user; in this case, DYLP substitutes phase I of the primal simplex whenever it would normally call the dual simplex.

In addition to optimality, it is possible for the primal simplex to find that the active constraint system is infeasible or unbounded, or to return any of a variety of errors. The dual simplex may

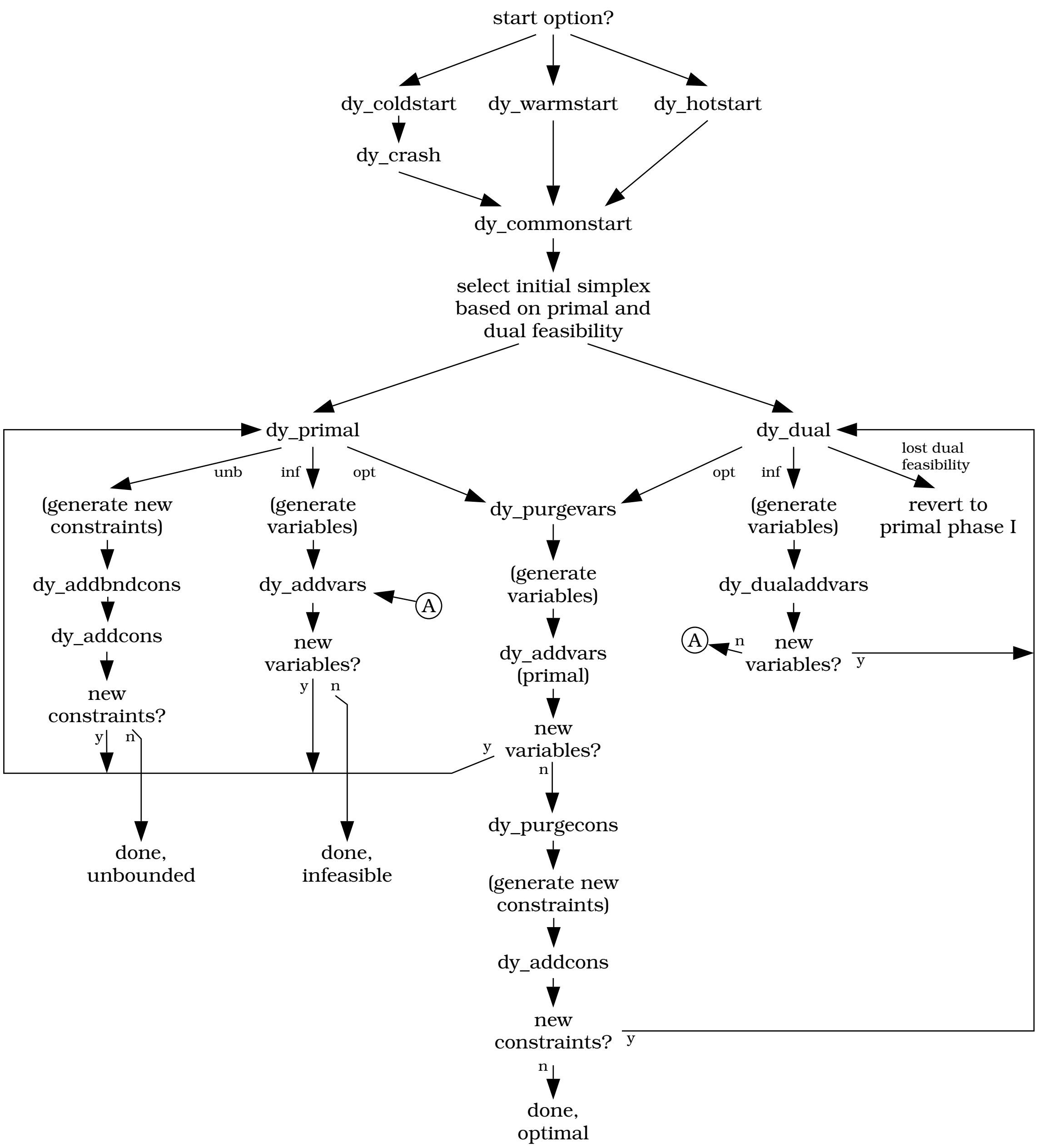

Figure 3: Dynamic Simplex Algorithm Flow

find a problem infeasible, or return an error indication, but unboundedness is not possible (an unbounded problem is not dual feasible).

If either the primal or dual simplex finds that the active system is infeasible, DYLP will attempt to activate variables and resume the simplex algorithm. For the primal simplex, activating variables by adding them to the nonbasic partition is guaranteed not to affect primal feasibility. If all variables with favourable reduced costs have been activated and the active system remains infeasible, the original constraint system is infeasible. For the dual simplex, it is possible that the variables needed to attain primal feasibility cannot be added to the nonbasic partition while retaining dual feasibility. DYLP copes with this situation by reverting to primal phase I, making a second attempt to add variables before calling dy_primal.

If the primal simplex finds that the active system is unbounded, it will attempt to add constraints which might bound the system (dy_addbndcons) as well as any that happen to be violated. The criteria for selection used by dy_addbndcons is that the normal for the constraint form an angle of less than 90° with the objective function (recall that DYLP solves a minimisation problem with ≤ constraints). If no constraints are violated and no constraints can be identified which might bound the objective, the original constraint system is unbounded.

Virtually all error recovery actions taken by DYLP are taken inside of the simplex algorithms, dy_primal and dy_dual. The exception is when the dual simplex algorithm loses dual feasibility (typically due to numerical error). In this case, DYLP will revert to phase I of the primal simplex and make another attempt to reach an optimal solution for the active system.

It is the responsibility of the user to supply appropriate routines for generating new variables (*e.g.*, column generation) and constraints (*e.g.*, cutting planes).

9 Dual Simplex

The primary role of the dual simplex in DYLP is reoptimisation following the addition of violated constraints, and the implementation reflects this secondary status. The dual simplex provides a phase II and lightweight anti-degeneracy using hyperplane alignment. There is no dual phase I, nor does the dual simplex provide a perturbation-based anti-degeneracy algorithm. A number of exceptional conditions will cause DYLP to abandon the dual simplex implementation in favour of the primal simplex implementation.

In dynamic simplex, apparent primal infeasibility can result because only a subset of the variables are present in the active constraint system. In some cases, the variables needed to regain feasibility cannot be activated into the nonbasic partition while maintaining dual feasibility. Put in the context of dual simplex, the problem is unbounded and any dual constraint which would bound the problem would also make the current solution dual infeasible. DYLP implements a variable activation procedure which can pivot a variable into the basis as it is activated, but it is still possible to encounter a situation where multiple pivots are required to regain dual feasibility. When this occurs, DYLP reverts to primal simplex.

If primal infeasible variables remain but they cannot be pivoted because their pivot coefficients do not satisfy the pivot selection rules, dy_dual will punt and DYLP will return to phase I of the primal simplex algorithm in the hope that addition of variables and/or the application of primal pivoting rules will allow pivoting to continue to optimality. In addition, if the dual simplex terminates due to stalling or loss of feasibility, DYLP will try the primal simplex algorithm before giving up.

Figure 4 shows the call structure of the dual simplex implementation.

9.1 Dual Top Level

dy_dual is a trivial shell. It assumes that the current solution is dual feasible. After calculating the objective (dy_calcobj), it calls the dual phase II routine dy_dual2 to do the optimisation.

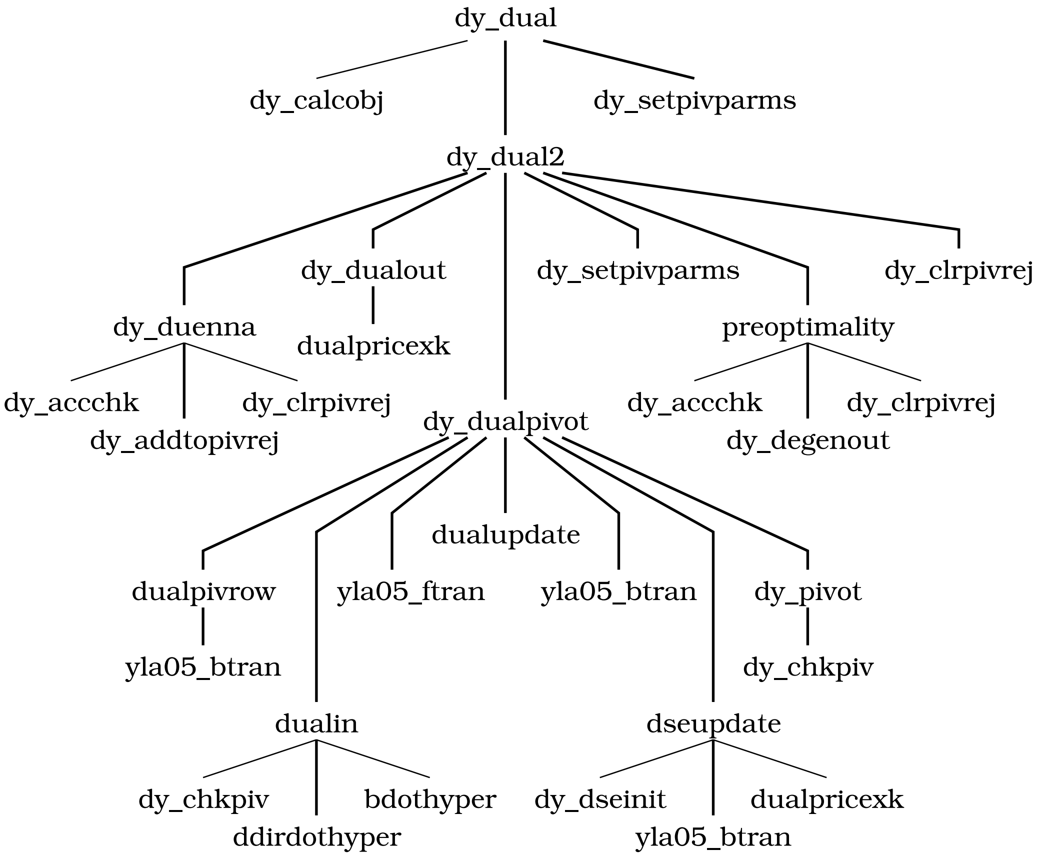

Figure 4: Call Graph for Dual Simplex

9.2 Dual Phase II

The overall flow of phase II of the dual algorithm is shown in Figure 5. The body of the routine is structured as two nested loops. The outer loop handles startup and termination, and the inner loop handles the majority of routine pivots.

