AT&T Bell Laboratories
Murray Hill, NJ 07974

Computing Science Technical Report No. 136

Pictures of Karmarkar's Linear Programming Algorithm

David M. Gay

January 1987

Pictures of Karmarkar's Linear Programming Algorithm

David M. Gay

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Karmarkar's linear programming algorithm handles inequality constraints by changing variables to make all constraints about equally distant; it moves in the steepest-descent direction seen by the new variables. This paper summarizes four variants of Karmarkar's linear programming algorithm (primal affine, primal projective, dual affine, and dual projective), discusses depicting polytopes (feasible regions), and presents pictures illustrating the latter three variants. These pictures give an algorithm's eye view of the variable changes and provide visual verification of some theoretical results.

Introduction

This paper shows an algorithm's eye view of several variants of Karmarkar's linear programming problem. All use changes of variables to bring the current iterate to the center of the feasible region and move in the steepest-descent direction seen by the changed variables. By considering problems having three degrees of freedom, we can depict the feasible regions before and after the variable changes, thus graphically illustrating their effect.

We begin by summarizing four variants of Karmarkar's linear programming algorithm: primal affine, primal projective, dual affine, and dual projective; the last is apparently new. Then we discuss depicting polytopes, i.e., the feasible regions of the linear programming problems, and present pictures that illustrate the dual affine, dual projective, and primal projective algorithms. (For irrelevant reasons, these three variants happened to be the most convenient to illustrate.)

The choice of variables matters. With the right choice, used in most of our pictures, the views we obtain are unique up to rotations, and they illustrate some properties of the projective algorithms. Unfortunately, the pictures in [9] are from a seemingly natural but ''wrong'' choice of variables. We discuss these variables at the end and show how differently things look from their perspective.

The norms $\|\cdot\|$ used below are Euclidean. As usual, I denotes an identity matrix, $\mathbf{0}$ a matrix or vector of zeros, e_k denotes the kth standard unit vector, and $e = \sum_k e_k$ denotes a vector of ones, all of dimension dictated by context. Other notation is defined as needed below.

The Problem

Linear programming (LP) problems can always be expressed in standard form: Find a vector $x \in \mathbb{R}^n$ (of length n) to

$$\text{(1a)} \qquad \text{minimize} \quad c^{\mathrm{T}} x$$

subject to

$$\text{(1b)} \qquad Ax = b$$

and

$$\text{(1c)} \qquad x \geq \mathbf{0},$$

where $A \in \mathbb{R}^{m \times n}$ is a given (real $m \times n$) matrix, $b \in \mathbb{R}^m$, and superscript T means ''transposed''.

(Superscript T will be omitted when vectors are spelled componentwise.) Associated with (1) is the *dual problem*

(2a) $$\text{maximize} \quad b^{\mathrm{T}} y$$

subject to

(2b) $$A^{\mathrm{T}} y \le c$$

(with no sign constraints on y). For simplicity, we assume A has full rank (m).

The hard part about (1) is determining which components of x can be strictly positive in solutions to (1). (This is equivalent to determining which components of (2b) can only be satisfied as equalities in solutions to (2).) Active set algorithms, such as the simplex method, repeatedly update an estimate of a set of x components that may be positive; in effect, they only allow a few (at most m) components of x to be positive at one time. Karmarkar's algorithm, on the other hand, works with a solution estimate that has all components positive. It repeatedly changes variables, computes the steepest descent direction in the changed variables, moves some distance in that direction, and translates the resulting point back to the original variables.

Karmarkar's paper [15] uses projective transformations to make its changes of variables. One can also use a linear change of variables; the resulting ''affine variant'' of Karmarkar's algorithm (references for which are given below) is a bit easier to describe than the ''projective variant''. Although the affine variant is appealing for its simplicity, stronger convergence results are known for the projective variant.

Affine Variant

In the affine variant of Karmarkar's LP algorithm, we proceed from iterate x^k to x^{k+1} as follows. We start with the change of variables $\hat{x} = D^{-1} x$, where

(3) $$D = D^k = \mathrm{diag}(x^k)$$

is the diagonal matrix whose (i,i) entry is x_i^k, the ith component of the current iterate x^k. The constraints (1b,c) become $\hat{A}\hat{x} = b$ and $\hat{x} \ge \mathbf{0}$, where $\hat{A} = AD$, and the objective (1a) becomes $\hat{c}^{\mathrm{T}} \hat{x}$, where $\hat{c} = Dc$. Note that the current iterate x^k transforms into a vector of ones (denoted e): $\hat{x}^k := D^{-1} x^k = e$. The (constrained) steepest descent direction is then

(4) $$\hat{p} = -P_{\hat{A}} \hat{c},$$

where P_B denotes projection onto the orthogonal complement of the row space of a matrix B. A step in this direction to $\hat{x}^{k+1} = \hat{x}^k + \hat{\alpha}\hat{p}$ translates into the step $x^{k+1} = x^k + \alpha p = x^k - \alpha D P_{AD} Dc$ in the original variables, where $\alpha = \hat{\alpha} > 0$ is small enough that $x^{k+1} > 0$ (i.e., that all components of x remain positive). People often choose α in the range $[0.9\alpha^{max},\ 0.99\alpha^{\max}]$, where $\alpha^{\max} = \max\{\tau > 0 : x^k + \tau p \ge \mathbf{0}\}$.

I first learned about the primal affine variant from Karmarkar in late 1984. Several other people or groups appear to have proposed this variant independently, including [3, 5, 7, 25].

Projective Variant

In [15], Karmarkar assumed the optimal value of $c^{\mathrm{T}} x*$ to be zero, that $b_i = 0$ for $i < m$ and that $A_{m,j} = 1$ for all j; he showed that one can always transform a problem to this form. There are various ways to make this transformation or relax these assumptions; see, e.g., [2, 8, 13, 14, 16, 19, 21, 22, 24], and the Concluding Remarks below. For making pictures it seems most convenient to do things as in [10].