On entry to dy_dual2, the outer loop is entered and dy_dualout is called to select the initial leaving variable. Then the inner loop is entered, and dy_dualpivot is called to perform the pivot. dy_dualpivot will calculate the coefficients of the pivot row (dualpivrow), select an entering variable (dualin), pivot the basis (dy_pivot), update the primal and dual variables (dualupdate), and update the DSE pricing information and reduced costs (dseupdate). For a routine pivot, dseupdate will also select a new leaving variable. dy_duenna evaluates the outcome of the pivot, handles error detection and recovery where possible, and performs the routine maintenance activities of accuracy checks and factoring of the basis. If there are no problems, the pivoting loop iterates, using the leaving variable selected in dseupdate. The loop continues until optimality is reached, the problem is determined to be primal infeasible (dual unbounded), or an exception or fatal error occurs.

The most common reason for a failure to select a leaving variable for the next pivot is that the current pivot was aborted due to numerical problems (an unsuitable pivot coefficient being the most common of these). Once dy_duenna has taken the necessary corrective action, the flow of control escapes to the outer loop and calls dy_dualout to select a new leaving variable.

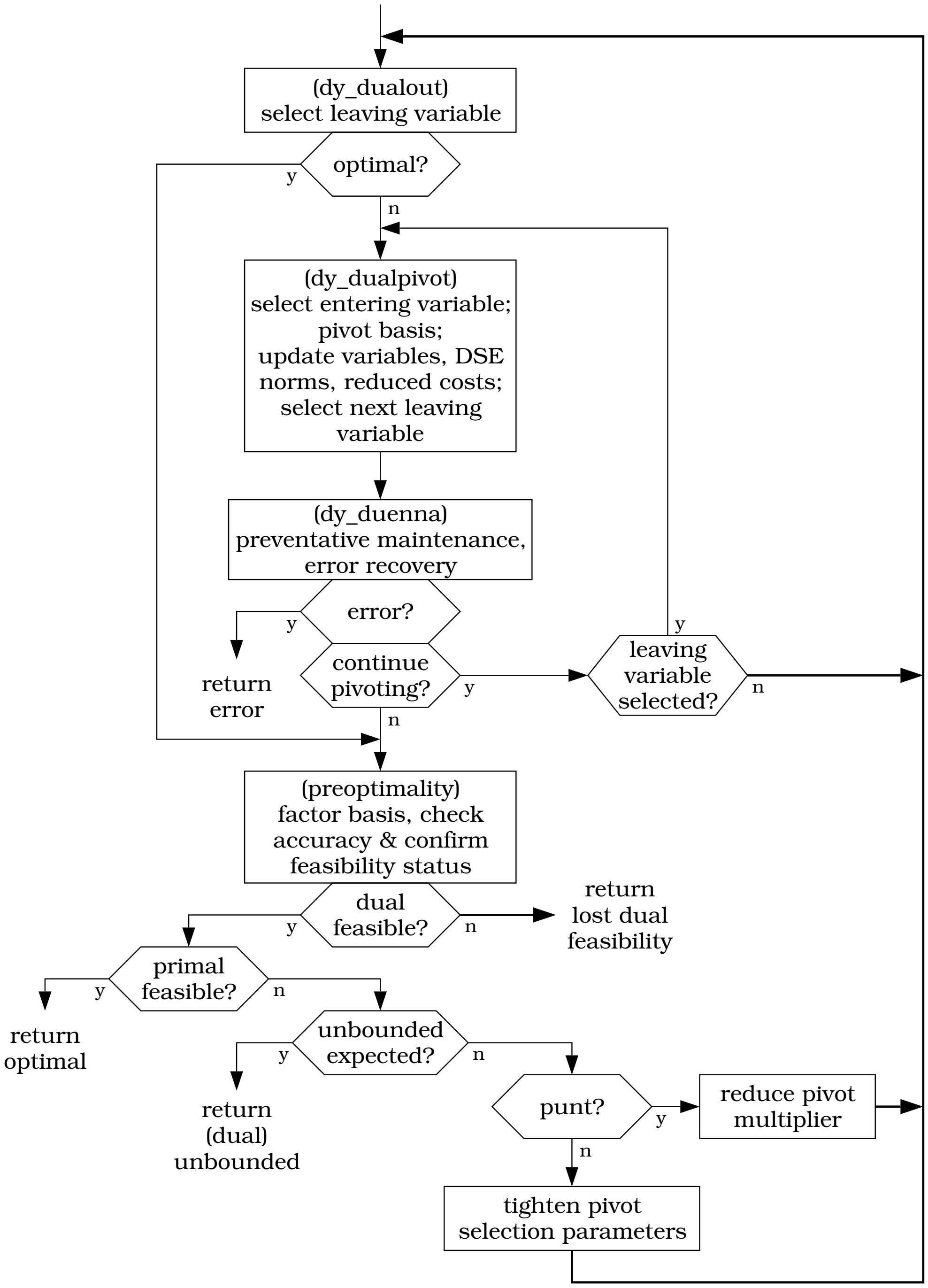

Figure 5: Dual Phase II Algorithm Flow

When dy_dualout indicates optimality (primal feasibility) or dy_dualpivot indicates optimality, dual unboundedness (primal infeasibility), or loss of dual feasibility, the inner loop ends and preoptimality is called for confirmation. preoptimality will factor the basis, check for accuracy, recompute the primal and dual variables, and confirm dual and primal feasibility status. If there are no surprises, dual phase II terminates with an indication of optimality, dual unboundedness, or loss of dual feasibility.

Loss of dual feasibility stems from loss of numeric accuracy, but it cannot be corrected within dual phase II. DYLP recovers by reverting to primal phase I.

Loss of primal feasibility can occur for two distinct reasons. The more common case is that some infeasible basic variables were flagged as ineligible for pivoting, resulting in a false indication of feasibility (hence optimality). This situation is termed a 'punt'. When these flags are removed as part of the preoptimality actions, primal feasibility is immediately lost. In this situation, the pivot selection multiplier dy_tols.pivot is loosened (multipled by .1) and pivoting resumes. If dy_tols.pivot is reduced to the zero tolerance dy_tols.zero without finding an acceptable pivot coefficient, the current solution is declared to be optimal. In the less common case, loss of primal feasibility occurs without a punt indication. The assumption is that the problem was loss of numerical accuracy. The pivot selection rules are tightened (*vid.* §7) and dual simplex iterations are resumed.

If the number of false terminations exceeds an arbitrary limit, the dual simplex algorithm is terminated. Other errors also result in termination of the dual simplex algorithm, and ultimately in an error return from DYLP.

9.3 Selection of the Leaving Variable

The selection of the leaving primal variable (entering dual variable) is made using the dual steepest edge criterion described in §4.2. As outlined above, the normal case is that the leaving variable for the following pivot will be selected as dseupdate updates the DSE pricing information for the current pivot. In various exceptional circumstances where this does not occur, the routine dy_dualout is called to make the selection.

In the event that all primal infeasible variables have been flagged as ineligible for pivoting, dy_dualout or dseupdate will signal a punt and pivoting will resume after the pivot selection rules have been relaxed as indicated above.

9.4 Selection of the Entering Variable

The selection of the entering primal variable (leaving dual variable) is made using standard dual pivoting rules and a set of tie-breaking strategies.

Let x_i be the leaving primal variable, for simplicity of exposition occupying basis position i, and let δ_i be the change in the entering dual variable associated with row i. β_i is obtained by calling yla05_btran to calculate $e_i B^{-1}$, where e_i is the unit row vector with 1 in position i. The pivot coefficient for x_k is $\overline{a}_{ik} = \beta_i a_k$.

Abstractly, we need to check $y_k = \overline{c}_k + \overline{a}_{ik}\delta_{ik}$ to find the maximum allowable δ_{ij} such that $y_k \geq 0 \; \forall k \in \mathcal{B}$ and $y_j = 0$ for some j. The index j of the entering primal variable x_j would be

$$j = \arg\min_k \left| \frac{\overline{c}_k}{\overline{a}_{ik}} \right|$$

leaving x_i	entering y_i	resulting $\overline{c}_i$	entering x_j	leaving y_j	initial $\overline{c}_j$	pivot $\overline{a}_{ij}$	$-\frac{\overline{c}_j}{\overline{a}_{ij}} = \overline{c}_i$
$\nearrow lb$	$0 \nearrow$	≥ 0	$lb \nearrow$	$\searrow 0$	≥ 0	< 0	$-\frac{(+)}{(-)} = (+)$
			$ub \searrow$	$\nearrow 0$	≤ 0	> 0	$-\frac{(-)}{(+)} = (+)$
$\searrow ub$	$0 \searrow$	≤ 0	$lb \nearrow$	$\searrow 0$	≥ 0	> 0	$-\frac{(+)}{(+)} = (-)$
			$ub \searrow$	$\nearrow 0$	≤ 0	< 0	$-\frac{(-)}{(-)} = (-)$

Table 1: Summary of Dual Simplex Pivoting Rules

for suitable $x_k \in N$.

In practice, it's impossible to explain this properly in the context of the dual problem without going deep into the details of the workings of the revised dual simplex algorithm (*vid.* [3]). Table 1 gives the rules in tabular form, from the perspective that when all is said and done, the leaving primal variable must end up nonbasic at bound and the sign of the reduced cost must be appropriate for that bound. Interpreting the table, the second line says that if the leaving variable will be made primal feasible by rising to its lower bound, the resulting reduced cost must be positive to retain primal optimality, hence the corresponding dual variable must enter by rising from 0. If the entering primal variable will be decreasing from its upper bound, the current reduced cost must be negative, hence the corresponding dual variable must leave by rising to 0. (Properly accounting for these *apparently* negative dual variables is the difficulty in trying to explain this from the dual simplex perspective.) The final columns simply illustrate that the sign of the pivot is well-defined from the update formula.

When degeneracy occurs and multiple variables x_k have equal values of δ_{ik} (typically all zero), DYLP provides a selection of tie-breaking strategies. The simplest is to select the first variable x_k such that $\delta_{ik} = 0$. A slightly more sophisticated strategy is to continue scanning variables eligible to enter and pick x_j such that $|\overline{a}_{ij}|$ is the maximum over the variables x_k with minimum δ_{ik}. DYLP also provides four additional strategies based on hyperplane alignment, as described in §5.