This version of the algorithm computes a sequence of pairs $(x^k,\ \beta^k)$ that maintain

(5a) $$Ax^k = b,$$

(5b) $$x^k > \mathbf{0},$$

and

(5c) $$\beta^k \ge -c^{\mathrm{T}} x*,$$

where $c^{\mathrm{T}} x*$ is the optimal objective function value (1a), and that always reduce the ''potential function''

(6) $$\phi(x,\beta) := (n+1)\log(c^{\mathrm{T}}x + \beta) - \sum_{i=1}^{n} \log(x_i)$$

by at least a constant amount: $\phi(x^{k+1},\beta^{k+1}) \le \phi(x^k,\beta^k) - \frac{1}{5}$. This forces $\lim_{k\to\infty} \phi(x^k,\beta^k) = -\infty$; if the feasible region is bounded, this forces $-\lim_{k\to\infty} \beta^k = \lim_{k\to\infty} c^{\mathrm{T}}x^k = c^{\mathrm{T}}x^*$. Later we discuss obtaining x^0 and β^0; for now assume them known.

Given iterate (x^k,β^k), we define D, as before, by (3). This time we use the projective change of variables

$$\hat{x} = \psi(x) := (e^{\mathrm{T}}D^{-1}x + 1)^{-1} \begin{bmatrix} D^{-1}x \\ 1 \end{bmatrix},$$

whose inverse is

$$x = \psi^{-1}(\hat{x}) = \hat{x}_{n+1}^{-1} D\hat{x}_{1..n},$$

where $\hat{x}_{1..n}$ stands for the first n components of $\hat{x}$. The constraints (1b,c) become

(7a) $$\begin{bmatrix} AD & -b \\ e^{\mathrm{T}} & 1 \end{bmatrix} \hat{x} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$$

(7b) $$\hat{x} \ge \mathbf{0},$$

and ϕ becomes

$$\hat{\phi}(\hat{x},\beta) = \sum_{i=1}^{n+1} \log\left[\frac{c^{\mathrm{T}}D\hat{x}_{1..n} + \beta\hat{x}_{n+1}}{\hat{x}_i}\right] - \log(\det(D)).$$

Let $\bar{A} := (AD, -b)$,

(8a) $$u := P_{\bar{A}} \begin{bmatrix} Dc \\ 0 \end{bmatrix},$$

(8b) $$v := P_{\bar{A}} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix},$$

and

(9) $$\mu(\beta) := \min\{(u + \beta v)_i : 1 \le i \le n+1\}.$$

The convergence theory requires β^{k+1} to satisfy both (5c) and $\mu(\beta^{k+1}) \le 0$. Either $\mu(\beta^k) \le 0$, in which case $\beta^{k+1} := \beta^k$ suffices, or else we can choose $\beta^{k+1} < \beta^k$ with $\mu(\beta^{k+1}) = 0$, in which case duality theory implies $\beta^{k+1} \ge -c^{\mathrm{T}}x^*$: see [10] for details. The steepest-descent direction for $\hat{\phi}$ in the $\hat{x}$ variables is

$$\hat{p} = -P_B \begin{bmatrix} Dc \\ \beta^{k+1} \end{bmatrix} = -(I - \frac{ee^{\mathrm{T}}}{n+1})(u + \beta^{k+1}v),$$

where $B = \begin{bmatrix} AD & -b \\ e^{\mathrm{T}} & 1 \end{bmatrix}$. The step $\hat{x}^{k+1} = \hat{x}^k + \hat{\alpha}\hat{p}$ translates back to $x^{k+1} = x^k + \alpha^k p^k$, where

(10) $$p^k = D\hat{p}_{1..n} - \hat{p}_{n+1}x^k$$

and $\alpha^k = \dfrac{\hat{\alpha}}{(n+1)^{-1} + \hat{\alpha}\hat{p}_{n+1}}$. Although a fixed choice of $\hat{\alpha}$, e.g. $\hat{\alpha} = 1/3$, suffices for the convergence theory, in practice one gets much faster convergence by choosing $\alpha = \alpha^k$ to minimize $\phi(x^k + \alpha p^k)$ approximately.

Dual Affine Algorithm

The algebra works out nicely when we apply the above algorithms to the dual problem (2). Moreover, the dual affine variant works even when there are considerable errors in the projections. This variant appears to have been proposed independently in [1] and [20].

To derive the dual algorithms, it is useful to convert (2) to standard form by introducing slack variables: we define $s \in \mathbb{R}^n$ by $s = c - A^{\mathrm{T}} y$. Although we shall compute with y, it is useful to eliminate y in deriving the dual algorithm. To do this, let A^+ be any generalized inverse of A, i.e., any matrix with $A = AA^+ A$. (The choice of A^+ is irrelevant, since A^+ will not appear in the final formulae. Since we're assuming A to have full rank, in fact $AA^+ = I$.) Then $y = A^{+\mathrm{T}}(c - s)$, so maximizing $b^{\mathrm{T}} y$ is the same as minimizing $b^{\mathrm{T}} A^{+\mathrm{T}}(s - c)$. Thus (2) is equivalent to

(11a) $$\text{minimize} \quad (A^+ b)^{\mathrm{T}} s - c^{\mathrm{T}} A^+ b$$

subject to

(11b) $$(I - A^{\mathrm{T}} A^{+\mathrm{T}}) s = (I - A^{\mathrm{T}} A^{+\mathrm{T}}) c$$

and

(11c) $$s \geq \mathbf{0},$$

where I denotes the identity matrix of appropriate dimension (i.e., $n \times n$).