9.5 Pivoting

The first activity in dy_dualpivot is the calculation of the coefficients of the pivot row, $\overline{a}_{ik} = \beta_i N$, then dy_dualin is called to select the entering variable. Once the entering and leaving variables have been chosen, the actual pivot is performed in several steps. Prior to pivoting, the vector $\tau = B^{-1}\beta_k^T$ is calculated for use during the update of the DSE pricing information. The basis is pivoted next; this involves calls to yla05_ftran and dy_pivot, as outlined in §6.1. If the basis change succeeds, the primal and dual variables are updated by dualupdate using the iterative update formulæ of §3, and then the DSE pricing information and reduced costs are updated by dseupdate, using the update formulæ of §4.2. As a side effect, a leaving variable will be selected for the next pivot.

9.6 Preoptimality Actions

When optimality or (primal) infeasibility is indicated, (dual) preoptimality is called for confirmation. Its first action is to call dy_clrpivrej to remove any 'ineligible to pivot' flags on the basic variables. It then calls dy_accchk, requesting it to factor the basis (to minimise roundoff error), perform primal and dual accuracy checks, recalculate the primal and dual variables, and then confirm that the current solution is still primal and dual feasible (*i.e.*, primal feasible and optimal).

10 Primal Simplex

The primal simplex implementation in DYLP is a two-phase algorithm which incorporates projected steepest edge (PSE) pricing as outlined by Forrest & Goldfarb [2], an anti-degeneracy algorithm which builds on a perturbation method described by Ryan & Osborne [7], and a lightweight anti-degeneracy algorithm based on hyperplane alignment. Figure 6 shows the call structure of the primal simplex implementation.

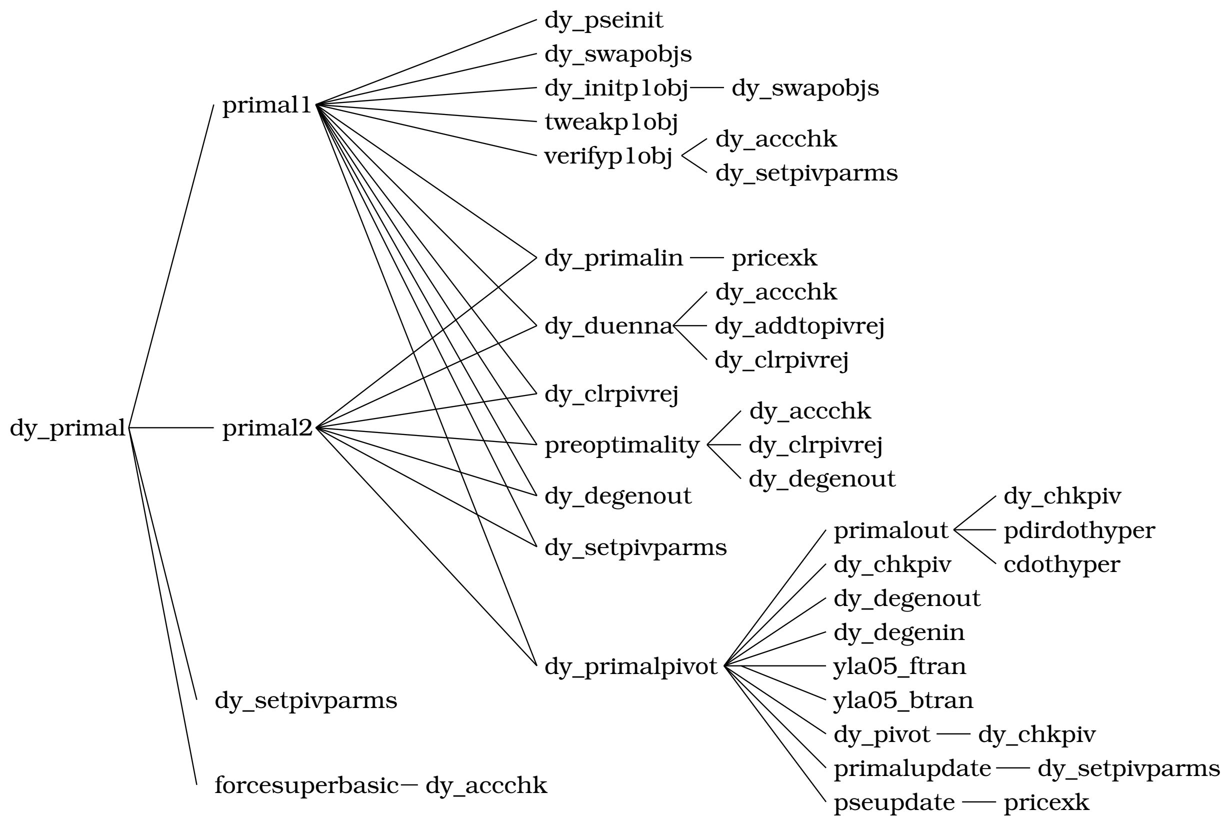

Figure 6: Call Graph for Primal Simplex

10.1 Primal Top Level

dy_primal controls the application of phase I (primal1) and phase II (primal2) of the primal simplex algorithm. On the initial call, primal1 is called only if the problem supplied to dy_primal is infeasible.

The primary purpose of dy_primal is to provide a loop which allows a limited number of reversions (currently 10) to phase I if primal feasibility is lost during phase II. For convenience, superbasic variables are forced to bound at this transition; this is handled by forcesuperbasic.

10.2 Primal Phase I

The overall flow of phase I of the primal simplex is shown in Figure 7.

An artificial objective is used to guide pivoting to feasibility. The (minimisation) coefficients assigned to variables are -1 for variables below their bound, 0 for variables within bounds, and +1 for variables above their bound. On entry to phase I, dy_initp1obj handles any necessary data structure manipulation, builds a working set containing all infeasible variables, and sets appropriate objective coefficients.

Pivoting is handled by two nested loops. The outer loop handles startup and termination, and the inner loop handles the majority of routine pivots. A pivot iteration normally consists of three stages: the actual pivot and variable updates, routine maintenance checks, and updating of the objective. On entry to the outer loop, dy_primalin is called to select the initial entering variable.

In the inner loop, dy_primalpivot is called to perform the pivot. Given an entering variable, it chooses a leaving variable (primalout), performs the pivot (dy_pivot), updates the primal and dual variables (primalupdate), and updates the PSE pricing information and reduced costs (pseupdate). For a routine pivot, pseupdate will also choose the entering variable for the next pivot.

dy_duenna evaluates the outcome of the pivot, handles error detection and recovery where possible, and performs the routine maintenance activities of accuracy checks and factoring of the basis.

After each pivot, tweakp1obj scans the working set and the objective function is adjusted to reflect any changes in feasibility for variables in the set. The reduced costs and dual variables are adjusted or recalculated as required. When all variables in the working set are primal feasible, tweakp1obj will give a preliminary indication of primal feasibility. If verifyp1obj confirms that all variables are primal feasible, the pivoting loop will end. If accumulated numerical inaccuracy has caused previously feasible variables to become infeasible, the pivot selection parameters will be tightened, dy_initp1obj will be called to build a new working set and objective, and pivoting will resume.

It can happen that no entering variable is selected in pseupdate, or changes to the objective coefficients make it necessary to select a new entering variable. The single most common reason is a pivot in which a nonbasic variable moves from one bound to the other — since there is no basis change, pseupdate is not called. The second situation arises when a variable gains feasibility but remains basic. Changing an entry of c^B other than the entry corresponding to the entering variable can potentially affect all reduced costs, and the variable selected in pseupdate may no longer be the best (or even a good) choice. The flow of control is redirected to the outer loop, where dy_primalin will be called to select an entering variable. If the current pivot is aborted due to numerical problems (an unsuitable pivot coefficient being the most common of these), no entering variable will be selected. Once dy_duenna has taken the necessary corrective action, the flow of control again moves to the outer loop to call dy_primalin.

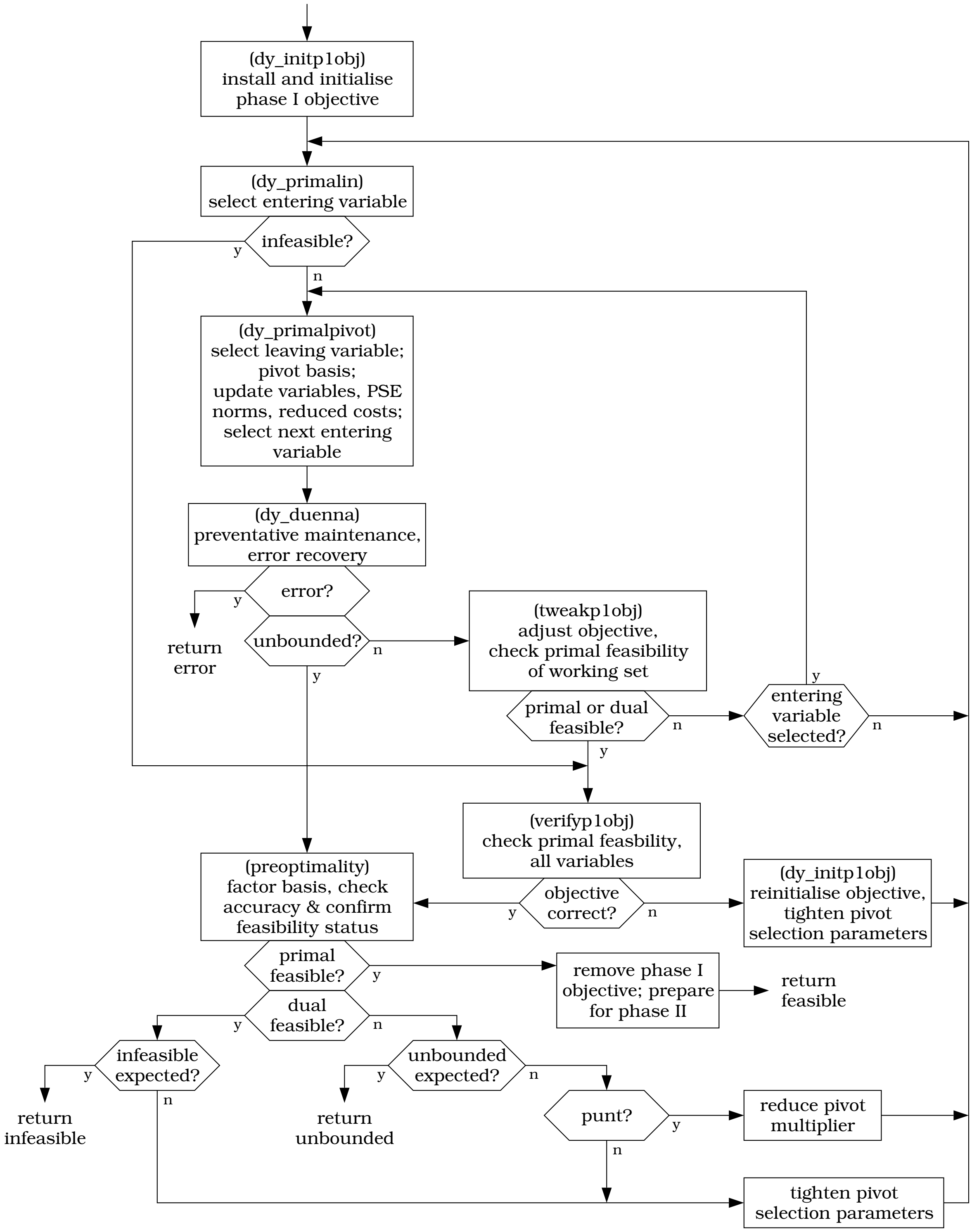

Figure 7: Primal Phase I Algorithm Flow

If no entering variable can be selected but some variables are still primal infeasible, the problem is infeasible; this is detected in dy_primalin and pseupdate. If no leaving variable can be selected, the problem is unbounded[3]; this is detected in primalout. Degeneracy is also detected when primalout attempts to select a leaving variable. The actions of the anti-degeneracy algorithms are described in §10.6.