The affine algorithm requires us to compute step direction (4), with $\hat{A}$ from (11b): $\hat{A} := (I - A^{\mathrm{T}} A^{+\mathrm{T}}) D$, where $D = \mathrm{diag}(s^k)$ and s^k is the current iterate. Fortunately, the rows of AD^{-1} span the null space of this constraint matrix: $(I - A^{\mathrm{T}} A^{+\mathrm{T}}) A^{\mathrm{T}} = \mathbf{0}$. Thus $P_{\hat{A}} = D^{-1} A^{\mathrm{T}} (AD^{-2} A^{\mathrm{T}})^{-1} AD^{-1}$. The constant $c^{\mathrm{T}} A^+ b$ in (11a) is irrelevant to the affine algorithm, so the $\hat{c}$ of (4) is $DA^+ b$; since $AA^+ = I$, the search direction $p = D\hat{p}$ boils down to $p = -A^{\mathrm{T}} (AD^{-2} A^{\mathrm{T}})^{-1} b$. But this is a search direction for s; since $s = c - A^{\mathrm{T}} y$, adding αp to s is the same as adding $\alpha (AD^{-2} A^{\mathrm{T}})^{-1} b$ to y.

In short, the dual affine variant takes the form

$$D := D^k = \mathrm{diag}(c - A^{\mathrm{T}} y^k)$$

$$y^{k+1} = y^k + \alpha^k (AD^{-2} A^{\mathrm{T}})^{-1} b,$$

with $\alpha^k > 0$ chosen to retain $c - A^{\mathrm{T}} y^{k+1} > \mathbf{0}$.

Dual Projective Algorithm

The dual projective algorithm must compute the u and v of (8) with $\bar{A} = [(I - A^{\mathrm{T}} A^{+\mathrm{T}}) D,\ -(I - A^{\mathrm{T}} A^{+\mathrm{T}}) c]$. As mentioned in [10], the columns of

$$Q = \begin{bmatrix} D^{-1} A^{\mathrm{T}} & D^{-1} c \\ \mathbf{0} & 1 \end{bmatrix}$$

span the null space of this $\bar{A}$. Thus (8) becomes $u + \beta v := Q(Q^{\mathrm{T}} Q)^{-1} Q^{\mathrm{T}} \begin{bmatrix} DA^+ b \\ \beta \end{bmatrix}$, except that this ignores constant $c^{\mathrm{T}} A^+ b$ in (11a). As shown in [10], adding a constant to the objective function effectively adds a constant to β, so (8) really becomes $u + \beta v := Q(Q^{\mathrm{T}} Q)^{-1} Q^{\mathrm{T}} \begin{bmatrix} DA^+ b \\ \beta - c^{\mathrm{T}} A^+ b \end{bmatrix}$; correspondingly, (6) becomes

(12) $$\phi(y, \beta) := (n+1) \log(\beta - b^{\mathrm{T}} y) - \sum_{i=1}^{n} \log(s_i),$$

(where $s = c - A^{\mathrm{T}} y$). But $Q^{\mathrm{T}} \begin{bmatrix} DA^+ b \\ \beta - c^{\mathrm{T}} A^+ b \end{bmatrix} = \begin{bmatrix} AA^+ b \\ \beta \end{bmatrix} = \begin{bmatrix} b \\ \beta \end{bmatrix}$, and we end up with

$$u + \beta v := Q(Q^{\mathrm{T}}Q)^{-1}\begin{bmatrix} b \\ \beta \end{bmatrix}. \tag{13}$$

It is convenient to note that

$$(Q^{\mathrm{T}}Q)^{-1}\begin{bmatrix} b \\ \beta \end{bmatrix} = \begin{bmatrix} (AD^{-2}A^{\mathrm{T}})^{-1}b \\ 0 \end{bmatrix} + \nu \begin{bmatrix} -(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c \\ 1 \end{bmatrix}, \tag{14a}$$

where

$$\nu = \frac{\beta - c^{\mathrm{T}}D^{-2}A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}b}{1 + c^{\mathrm{T}}D^{-2}c - c^{\mathrm{T}}D^{-2}A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c}, \tag{14b}$$

so that

$$u + \beta v = \begin{bmatrix} D^{-1}\left[A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}b + \nu\cdot[c - A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c]\right] \\ \nu \end{bmatrix}. \tag{14c}$$

Rather than working directly with (9), it is convenient to work with ν: if we can choose ν to make the first n components of (14c) nonnegative, then D^{-1} times these components, i.e., $x := D^{-2}[A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}b + \nu\cdot(c - A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c)]$ is feasible for (1), and $c^{\mathrm{T}}x = b^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c + \nu\cdot(c^{\mathrm{T}}D^{-2}c - c^{\mathrm{T}}D^{-2}A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c)$ is a candidate for updating β: if $c^{\mathrm{T}}x < \beta$, then we set $\beta := c^{\mathrm{T}}x$. Once β has been updated, the step direction (10) for s becomes $p = A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}(b - \nu AD^{-2}c) + \nu(c - s)$, with ν from (14b); since $c - s = A^{\mathrm{T}}y$, this boils down to the step direction $\bar{p} = (AD^{-2}A^{\mathrm{T}})^{-1}(b - \nu AD^{-2}c) + \nu y$ for y.

To summarize, the dual projective algorithm works as follows: given $\beta^k \geq c^{\mathrm{T}}x^*$ (= $b^{\mathrm{T}}y^*$, the optimal objective function value) and y^k rendering $s^k := c - A^{\mathrm{T}}y^k > \mathbf{0}$, let $D := D^k = \mathrm{diag}(s^k)$; if such exists, let ν^k be the minimal ν for which

$$A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}b + \nu(c - A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c) \geq \mathbf{0},$$

and if

$$\beta^k > c^{\mathrm{T}}D^{-2}[A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}b + \nu^k(c - A^{\mathrm{T}}(AD^{-2}A^{\mathrm{T}})^{-1}AD^{-2}c)],$$

then set β^{k+1} to this value; otherwise let $\beta^{k+1} := \beta^k$, and determine ν^k from (14b) (with $\beta = \beta^k$). Compute $\bar{p}^k := (AD^{-2}A^{\mathrm{T}})^{-1}(b - \nu^k AD^{-2}c) + \nu^k y^k$, choose α^k to be a value of α that approximately minimizes $\phi(y^k + \alpha\bar{p}^k, \beta^{k+1})$, and set $y^{k+1} := y^k + \alpha^k\bar{p}^k$.

The theory behind the projective algorithms described above assumes exact arithmetic. It should be possible to use the ideas in [13, 14, 19] to develop variants of the projective algorithms that do not require exact arithmetic. However, if one computes exact factorizations to do the necessary linear algebra (or runs some iterative method, e.g., some flavor of conjugate gradients, long enough), then it seems reasonable to consider running the algorithms described above. In this regard, the denominator in (14b) deserves brief discussion: it has the form $1 + \tilde{c}^{\mathrm{T}}B\tilde{c}$, where B is a projection and hence is positive semi-definite. If we have a Cholesky factor L of $AD^{-2}A^{\mathrm{T}} = LL^{\mathrm{T}}$, then we can compute the denominator of (14b) as $1 + (\|D^{-1}c\|_2 + \|L^{-1}AD^{-2}c\|_2)(\|D^{-1}c\|_2 - \|L^{-1}AD^{-2}c\|_2)$; if ever the computed value of $\|D^{-1}c\|_2$ is less than that of $\|L^{-1}AD^{-2}c\|_2$, the computed quantities are probably too inaccurate to justify continuing the dual projective algorithm, and it seems reasonable to switch to the affine algorithm.

Bounds

Many problems have simple bounds on some of the variables: in addition to (1c), one has the upper-bound constraints $x \leq u$. Of course, we can express these constraints in the standard form (1b,c) by adding slack variables; this effectively replaces A by $\tilde{A} = \begin{bmatrix} A & \mathbf{0} \\ I & I \end{bmatrix}$ and D by $\tilde{D} = \begin{bmatrix} D_l & \mathbf{0} \\ \mathbf{0} & D_u \end{bmatrix}$, where $(D_l)_{ii} = x_i$ and $(D_u)_{ii} = u_i - x_i$. At first glance this looks undesirable, as it increases the size of the linear equation systems that must be solved. However, by factoring $\tilde{A}\tilde{D}^{-2}\tilde{A}^{\mathrm{T}}$ appropriately, we can reduce the linear equations to systems of the same size and sparsity as without upper bounds. Specifically,

(15)
$$\tilde{A}\tilde{D}^{-2}\tilde{A}^{\mathrm{T}} =$$
$$\begin{bmatrix} I & AD_l^{-2}(D_l^{-2}+D_u^{-2})^{-1} \\ \mathbf{0} & I \end{bmatrix} \begin{bmatrix} AD_l^{-2}D_u^{-2}(D_l^{-2}+D_u^{-2})^{-1}A^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & D_l^{-2}+D_u^{-2} \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ (D_l^{-2}+D_u^{-2})^{-1}D_l^{-2}A^{\mathrm{T}} & I \end{bmatrix},$$

so we end up factoring $AD_l^{-2}D_u^{-2}(D_l^{-2}+D_u^{-2})^{-1}A^{\mathrm{T}}$ rather than $AD^{-2}A^{\mathrm{T}}$. (Note that $D_l^{-2}D_u^{-2}(D_l^{-2}+D_u^{-2})^{-1}$ is a diagonal matrix.) For other discussions of simple bounds, see §3 of [6], §2.4 of [11], §5.1 of [23], §6 of [25].

The primal projective algorithm depicted below imposes simple bounds of 10^4 on all variables, using (15). This is an ingredient in the *Record* pictures [9] described at the end. (The dual algorithms depicted below also use (15), but they can conveniently ignore simple bounds of $+\infty$.)

Pictures of Polytopes

Any x value that satisfies (1b) and (1c) is called a *feasible solution* to the LP problem (1). The set of all such x values is a *polytope*, i.e., an intersection of finitely many half-spaces. Similarly, any y that satisfies (2b) is a feasible solution to the LP problem (2), and the set of all such y values is a polytope.

If A or A^{T} has the right shape — two or three more columns than rows — then we can depict the polytope of feasible solutions graphically, as we shall now see. The cases $m = 2$ and $n = m + 2$ (two rows or two more columns than rows) can be misleading, since some things that are true in two dimensions do not carry over to higher dimensions. Moreover, the cases $m = 3$ and $n = m + 3$ lead to pictures that have greater visual appeal. Therefore we shall concentrate on the latter cases.

If $m = 3$, then for $F := A^{\mathrm{T}}$ and $f := c$ the feasible region for (2) has the form

(16)
$$\{z \in \mathbb{R}^3 \colon Fz \le f\}.$$

It is easy to visualize such a region. If F_i denotes the ith row of F, then each row F_i and corresponding component f_i of f define a half-space, $\{z \in \mathbb{R}^3 \colon F_i z \le f_i\}$, and (16) is the intersection of these half-spaces. To depict (16), we may simply plot the edges of (16) that are visible when we look in one or more view directions from one or more vantage points. (We discuss the choice of view direction, vantage point, and view angle later.)

On the other hand, if $n = m + 3$, then it is only slightly harder to visualize the feasible region for (1). This is slightly harder, because it requires a change of variables. Assume A has full rank, and let F by any matrix whose columns span the null space of A. By assumption, A has full rank, so F has 3 columns; $F \in \mathbb{R}^{n\times 3}$ can be any rank 3 matrix such that $AF = \mathbf{0}$. If $x^\circ \in \mathbb{R}^n$ is any feasible point for (1), then for any feasible x there is a unique $z \in \mathbb{R}^3$ such that $x = x^\circ - Fz$, and, conversely, if $z \in \mathbb{R}^3$ satisfies $Fz \le x^\circ$, then $x := x^\circ - Fz$ is feasible for (1). Thus the feasible region for (1) corresponds to (16) with $f := x^\circ$.

One complication with this latest choice of (16) is that F is not unique. Another way to say this is that, given (16), we could choose some nonsingular $G \in \mathbb{R}^{3\times 3}$ and change to the variables $w := G^{-1}z$; this amounts to replacing F by FG in (16). Such a change of variables can profoundly affect pictures of the polytope (drawn as suggested above), but it does not affect the polytope's structure. Indeed, the edges of (16) may be characterized by a set $S = S(F,f)$ of quadruples of distinct integers, such that $(i,j,k,l) \in S$ corresponds to the edge

(17)
$$\{z \in \mathbb{R}^3 \colon F_i z = f_i,\ F_j z = f_j,\ F_k z \le f_k,\ F_l z \le f_l\}.$$

S remains the same if we change F to FG, i.e., $S(FG,f) \equiv S(F,f)$.