Detection of primal feasibility, unboundedness, or infeasibility terminates the inner pivoting loop and preoptimality is called for confirmation. It factors the basis, performs accuracy checks, and confirms primal and dual feasibility. If there are no surprises, primal phase I terminates with an indication of optimality (primal feasibility), unboundedness, or infeasibility. In any event, if preoptimality reports that the solution is primal feasible, phase I will end with an indication of optimality even if it was not expected from the pivot loop termination condition.

Loss of dual feasibility can occur for two distinct reasons. The more common case is that some nonbasic variables were flagged as ineligible for pivoting, resulting in a false indication of dual feasibility). This situation is termed a 'punt'. When these flags are removed as part of the preoptimality actions, dual feasibility is immediately lost. In this situation, the pivot selection multiplier dy_tols.pivot is loosened (multipled by .1) and pivoting resumes. If dy_tols.pivot is reduced to the zero tolerance dy_tols.zero without finding an acceptable pivot coefficient, the problem is declared to be infeasible. In the less common case, loss of dual feasibility occurs without a punt indication. The assumption is that the problem was loss of numerical accuracy. The pivot selection rules are tightened (*vid.* §7) and primal simplex iterations are resumed.

Loss of primal feasibility can occur when the basis is factored during the preoptimality actions. The pivot selection parameters are tightened and pivoting resumes.

If the number of false terminations exceeds an arbitrary limit, the primal simplex algorithm is terminated. Other errors also result in termination of the primal simplex algorithm, and ultimately in an error return from DYLP.

If a primal feasible solution has been found, the original objective will be restored before returning from primal1. The transition to phase II entails calculating the objective, dual variables, and reduced costs for the original objective. If the problem is infeasible or unbounded, the phase I objective is left in place and DYLP will use it as it attempts to activate variables or constraints to deal with the problem.

10.3 Primal Phase II

The overall flow of phase II of the primal simplex is shown in Figure 8. The major differences from phase I are that the problem is know to be feasible and the original objective function is used instead of an artificial objective function. This considerably simplifies the flow of primal2.

The inner pivoting loop has only two steps: the pivot itself (dy_primalpivot) and the maintenance and error recovery functions (dy_duenna). The inner pivoting loop ends when the problem is found to be optimal (dy_primalin or pseupdate) or unbounded (primalout). preoptimality is called to factor the basis, perform accuracy checks, and confirm primal and dual feasibility. If there are no surprises, primal phase II will end with an indication of optimality or unboundedness. Loss of dual feasibility (including punts) is handled as described for primal phase I. Loss of primal feasibility causes primal2 to return with an indication that it has lost primal feasibility, and DYLP will revert to primal phase I.

[3]An unbounded result is possible in the context of DYLP, even in phase I, because the active constraint system may not contain sufficient constraints to bound the solution polytope.

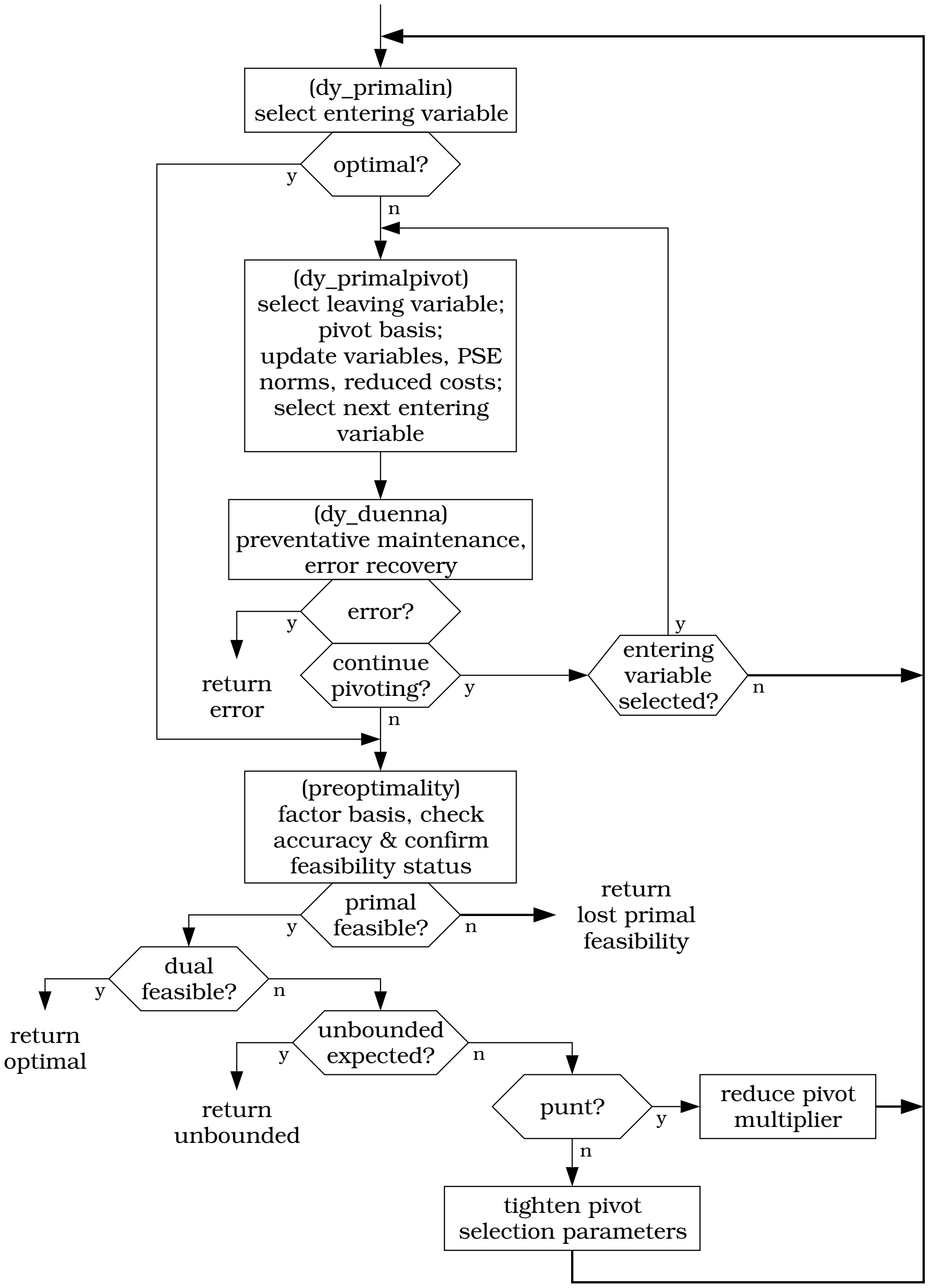

Figure 8: Primal Phase II Algorithm Flow

10.4 Selection of the Entering Variable

Selection of the entering variable x_j for a primal pivot is made using the primal steepest edge criterion described in §4.1. As outlined above, the normal case is that the entering variable for the following pivot will be selected as pseupdate updates the PSE pricing information for the current pivot. In various exceptional circumstances where this does not occur, the routine dy_primalin is called to make the selection.

In the event that all suitable nonbasic variables have been flagged as ineligible for pivoting, dy_primalin or pseupdate will signal a punt and pivoting will resume after the pivot selection rules have been relaxed as indicated above.

10.5 Selection of the Leaving Variable

Selection of the leaving variable x_i is made using standard primal pivoting rules and a set of tie-breaking strategies.

Abstractly, we need to check $x_k = \overline{b}_k - \overline{a}_{kj}\Delta_{kj}$ to find the maximum allowable Δ_{kj} such that $l_k \leq x_k \leq u_k \; \forall k \in B$ and $x_i = l_i$ or $x_i = u_i$ for some i. The index i of the leaving variable will be

$$i = \arg\min_k \left| \frac{\overline{b}_k}{a_{kj}} \right|$$

for suitable $x_k \in B$.

The primal pivoting rules are the standard set for revised simplex with bounded variables, and are summarised in Table 2. During phase 1, when a variable is infeasible below its lower bound

leaving x_i	entering x_j	pivot $\overline{a}_{ij}$
$\nearrow ub$	$lb \nearrow$	< 0
	$ub \searrow$	> 0
$\searrow lb$	$lb \nearrow$	> 0
	$ub \searrow$	< 0

Table 2: Summary of Primal Simplex Pivoting Rules

and must increase to become feasible, DYLP sets the limiting Δ_j based on the upper bound, if it is finite, and uses the lower bound only when the upper bound is infinite. Similarly, when a variable must decrease to its upper bound, the lower bound is used to calculate the limiting Δ_j if it is finite.

When degeneracy occurs and multiple variables x_k have equal values of Δ_{kj} (typically all zero), DYLP provides a selection of tie-breaking strategies. The simplest is to select the first variable x_k such that $\Delta_{kj} = 0$. A slightly more sophisticated strategy is to continue scanning variables eligible to leave and pick x_i such that $|\overline{a}_{ij}|$ is the maximum over the variables x_k with minimum $|\Delta_{kj}|$. DYLP also provides four additional strategies based on hyperplane alignment, as described in §5.