We can make F unique, modulo rotations, by insisting that its columns be orthonormal, i.e., that $F^{\mathrm{T}}F = I$. This has the added benefit that the necessary projections then have a natural interpretation: in terms of the variables z in (16), they are just FF^{T}. (For most of the pictures shown below, F was orthogonalized by the stabilized Gram-Schmidt algorithm; for views of the original polytope, e.g., figures 2–14, and for the *Record* pictures shown at the end, F was *not* orthogonalized.)

Generating and Viewing Polytopes

To generate 3 dimensional polytopes, i.e., the feasible regions for linear programming problems (1) or (2) with $n = m + 3$ or $m = 3$, it was convenient to select tangents to an ellipsoid about the origin, and then to jiggle them in or out a little. Table 1 gives the resulting *A*, *b*, and *c* defining the dual problem (2) we shall consider below.

i	$A_{1,i}$	$A_{2,i}$	$A_{3,i}$	c_i	i	$A_{1,i}$	$A_{2,i}$	$A_{3,i}$	c_i
1	1	0	0	10	48	0.776419	1.1329	–0.891049	2.64
2	–1	0	0	10	49	0.683382	1.1012	–0.696962	2.64
3	0	1	0	10	50	0.516868	0.993245	–0.392579	2.64
4	0	–1	0	10	51	0.465984	0.952607	–0.305981	2.64
5	0	0	1	10	52	0.343589	0.844541	–0.106338	2.64
6	0	0	–1	10	53	0.19781	0.700211	0.118341	2.64
7	–0.948109	0.326798	–0.55927	2.64	54	0.0306888	0.517064	0.361075	2.64
8	–0.959071	0.457124	–0.459551	2.64	55	0.892578	1.1356	–1.03499	2.64
9	–0.920668	0.539314	–0.356824	2.63736	56	0.825649	1.13888	–0.883178	2.64
10	–0.850045	0.594019	–0.248836	2.64132	57	0.723153	1.10255	–0.685394	2.64
11	–0.726585	0.630421	–0.109757	2.64	58	0.631666	1.05054	–0.525279	2.64
12	–0.524343	0.630806	0.0683571	2.63472	59	0.514144	0.967662	–0.333092	2.64
13	–0.281476	0.582318	0.241176	2.6136	60	0.398956	0.873909	–0.155221	2.64
14	–0.277238	0.649988	–0.910721	2.64	61	0.276002	0.763148	0.0256707	2.64
15	–0.273845	0.808338	–0.760027	2.64	62	1.21797	1.02965	–1.27827	2.64
16	–0.26471	0.841367	–0.684334	2.64	63	1.1142	1.16083	–0.985681	2.64
17	–0.220614	0.867007	–0.431214	2.64	64	0.877339	1.09433	–0.624873	2.64
18	–0.183672	0.838421	–0.261174	2.64	65	0.628164	0.956773	–0.302028	2.64
19	–0.144191	0.78565	–0.0980917	2.64	66	0.148637	0.606655	0.247611	2.64
20	–0.0724596	0.652504	0.166939	2.64	67	1.33984	1.14455	–1.11092	2.64
21	–0.0341189	0.566164	0.295867	2.64	68	1.23592	1.15942	–0.93779	2.64
22	0.0403285	0.955243	–0.772093	2.6532	69	1.13625	1.14113	–0.799053	2.64
23	0.0312882	0.935653	–0.435773	2.6268	70	0.997802	1.09404	–0.62453	2.64
24	0.02042	0.817498	–0.110836	2.64	71	0.815322	1.0102	–0.41278	2.64
25	0.0934504	0.977023	–0.589586	2.6664	72	0.586217	0.882642	–0.165629	2.64
26	0.0695163	0.900807	–0.292568	2.64	73	0.335572	0.722656	0.0876082	2.64
27	0.0356949	0.72303	0.0683491	2.63208	74	0.0725957	0.536115	0.337626	2.64
28	0.434916	0.898464	–1.15598	2.64	75	1.64933	1.08863	–1.12758	2.64
29	0.421701	1.03648	–0.982135	2.64	76	1.43556	1.12276	–0.834403	2.63736
30	0.401872	1.05712	–0.877737	2.64	77	1.0079	0.998721	–0.409256	2.6532
31	0.348666	1.04371	–0.655343	2.64	78	0.348205	0.68317	0.142344	2.63208
32	0.297745	0.995979	–0.471783	2.64	79	1.53321	1.13218	–1.01482	2.64
33	0.231351	0.908603	–0.253538	2.64	80	1.39856	1.13052	–0.843658	2.63472
34	0.127643	0.736479	0.0574528	2.64	81	1.21252	1.08733	–0.641492	2.64264
35	0.0380502	0.561958	0.304447	2.64	82	0.689548	0.871521	–0.152374	2.64
36	0.733312	1.11609	–0.995647	2.64	83	1.89625	0.970384	–1.07123	2.64
37	0.695319	1.12071	–0.891662	2.64	84	1.83863	1.02483	–0.968253	2.64
38	0.64696	1.10863	–0.774384	2.64	85	1.64847	1.04786	–0.759851	2.64
39	0.597801	1.0851	–0.664601	2.64	86	1.34597	0.99948	–0.499665	2.64
40	0.532214	1.04265	–0.527407	2.64	87	0.951747	0.882985	–0.205429	2.64
41	0.447604	0.975156	–0.361111	2.64	88	0.492661	0.706316	0.102808	2.64
42	0.366364	0.900473	–0.209721	2.64	89	2.01458	0.842735	–0.834525	2.63736
43	0.292333	0.825726	–0.077384	2.64	90	1.854	0.902107	–0.661818	2.64264
44	0.201976	0.72715	0.0779788	2.64	91	1.56706	0.900002	–0.44401	2.64
45	0.860133	0.904507	–1.28126	2.64	92	1.17334	0.836564	–0.195943	2.64528
46	0.879349	1.02706	–1.22401	2.64	93	0.699656	0.716115	0.0654775	2.63472
47	0.826318	1.12981	–1.012	2.64	94	–7.13499e–16	0.481514	0.404038	2.63208

Table 1: *A* and *c* for (2); $b = (3.305617,\ 2.881186,\ -1.11435)$.