10.6 Anti-Degeneracy Using a Perturbed Subproblem

In the primal simplex, DYLP implements an anti-degeneracy algorithm using a perturbed subproblem in addition to the lightweight methods using hyperplane alignment described in §5. It builds on a method described by Ryan & Osborne [7].

In their presentation, all variables are assumed to have lower bounds of 0 and upper bounds of ∞. When degeneracy is detected, a restricted subproblem is formed consisting only of the constraints involved in the degeneracy (*i.e.*, constraints i such that $\overline{b}_i = 0$). The values $\overline{b}_i$ are given (relatively) large perturbations and pivots are performed within the context of the restricted subproblem until a direction of recession from the degenerate vertex is found (indicated by apparent unboundedness). The original unperturbed values of $\overline{b}_i$ are then restored (since all pivots were, in actuality, simply changes of basis while remaining at the degenerate vertex) and the full problem is resumed.

The trick to implementing this algorithm in the context of variables with arbitrary upper and lower bounds is to distinguish between apparent motion due to the introduced perturbations and real motion which is nonetheless limited by a bound on a variable. DYLP uses an array, dy_brkout, to record the proper direction of change (away from the current bound) for a nondegenerate but bounded movement.

A second, more subtle problem, is that the perturbation for a given variable must be sufficiently small to avoid a false indication of a nondegenerate pivot. DYLP scales the perturbation to be at most $.001(u_i - l_i)$, but there is no easy way to guarantee that this is sufficiently small. Consider two variables x_i and x_k, and assume that they occupy rows i and k in the basis, with perturbed values $\tilde{b}_i$ and $\tilde{b}_k$, respectively. For concreteness, assume that each was originally degenerate at its lower bound, so that a pivot which resulted in one variable leaving at its upper bound would be nondegenerate. For $\overline{a}_{ij}$ and $\overline{a}_{kj}$ of appropriate sign to move x_i toward l_i and x_k toward u_k, given a situation where $|\overline{a}_{ij}| \ll |\overline{a}_{kj}|$, it is not possible to assure that

$$\frac{\tilde{b}_i - l_i}{\overline{a}_{ij}} < \frac{u_k - \tilde{b}_k}{\overline{a}_{kj}}$$

without actually testing each pair. In this case, the perturbation introduced for x_i is too large, and the resulting Δ_{ij} *appears* to allow x_k to become the limiting variable, leaving the basis with a bounded but nondegenerate change. When DYLP detects this problem, it will reduce the perturbation by a factor of 10 and form the restricted subproblem again. If a (small) limit on the number of attempts is exceeded, DYLP simply gives up and takes a degenerate pivot.

A second problem occurs when a perturbation is so small as to be indistinguishable next to the bound. Specifically, the test to determine if a variable x_i is at bound is dy_tols.zero$(1 + |bnd_i|) < |x_i - bnd_i|$. If bnd_i is large, the perturbation can be swamped. This situation can arise if u_i and l_i as given to DYLP are nearly equal, or due to reduction of the perturbation as described in the previous paragraph.

11 Constraint Management

Constraint management activities can be separated into selection of the initial constraint set, activation of violated or bounding constraints, and deactivation of loose constraints. In general, the goal is to maintain an active constraint system which is a subset of the original constraint system that consists only of equalities and those inequalities necessary to define the optimal solution. DYLP expects that all constraints will be equalities or ≤ inequalities. Figure 9 shows the call structure for the constraint activation and deactivation routines.

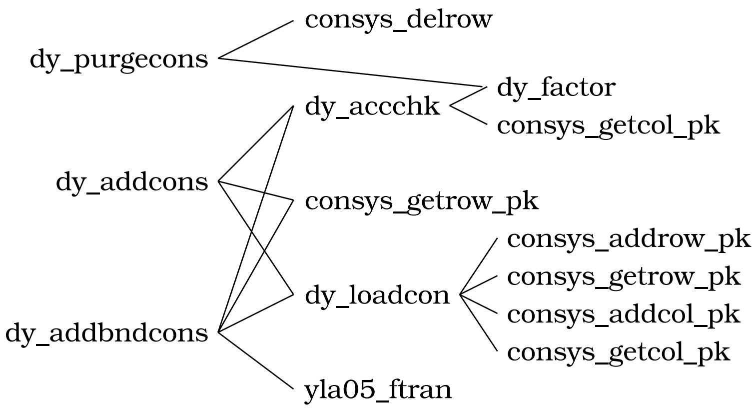

Figure 9: Call Graph for Constraint Management Routines

During construction of the initial constraint system, any variables referenced in a constraint are activated with the constraint. During subsequent constraint activation phases, no variables are activated with the exception of the logical variable for the constraint, which is created and made basic. This ensures that dual feasibility is maintained. Additional variables referenced by the constraint may be activated in subsequent variable activation phases if they are needed. Similarly, to avoid potential problems with the set of basic variables, when a constraint is deactivated only the slack variable for the constraint is deleted.

11.1 Initial Constraint Selection

For a warm or hot start, the initial active constraint system is completely determined from the basis supplied by the client.

For a cold start, where no initial basis is supplied, the size of the initial active constraint system defaults to 25% of the constraints of the original system. All equalities are activated first. If space remains, an initial set of inequalities is selected based on alignment with the objective function. The inequalities are sorted according to $-c \cdot a/\|c\|\|a\|$ and activated in sort order until the active constraint system is filled out.

11.2 Activation of Constraints

DYLP enters the constraint activation phase whenever the system is found to be unbounded or optimal for the set of active constraints and all variables (active and inactive). When the system is found to be optimal, DYLP calls dy_addcons to search the inactive constraints for violated constraints. When the system is found to be unbounded, DYLP also calls dy_addbndcons to search the inactive constraints for constraints which might block the direction of recession.

dy_addcons searches for violated constraints using a straightforward scan of the inactive constraints, evaluating each one at the at the current value of x and activating any which are violated.

Unboundedness is detected by the primal simplex. (In the context of DYLP, the dual simplex is not called until the primal simplex has found an initial optimal solution.) When it is discovered, DYLP calls dy_addbndcons to search for constraints which could potentially block the direction of recession η_j by looking for constraints such that $a_i \cdot \eta_j > 0$ for x_j increasing, or $a_i \cdot \eta_j < 0$ for x_j decreasing.

When a constraint is activated, the set of basic variables is augmented with the slack variable for the constraint. Because the constraint is not tight, the value of the associated dual is zero. The basis will change, but the values of other active primal variables will remain the same. Since the new slack variables are not part of the PSE reference frame, the projected column norms associated with PSE pricing are unchanged. Because the objective coefficients associated with the slack variables are 0, the values of the preexisting dual variables and the reduced costs remain unchanged.

DYLP refactors the basis and performs accuracy checks at the end of each constraint activation phase. For accuracy, it also recalculates the primal and dual variables.

11.3 Deactivation of Constraints

Constraint deactivation is handled by dy_purgecons. DYLP implements three options for constraint deactivation, normal aggressive, and fanatical.

When normal constraint deactivation is specified, DYLP will only deactivate inequalities which are strictly loose. The inequalities are located by scanning the basis for slack variables which are strictly within bounds. The corresponding inequality is deactivated and removed from the active constraint system along with its associated slack variable. The basis is patched, if necessary, by moving the variable which is basic in the position of the deactivated inequality to the basis position which was occupied by the inequality's associated slack.

Under aggressive constraint deactivation (the default), DYLP will deactivate tight inequalities whose associated dual variable is zero, in addition to loose inequalities.

Under fanatical constraint deactivation, DYLP will deactivate any constraint (equality or inequality) whose associated dual is zero.

As with activation of constraints, deactivation of constraints changes the basis and DYLP will refactor and recalculate the primal and dual variables. The dual variables do not change, nor do the reduced costs of the remaining variables, provided that the cost coefficient of the slack variable is 0. In general, the PSE column norms will be changed because the deleted slack variables may be part of the reference frame. DYLP opts to reset the reference frame to deal with this, rather than updating or recalculating the column norms.

12 Variable Management

Variable management activities can be separated into the activation of an initial set of variables when initialising the constraint system, activation of variables with favourable reduced costs, and deactivation of variables with unfavourable reduced costs. When variable activation takes place because it is not possible to attain primal feasibility with the active variable set, the situation becomes a little more complicated. Figure 10 shows the call structure for the variable activation and deactivation routines.

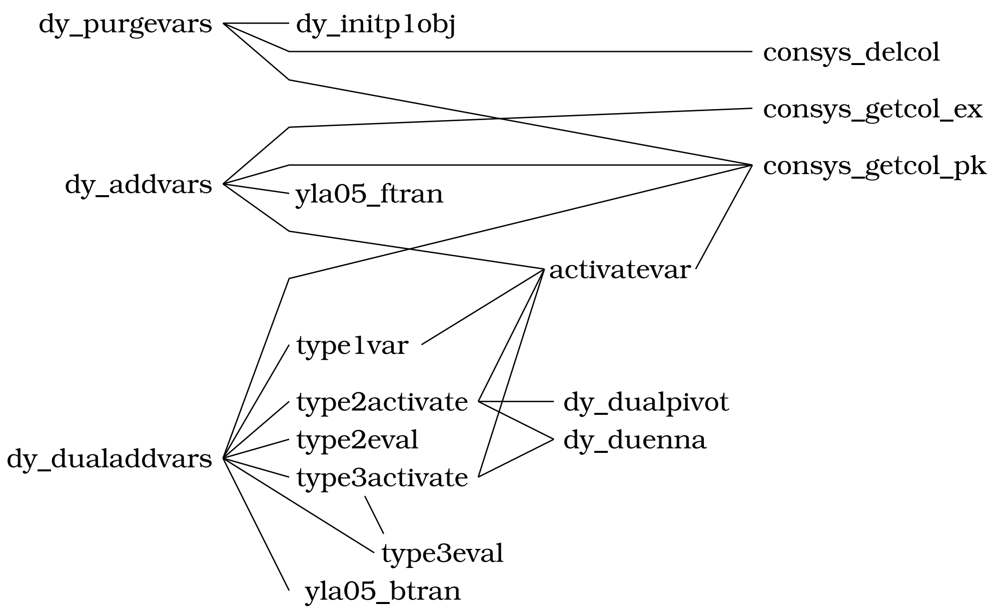

Figure 10: Call Graph for Variable Management Routines

12.1 Initial Variable Selection

For a cold start, the initial set of active variables is completely determined by the initial set of constraints. All variables mentioned in the constraints are activated.

For a hot start, the initial set of active variables is the active set for the optimal solution from the previous call to dylp.

For a warm start, the set of active constraints is specified by the basis. The initial set of active variables can be determined from the constraints as for a cold start, or the client can specify a set of active variables.

12.2 Activation of Variables

DYLP looks for variables to activate whenever optimality is attained for the current set of constraints and variables, or when the active system is found to be infeasible. The set of inactive variables is scanned and any variables with favourable reduced costs are activated and placed in the primal nonbasic partition (with some exceptions outlined below).

If an optimal solution has been found for the active constraint system by either primal or dual simplex, dy_addvars is called to select a set of variables to be activated. The reduced costs are calculated using the original objective function for the problem.

If phase I of the primal simplex has found the problem to be infeasible, dy_addvars is again used to select the set of variables to be activated, but the reduced costs are calculated using the phase I objective (as described in §10.2).

Activating a variable in these two cases requires calculating the reduced cost and projected column norm for use in PSE pricing. Primal feasibility is retained (or at least made no worse), and there is no change to the basis, primal or dual variables, or DSE pricing information. Following the activation of variables, DYLP returns to phase I or II of the primal simplex.

If the dual simplex has found the problem to be primal infeasible (dual unbounded), the problem of selecting variables to add is more complicated. The goal is to activate one or more variables and return to dual simplex iterations. The activity is perhaps better viewed from the perspective of looking for dual constraints which will bound the problem.

The selection of the candidate entering dual variable y_i (leaving primal variable x_i) has fixed the direction of travel, ζ_i. The best outcome will be to add variables (dual constraints) which block travel in the direction ζ_i. If that isn't possible (because activating any bounding dual constraint would result in the loss of dual feasibility) a second possibility is to activate variables which will change the dual reduced costs (the values of the primal basic variables) so that a different dual variable y_k is selected to enter. The hope is that motion in a different direction ζ_k may make it possible to activate constraints which will bound the dual without loss of feasibility.

The subroutine dy_dualaddvars controls the search process, and can activate three classes of variables, for convenience called type 1, type 2, and type 3. Type 1 variables are preferred, as dy_dualaddvars can activate any number of them in a given call. If there are no type 1 variables, a single type 2 or type 3 variable will be activated, with type 2 variables preferred over type 3. A single type 2 or type 3 variable is activated because predicting the effect of the activation would entail duplicating large pieces of the dual simplex code within dy_dualaddvars. In the circumstance, simply returning to dual simplex iterations is preferable.

Type 1 variables are those variables which constitute dual constraints that may bound the dual problem and which can be activated and placed in the primal nonbasic partition without losing dual feasibility.

Type 2 variables are those variables which constitute constraints that may bound the dual problem and which, while not dual feasible if activated into the primal nonbasic partition, will give a dual feasible solution if activated and immediately pivoted into the basis. This step amounts to activating a dual constraint which renders the current solution infeasible, but feasibility can be regained with a single pivot. In the context of Table 1 in §9.4, this amounts to selecting a pivot with the signs of $\overline{c}_j$ and $\overline{a}_{ij}$ reversed. The pivot is sufficiently similar to a normal dual pivot that it can be handled by dy_dualpivot. It is not standard in that the entering primal variable will move away from its bound toward the infeasible side (*e.g.*, x_j would enter falling from its lower bound with $\overline{c}_j < 0$ and $\overline{a}_{ij} > 0$). One such variable can be activated on each call to dy_dualaddvars.

Type 3 variables are variables which are not dual feasible at their current bound but which will reduce the infeasibility of the leaving primal variable if activated and moved to their opposite bound. The motivation for activating a type 3 variable is that it makes the reduced cost of y_i less desireable, so that some other variable y_k can be selected to enter (thus moving in a different direction ζ_k). This is a bound-to-bound pivot, entirely outside the normal realm of dual pivots, and as such is handled entirely within type3activate.

The effect of activating type I variables is just as outlined for the case of activating variables and proceeding to primal simplex, with the exception that PSE pricing information is not updated. (The PSE reference frame will be reinitialised when primal simplex is resumed.)

For type 2 variables, the pivot will cause a change of basis. dy_dualpivot will take care of the required calculations and updates in the context of dual simplex. For type 3 variables, the basis doesn't change, and the values of the dual variables and DSE norms are unchanged. The values of the primal variables do change, however, and this changes the DSE pricing information.

12.3 Deactivation of Variables

Deactivation of variables occurs only after an optimal solution is found for the active constraint system. The subroutine dy_purgevars is called to deactivate variables according to a user-specified threshold, expressed as a percentage of the maximum unfavourable reduced cost over all active variables.

Specifically, dy_purgevars scans the reduced costs of the active variables and determines $\check{c} = \max_{\{k:c_k<0\}} |c_k|$ and $\hat{c} = \max_{\{k:c_k>0\}} |c_k|$. It then deactivates variables with $c_k > \hat{c}$(dy_tols.purgevar) or $c_k < \check{c}$(dy_tols.purgevar).

13 DYLP Performance

DYLP was developed in the context of the **bonsaiG** MILP code, and the data presented here shows its performance on the LP relaxations of the 59 problems in MIPLIB 3.0 [1]. Table 3 gives the problem size for each problem, and the number of pivots and the time (in cpu seconds) required for the solution of the LP relaxation. The machine used to run the benchmarks was a 333 MHz. Sun Ultra-5 with 2 MB cache and 128 MB memory.

14 DYLP Interface

DYLP is called as

**lpret_enum dylp (lpprob_struct *orig_lp, lpopts_struct *orig_opts, lptols_struct *orig_tols,
lpstats_struct *orig_stats)**

The orig_lp structure (§14.1) specifies the constraint system and (optionally) control flags, an initial active variable set, and a (primal or dual feasible) basic solution. It returns the final solution and some additional information about the solution.

The orig_opts structure (§15) specifies option settings to control DYLP's actions, and the orig_tols structure (§16) specifies a number of numeric tolerances and related control information.

The optional structure orig_stats (§17) can be used (in conjunction with conditionally compiled code) to return detailed statistics about DYLP's actions.

In addition to the main routine, dylp, various printing and utility routines are provided.

14.1 The LP Problem Specification

The structure lpprob_struct *orig_lp is used to define the LP problem to DYLP and to return the answer to the client. It contains status fields, the constraint system, an active variable vector, a basis vector, a status vector, and vectors for the primal and dual variables. Each field is discussed below; for precise details, the reader should consult the file dylp.h.

actvars A data structure to specify the set of active variables.

Problem	Rows	Columns	Pivots	Time
10teams	230	2025	986	03.17
air03	124	10757	425	16.48
air04	823	8904	5034	02:57.30
air05	426	7195	2025	47.77
arki001	1048	1388	1691	04.77
bell3a	123	133	208	00.07
bell5	91	104	136	00.06
blend2	274	353	405	00.59
cap6000	2176	6000	6033	11.29
dano3mip	3202	13873	71228	50:47.64
danoint	664	521	1143	02.11
dcmulti	290	548	1251	01.32
dsbmip	1182	1886	3351	13.00
egout	98	141	53	00.03
enigma	21	100	16	00.01
fast0507	507	63009	8890	08:16.47
fiber	363	1298	228	00.33
fixnet6	478	878	399	00.35
flugpl	18	18	11	00.00
gen	780	870	321	00.31
gesa2	1392	1224	2213	03.44
gesa2_o	1248	1224	811	01.42
gesa3	1368	1152	2838	05.43
gesa3_o	1224	1152	1082	02.16
gt2	29	188	153	00.04
harp2	112	2993	1884	01.26
khb05250	101	1350	590	00.60
l152lav	97	1989	387	01.29
lseu	28	89	76	00.02
misc03	96	160	65	00.04
misc06	820	1808	3345	15.82
misc07	212	260	149	00.17
mitre	2054	10724	8619	57.05
mod008	6	319	46	00.03
mod010	146	2655	499	01.92
mod011	4480	10958	9341	07:05.38
modglob	291	422	358	00.21
noswot	182	128	165	00.09
nw04	36	87482	163	01:45.68
p0033	16	33	41	00.00
p0201	133	201	114	00.07
p0282	241	282	262	00.11
p0548	176	548	479	00.25
p2756	755	2756	104	00.29
pk1	45	86	57	00.02
pp08a	136	240	184	00.10
pp08aCUTS	246	240	344	00.31
qiu	1192	840	2186	17.83
qnet1	503	1541	1564	03.74
qnet1_o	456	1541	1719	03.91
rentacar	6803	9557	2971	04:31.68
rgn	24	180	73	00.02
rout	291	556	579	00.76
set1ch	492	712	492	01.29
seymour	4944	1372	11135	54.41
stein27	118	27	41	00.02
stein45	331	45	67	00.04
vpm1	234	378	178	00.18
vpm2	234	378	261	00.24

Table 3: DYLP performance on the LP relaxations of the MIPLIB problem set

(i) For a cold or host start, a structure can be provided to return the final set of active variables. For a warm start, an initial set of active variables can be provided. This information will be used only if the lpctlACTVARSIN flag is set in the ctlopts field.

(o) The final set of active variables. If no vector was supplied as an input parameter, DYLP will allocate one on output. Active variable information is returned only if the lpctlACTVARSOUT flag is set when DYLP is called. Valid information is returned only if an optimal solution is found. If valid information is not returned, the lpctlACTVARSOUT will be reset.

basis A data structure for the LP basis. Because the set of active constraints at optimum will not, in general, include all constraints, the basis vector specifies the constraint and the primal variable in each basis position.

(i) For a cold or hot start, a structure can be provided to return the final basis. For a warm start, an initial basis must be provided.

(o) The final basis. If no vector was supplied as an input parameter, DYLP will allocate one on output.

colsze The allocated column capacity of the data structure. If provided, the status and actvars data structures must be capable of holding this many entries. (If this is insufficient to return the answer, DYLP will reallocate the data structures.)

consys The constraint system, in the format described for the CONSYS constraint system subroutine library [4].

ctlopts A vector of flags used to specify optional actions and status. The current set of flags can be used to control allocation and deallocation of internal DYLP data structures, specify the presence of changes to the problem bounds and objective, and specify the exchange of active variable information.

iters The total number of successful simplex pivots.

lpret The return code from the simplex routine.

If no errors occur, the code should be one of lpOPTIMAL (optimal), lpINFEAS (primal infeasible), or lpUNBOUNDED (primal unbounded).