Figure 1 depicts some details of drawing a two-dimensional picture of a three-dimensional polytope. Before drawing the polytope, we must choose a viewpoint *v*, view direction *d*, and angle of view α. We then imagine a view screen orthogonal to the direction of view and between the viewpoint and the polytope. (Because of how things scale, the exact position of the view screen is immaterial.) The view angle defines a view cone whose vertex is at the viewpoint *v* and whose center lies along the view direction *d*; we will only be able to see parts of the polytope that fall within this cone. [We could let the cone have a circular cross section, so that its intersection with the view screen would be a circle, but it is more convenient if this intersection is rectangular. For simplicity we shall insist that this intersection be square, though allowing separate horizontal and vertical angles of view would complicate matters little.] To draw the polytope, we project its visible edges onto the view screen; ''visibility'' is the subject of the next paragraph. To project an edge, we project the edge's endpoints and draw the segment between their images on the view screen. Thus in figure 1, the segment from x' to y' on the view screen is the image of the polytope edge from x to y. Finally, we scale the points on the image screen so that those on the edges of the view cone's intersection with the view screen are at the edge of our picture.

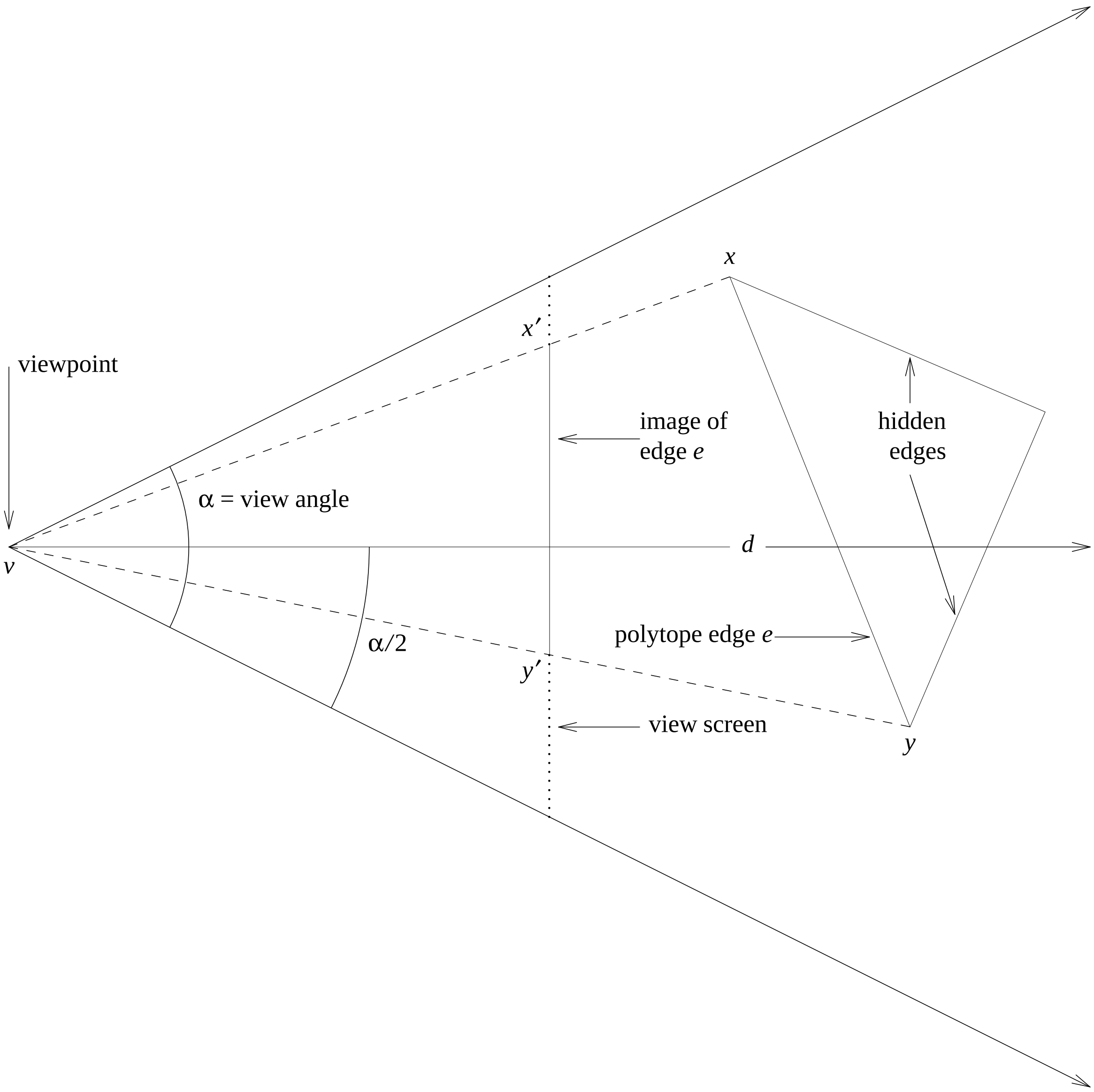

Figure 1

Consider now how to tell what portions of which polytope edges are visible. Because we are using a rectangular view cone, telling this is easy: we simply adjoin four view-cone constraints to (16) to obtain a modified polytope (16′); this reduces the visibility problem to that of deciding whether or not each segment (17) of (16′) is visible. If the viewpoint v is strictly outside the original polytope (16), then edge (17) of (16′) is visible only if v satisfies $F_i v \geq f_i$ or $F_j v \geq f_j$ (or both). And if v is inside (16), then all edges of the modified polytope (16′) are visible.

The view angle can strongly affect a polytope's appearance. Figures 2–5, for example, show four views from outside of the same polytope, with the same view direction but different angles of view (30°, 60°, 90°, and 120° respectively). The view point in each case was chosen to make the larger of the image's width and height be 85% of the viewing window's width and height. (Aside from details about focusing, the effect shown in figures 2–5 is that of decreasing the focal length of a camera's lens.)

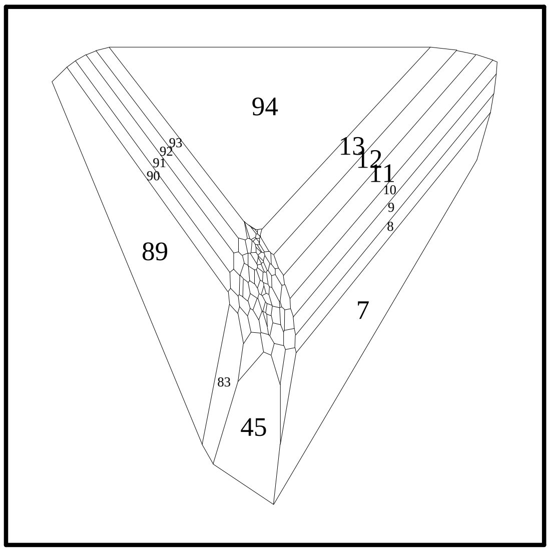

Figure 2: view angle 30°

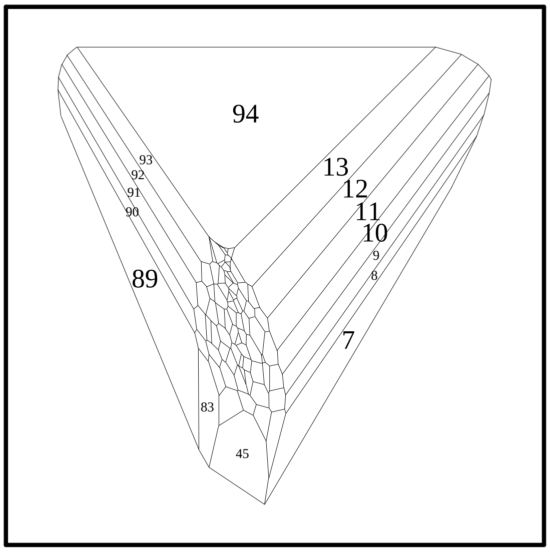

Figure 3: view angle 60°

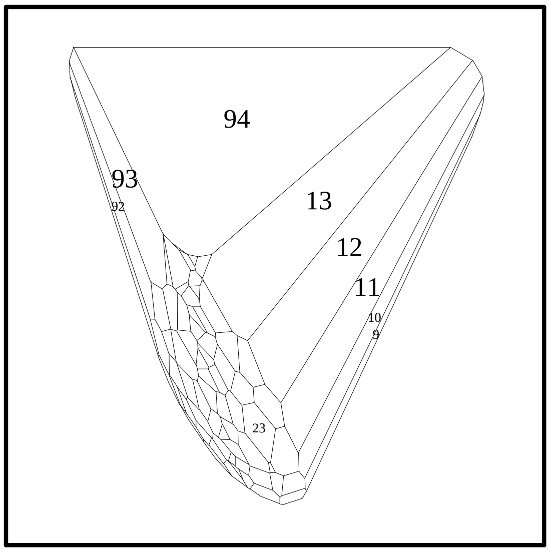

Figure 4: view angle 90°

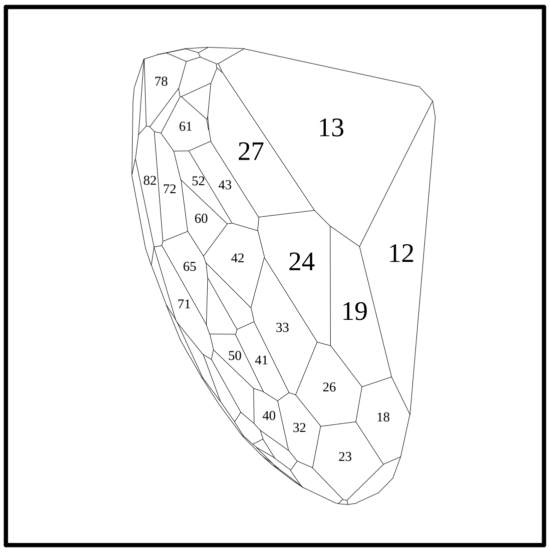

Figure 5: view angle 120°

I played some with the polytope of figures 2–5, looking for interesting viewpoints and view directions. Figure 6 shows my favorite perspective, one that my wife and I both liked because it looks something like a dinosaur's head.