Error returns include lpPUNT (nonbasic variables exist with favourable reduced costs, but the cannot be pivoted due to unsuitable pivot coefficients), lpLOSTFEAS (primal feasibility has been lost and DYLP has exceeded its limit on attempts to regain feasibility), lpCYCLING (the limit on pivots without improvement in the objective has been exceeded, due to cycling or stalling), lpITERLIM (a limit on pivots per phase or total pivots has been exceeded), lpACCCHK (a numerical accuracy check has occurred), lpSINGULAR (the basis has become singular), lpNOSPACE (LA05 could not acquire sufficient space to maintain the basis inverse), lpFATAL (an unspecified fatal error has occurred), and lpINV (DYLP aborted due to internal confusion).

obj For an optimal result, the value of the objective function. For an infeasible result, the total primal infeasibility. For an unbounded result, the index of the unbounded variable, negated if the variable can decrease without bound, positive if it can increase without bound. For any other return status, this field is undefined.

phase Identifies the phase of the dynamic simplex algorithm, as outlined in §8. Used to specify a starting simplex phase on input, and return the ending phase on output.

(i) If no basis is supplied, must be set to dyINV. If a basis is supplied, the phase should be set to dyPRIMAL1 unless the basis is known to be primal feasible (dyPRIMAL2) or dual feasible (dyDUAL). If the phase is set to dyDONE, DYLP will assume that the only purpose of the call is to free previously allocated data structures.

(o) The termination phase of the dynamic simplex algorithm; should be dyDONE unless an error has occurred, in which case it'll be dyINV.

rowsze The allocated row capacity of the data structure. If provided, the basis, x, and y data structures must be capable of holding this many entries. (If this is insufficient to return the answer, DYLP will reallocate the data structures.)

status A data structure to hold the status of variables. For nonbasic variables, an entry is a DYLP status code. For basic variables, an entry is the negative of the basis position.

(i) For a cold or hot start, a structure can be provided to return the final status. For a warm start, an initial status must be provided.

(o) The final status vector. The value of nonbasic primal variables is returned through this vector. If no vector was supplied as an input parameter, DYLP will allocate one on output.

x A data structure to hold the values of the basic primal variables.

(i) A structure can be provided to return the final values.

(o) The values of the basic primal variables, indexed by basis position. If no vector was supplied as an input parameter, DYLP will allocate one on output.

y A data structure to hold the values of the dual variables.

(i) A structure can be provided to return the final values.

(o) The values of the dual variables, indexed by basis position. If no vector was supplied as an input parameter, DYLP will allocate one on output.

14.2 Other External Routines

As mentioned in the introduction, DYLP was developed for use in a mixed-integer linear programming environment. It provides two additional routines, for pricing a column and calculating up and down penalties, which are useful in this environment. They are described briefly here; for details of the interface, consult the code in dy_penalty.c.

bool dy_pricenbvars (lpprob_struct *orig_lp, flags priceme,
double **p_ocbar, int *p_nbcnt, int **p_nbvars)

This routine calculates the reduced cost of nonbasic variables, tapping the DYLP data structures for active variables and calculating the reduced cost as needed for inactive variables. Where nonbasic slack variables are present in the active system, they are identified with the negative of the index of the associated constraint. The interface is designed to return a compact list of priced variables in p_nbvars and p_ocbar. priceme provides limited additional control by allowing the client to restrict the status of the nonbasic variables to be priced.

bool dy_calcpeni (lpprob_struct *orig_lp, int oxindx,
int nbcnt, int *nbvars, double *cbar, double *p_upeni, double *p_dpeni)

This routine calculates the up and down penalties associated with forcing the basic variable x_i to $\lceil x_i \rceil$ and $\lfloor x_1 \rfloor$, respectively.

The up penalty is $upen_i = \min_k -(\lceil x_i \rceil - x_i)\dfrac{\overline{c}_k}{\overline{a}_{ik}}$ for $\{k \in N | \overline{a}_{ik} > 0 \wedge x_k < u_k \vee \overline{a}_{ik} < 0 \wedge x_k > l_k\}$.

The down penalty is $dpen_i = \min_k -(\lfloor x_i \rfloor - x_i)\dfrac{\overline{c}_k}{\overline{a}_{ik}}$ for $\{k \in N | \overline{a}_{ik} < 0 \wedge x_k < u_k \vee \overline{a}_{ik} > 0 \wedge x_k > l_k\}$.

15 DYLP Options

DYLP is intended to be flexible, and as such has a large number of options which are specified as fields of the lpopts_struct *orig_opts structure. Each is described briefly below; again, the reader is encouraged to consult dylp.h for details [4].

active The three values vars, cons, and init control the initial allocated size of the active constraint system and the number of constraints activated at initialisation

- vars, cons These options specify the amount of space allocated with the active constraint system is first created. They do not represent limits (the constraint system will be expanded as required); rather, they are provided for efficiency in the event that the client program can provide a better estimate for the size of the active constraint system.
- init Specifies the fraction of the constraints in the original system which will be activated during a cold start. If this value is set to 1.0, the entire system is activated (since all variables referenced by a constraint are activated with the constraint).

addcon If addcon > 0, it specifies the maximum number of constraints which can be activated in any one occurrence of the constraint activation phase. If addcon ≤ 0, no limit is enforced. The default is a limit of 100 constraints.

addvar If addvar > 0, it specifies the maximum number of variables which can be activated in any one occurrence of the variable activation phase. If addvar ≤ 0, no limit is enforced. By default, no limit is enforced.

check The nominal interval between accuracy checks, expressed in terms of the number of pivots which actually change the basis. This is an attempt to detect the accumulation of numerical inaccuracy, and DYLP will perform a check earlier if it suspects numerical problems. By default, this is set to ½ the refactor interval factor.

coldvars If the number of active variables in the constraint system on a cold start exceeds coldvars, DYLP will attempt to deactivate variables before beginning simplex iterations. The default value is 5000 variables.

degen If set to FALSE, this option prevents the use of the perturbation-based anti-degeneracy algorithm described in §10.6. This option is TRUE by default.

degenlite This option specifies the tie-breaking strategy used for choosing between candidates with equal deltas when selecting the leaving primal or dual variable, as described in §5. The default strategy is pivot. The options are:

[4] One can easily argue that DYLP has far more options than necessary. If you look through the code, you may notice a few that aren't documented here. By and large, this is because the appropriate action is clear and choices other than the current default give uniformly poor performance.

pivotabort — Break ties using the magnitude of the pivot coefficient, and abort the search at the first basic variable which gives a delta of zero.

pivot — Break ties using the magnitude of the pivot coefficient, scanning all basic variables.

alignobj — Break ties by choosing the leaving variable which will make tight the hyperplane most closely aligned with the normal of the objective function (*i.e.*, the normal most nearly lies in the hyperplane).

alignedge — Break ties by choosing the leaving variable which will make tight the hyperplane most closely aligned with the direction of motion specfied by the entering variable (*i.e.*, the edge most nearly lies in the hyperplane).

perpobj — Break ties by choosing the leaving variable which will make tight the hyperplane most nearly perpendicular to the normal of the objective function (*i.e.*, the hyperplane most nearly blocks motion in the direction of the normal of the objective)

perpedge — Break ties by choosing the leaving variable which will make tight the hyperplane most nearly perpendicular to the direction of motion specified by the entering variable.

degenpivlim This option specifies the number of consecutive degenerate pivots which are required to trigger the perturbation-based anti-degneracy algorithm. Defaults to 5 consecutive degenerate pivots.

dualadd This option controls the amount of effort that DYLP will expend attempting to add variables to bound an unbounded dual problem.

0 Variable activation is not attempted.

1 Type 1 variables are activated. These are variables which could potentially bound the dual problem and which will be dual feasible if activated and placed in the nonbasic partition. Multiple variables of this type can be activated simultaneously.

2 Type 2 variables will be activated when there are no more type 1 variables. Type 2 variables are variables which would be dual infeasible if placed in the nonbasic partition, but which can be activated and immediately pivoted into the basis to regain dual feasibility. Only one variable of this type can be activated at a time, so this level is computationally expensive.

3 Type 3 variables will be activated when there are no more type 1 or type 2 variables. Type 3 variables are variables which can be activated and placed in the nonbasic partition with a bound-to-bound pivot. The user should think very carefully about using this level. It seemed like an interesting idea, but the value in practice is questionable.

If the limits placed on dual variable activation do not allow the dual to be bounded DYLP will revert to primal simplex.

factor The nominal interval for refactoring the basis, in terms of the number of pivots which actually change the basis. (Put another way, factor limits the total number of eta matrices in the multiplicative representation of the basis.) This is an attempt to control the accumulation of numerical inaccuracy, and DYLP will refactor earlier if it suspects numerical problems. The default refactor interval is 20 pivots.

forcecold When set to TRUE, this option will force DYLP to perform a cold start. forcecold dominates forcewarm.

forcewarm When set to TRUE, this option will force DYLP to perform a warm start.

groom Specifies the action taken when DYLP detects a nontrivial change in the status of a variable following refactoring. The working assumption is that refactoring the basis removed accumulated numerical inaccuracy, and that this inaccuracy caused the status to be set incorrectly. The default behaviour is to issue a warning message. Other options are to remain silent, or issue an error message and abort.

idlelim The limit on the number of pivots allowed without an improvement in the value of the objective function. A pivot in which the change in the objective function value is less than dy_tols.dchk is defined to be an idle pivot. Too many consecutive idle pivots are an indication that the LP has stalled and may be cycling. The precise action taken depends on the simplex phase.

Because the dual simplex algorithm does not implement the perturbation-based antidegeneracy algorithm used in the primal simplex, the dual simplex will abort if the number of pivots without change in the objective exceeds $.8 * \mathsf{idlelim}$ and DYLP will revert to the primal simplex algorithm.

During primal simplex, if the number of pivots without change in the objective exceeds iterlim, DYLP aborts and returns lpSTALLED.

iterlim The pivot limit for each occurrence of a simplex phase (primal phases I and II and dual phase II). By default, it is set to $3 * \mathsf{concnt} \leq \mathsf{MAXINT}/4$. The overall pivot limit (cumulative over all occurrences of all phases) is $3 * \mathsf{iterlim}$. If either the per phase or total limit is exceeded, DYLP terminates the problem and returns lpITERLIM. If $\mathsf{iterlim} \leq 0$ is specified, no limit is enforced.

patch If set to FALSE, DYLP is forbidden from patching a singular basis. By default, this option is TRUE.

print The print options control the amount of output which DYLP produces as it runs. This can be varied from absolutely nothing to copious output useful only during detailed debugging. Printing options are covered in detail in §18, which describes debugging options and capabilities. If DYLP is compiled with the compile-time constant NDEBUG defined, virtually all informational printing is removed.

purgeconlvl Specifies the constraint deactivation strategy. There are three levels:

normal Deactivate only inequalities which are strictly loose (*i.e.*, the associated slack is basic and not at bound).

aggressive Deactivate loose inequalities and tight inequalities whose associated dual variable is zero. (The default strategy.)

fanatic Deactivate loose inequalities and any tight constraint (inequality or equality) whose associated dual variable is zero.

scan Specifies the number of columns which are scanned to select a new candidate entering variable when the variable selected during reduced cost updating is rejected. If not supplied b the user, it defaults to $200 \leq \mathsf{archvcnt}/2 \leq 1000$.

usedual When set to TRUE, this option instructs DYLP to use the dual simplex when reoptimising after adding constraints, or when provided with a dual feasible initial basis. By default, DYLP will use dual simplex when possible. For some problems, however, the dual simplex implementation simply does not converge to an optimal solution, and in these instances it is helpful to forbid the use of dual simplex.

16 DYLP Tolerances

DYLP has a number of numeric tolerances and related control information which are used in equality and accuracy checks and associated algorithms which attempt to control the accumulation of numerical accuracy. Each is described briefly below; again, the reader is encouraged to consult dylp.h for details.

bogus The 'bogus number' tolerance. Values such that $\mathsf{zero} < |x| \le \mathsf{zero} * \mathsf{bogus}$ are considered likely to be the result of accumulated numerical inaccuracy, rather than legitimate values. Pivot coefficients and primal or dual variable values within this range will trigger refactoring of the basis. The default value is 1.0×10^3. If the problem scaling is such that numbers close to the zero tolerance are legitimate, reducing this value will often increase performance by reducing the number of times the basis is refactored. (At the risk of less frequent but more severe corrections to regain accuracy.)

cost The zero tolerance for checks involving the objective function coefficients, reduced costs, and related values. By default, set to the same value as zero, but provided for cases where the scaling of objective function coefficients is significantly different from the scaling of constraint coefficients.

dchk The dual accuracy check tolerance, as described in §7. The default value is 1.0×10^{-6}.

dfeas The dual feasibility check tolerance, scaled from cost as described in §7.

dfeas_scale Decoupling multiplier for scaling dfeas. Useful on occasion when the default scaling formula isn't quite adequate. Defaults to 1.0.

inf Infinity. By default, set to IEEE 754 infinity. Provided as a tolerance for compatibility with older code (such as LA05) which prefers that $\infty - \infty = 0$ whereas IEEE 754 specifies that $\infty - \infty = \infty$. DYLP is very careful not to pass IEEE 754 infinity to LA05.

pchk The primal accuracy check tolerance, as described in §7. The default value is 1.0×10^{-6}.

pfeas The primal feasibility check tolerance, scaled from zero as described in §7.

pfeas_scale Decoupling multiplier for scaling pfeas. Useful on occasion when the default scaling formula isn't quite adequate. Defaults to 1.0.

pivot The pivot selection multiplier. A pivot coefficient $\overline{a}_{ij}$ will be accepted as numerically stable in the primal algorithm if $|\overline{a}_{ij}| \ge \mathsf{pivot} * \mathsf{la05.stable} * \|\overline{a}_j\|_1$, where la05.stable is the stable pivot tolerance for partial pivoting in LA05. In the dual algorithm, the 1-norm is calculated over the row $\overline{a}_i$. The default value is 1.0×10^{-4}.

purge The required percentage change in the value of the objective function before constraint or variable purging is allowed. The default value is 1.0×10^{-4}.

purgevar Used to calculate the variable deactivation threshold as a percentage of the maximum unfavourable reduced costs, as described in §12.3. The default value is .5.

reframe The percentage error in the updated column or row norms which is required to trigger a reset of the PSE or DSE reference frame, respectively. The default value is .1.

zero The zero tolerance. Values smaller than |zero| are set to a clean floating-point zero. By default, set to 1.0×10^{-11}.

17 DYLP Statistics

DYLP will collect detailed statistics if the compile-time constant DYLP_STATISTICS is defined during compilation. The statistics types are described briefly in the paragraphs which follow; for details on subfields, consult dylp.h. It is expected that this capability will expand significantly in future versions of DYLP.

angle Statistics on the angles of inequality constraints to the objective function. For constraint i, this is calculated as $\frac{180}{\pi} \cos^{-1} \frac{a_i c}{\|a_i\| \|c\|}$. The maximum and minimum angle is recorded, and a histogram in 20° increments.

factor Statistics about basis factoring. The maximum and average number of pivots between refactoring is recorded, as well as the total number of times the basis is refactored.

infeas Statistics on the resolution of infeasibility during primal phase I. The maximum number of infeasible variables is recorded, as well as the total pivots in phase I and the maximum number of pivots with no change in the number of infeasible variables. DYLP also counts the number of times that the number of infeasible variables changed without requiring recalculation of the reduced costs, and the number of times when it did. Specifically, if exactly one variable gains feasibility, and it leaves the basis as it does so, the reduced costs do not have to be recalculated. This statistic is collected to see how much extra effort (beyond normal updating with each pivot) is required to maintain the vector of reduced costs during phase I.

degen Statistics on the amount of time spent in restricted subproblems trying to escape degeneracy. For each level (*i.e.*, each nested level of restricted subproblem), DYLP records the number of times this level was reached, the maximum and average number of variables involved in the degeneracy, and the total, maximum, and average number of pivots spent at this level. This item is sized (by compile time constant) to accommodate a maximum of 25 levels; in practice 3 is the maximum observed.

18 DYLP Debugging Features

DYLP incorporates two types of debugging features: a controllable printing facility and paranoid checks. The printing facility is enabled when the compile time constant NDEBUG is not defined, and is intended to allow the generation of log information at whatever level of detail is desired by the user. The paranoid checks are enabled by defining the compile time constant PARANOIA and are intended to provide significant (and expensive) cross-checks during code development.

18.1 Printing

The amount of output generated by DYLP can be varied from next to nothing to a level of detail intended only for detailed debugging. The paragraphs which follow briefly outline the capabilities; for specific output at a given print level, please refer to the file dylp.h.

conmgmt Prints information on the management of constraints, including activation and deactivation, changes to primal and dual variables, and (at the highest level) a running commentary on all constraint and variable additions, deletions, and motions attributable to activation and deactivation of constraints.

crash Prints information regarding the generation of the initial basis, including factoring, the initial set of basic variables, and their values. For a cold start, information on the selection of the basic variables can be printed.

degen Prints information about degenerate pivots and restricted subproblem formation to deal with degeneracy.

dual Prints information about the execution of the dual simplex, with capabilities similar to phase1.

la05 Prints information about the activities of LA05 as it performs basis pivoting and factoring, including handling of singular bases, numerical instability, changes in pivot tolerances, and (at the highest level) dumps of the actual coefficients of L^{-1} and U.

major Tracks the major state transitions of the dynamic simplex algorithm as DYLP solves an LP.

phase1 Prints information about the execution of phase I of the primal simplex. At the low end, messages are printed for extraordinary events — unboundedness, serious pivoting problems, *etc.* At a medium level, a one line message is printed summarising each pivot, as well as messages about routine but infrequent events — refactoring, accuracy checks, and various minor problems. At the highest level, all primal and dual variables are printed as they are recalculated for each pivot, along with detailed information about reduction of infeasibility and changes to the phase I objective function. This is *a lot* of output.

phase2 Prints information about the execution of phase II of the primal simplex, with capabilities similar to phase1.

pivoting Prints information on the evaluation of candidates for the leaving primal or dual variable and details of the pivot column or row. At the highest level, produces *a lot* of output.

pricing Prints information regarding the pricing of candidates for the entering primal or dual variable. At the highest level, produces *a lot* of output.

setup Prints information regarding the loading and initialisation of an LP problem, including the constraints and variables which are activated, and the angle of inequalities to the objective function.

varmgmt Prints information on the activation and deactivation of variables, much as conmgmt.

18.2 Paranoia

Because it is intended as a development code, DYLP incorporates a large number of sanity checks, enabled by defining the compile time constant PARANOIA. Many of these tests are cheap and simple — checks for null parameters, sensible constraint and variable counts, proper major phase, and range checks on indices. Others are more elaborate and expensive, including two dedicated subroutines:

- dy_chkstatus implements extensive checks to make sure that the status and value of a primal variable agree across multiple data structures and are appropriate for the current major phase.
- dy_chkdysys implements extensive checks to ensure that the active constraint system and associated data structures are correct and consistent.

Two particular checks, which check the accuracy of iterative updating for PSE column norms and DSE row norms, are sufficiently expensive that they are controlled separately by the compile-time constants CHECK_PSE_UPDATES and CHECK_DSE_UPDATES. They calculate the norms directly for comparison with the updated values, and the computational expense is unacceptable unless there is specific reason to suspect an error.

References

[1] R. Bixby, S. Ceria, C. McZeal, and M. Savelsbergh. An Updated Mixed Integer Programming Library: MIPLIB 3.0. Available at http://www.caam.rice.edu/~bixby/miplib/miplib.html, January 1996.

[2] J. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.

[3] L. Hafer. A Note on the Relationship of Primal and Dual Simplex. Technical Report SFU-CMPT TR 1998-21, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, December 1998.

[4] L. Hafer. CONSYS: a dynamic constraint system. Technical Report SFU-CMPT TR 1998-22, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, December 1998.

[5] M. Padberg. *Linear Optimization and Extensions*, volume 12 of *Algorithms and Combinatorics*. Springer-Verlag, New York, 1995.

[6] J. Reid. A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases. *Mathematical Programming*, 24(1):55–69, September 1982.

[7] D. Ryan and M. Osborne. On the Solution of Highly Degenerate Linear Programmes. *Mathematical Programming*, 41:385–392, 1988.

[8] U. Suhl and L. Suhl. Computing Sparse LU Factorisations for Large-Scale Linear Programming Bases. *ORSA Journal on Computing*, 2(4):325–335, Fall 1990.

[9] Z. Zlatev. On Some Pivotal Strategies in Gaussian Elimination by Sparse Technique. *SIAM Journal on Numerical Analysis*, 17(1):18–30, February 1980.