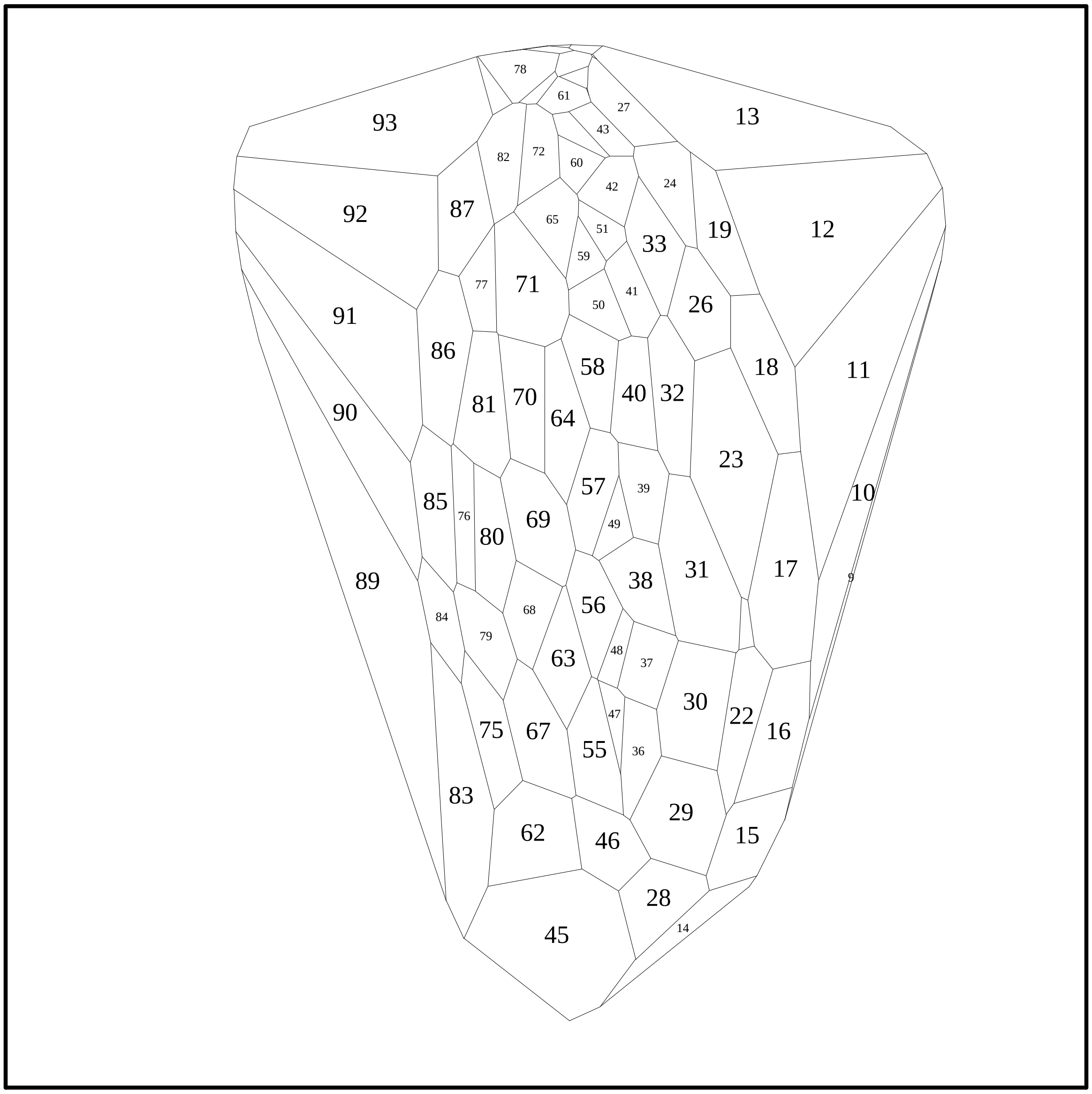

Figure 6: view angle 95°, viewpoint (0,5,0), view direction (0,–5,1)

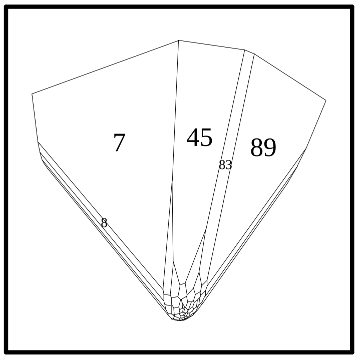

Fig. 7: d = (0,0,1), 30° view

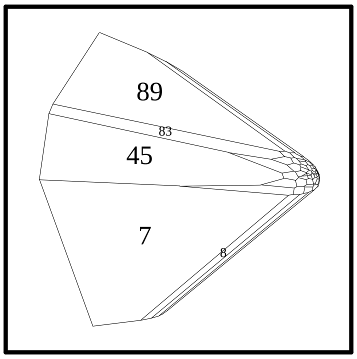

Fig. 8: $d = (-10^{-10},0,1)$

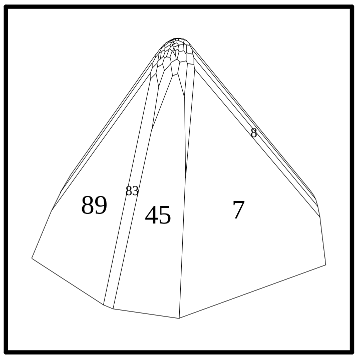

Fig. 9: Figure 7, rotated 180°

The viewpoint, view direction, and view angle do not completely determine our view of a polytope: we are still free to rotate it about the view direction. Following Tom Duff's convention of changing coordinates so the view direction is a positive multiple of (0,1,0), I first rotate things in the (x, y) plane (i.e., about the z axis) to zero the first coordinate of the view direction, then rotate things in the (y, z) plane to zero the third coordinate of the view direction. This determines an orthogonal matrix Q, the product of two Givens rotations, such that a point p is transformed to $p' = Q(p - v)$ before being plotted. (To be visible, $p' = (p'_1, p'_2, p'_3)$ must have $p'_2 > 0$ and $\max\{|p'_1/p'_2|, |p'_3/p'_2|\} \le \tan(\alpha/2)$.)

The above choice of Q makes the least possible change to the vertical, i.e., to the z axis. If the view direction d is nearly parallel to the z axis, then small changes in d can make big changes in Q. Figures 7–9 illustrate this (with a 30° view angle): $d = (0,0,1)$ in figure 7 (and I use no rotation in the (x, y) plane); in figure 8, $d = (-10^{-10},0,1)$ makes the figure turn 90°; and rotating figure 7 by 180° (or, equivalently, choosing d to be, e.g., $(-10^{-20},-10^{-10},1)$) gives figure 9, which is the ‘‘right’’ view relative to figure 2, as it rotates this figure by 90° about its x axis (rotating the top away from us, the bottom towards us).

Given a Q, say $Q^{(1)}$, for a particular view direction $d^{(1)}$, and given a second view direction $d^{(2)}$, we could choose an orthogonal $Q^{(2)}$ to minimize $\|Q^{(1)} - Q^{(2)}\|$ subject to $Q^{(2)} d^{(2)} = \|d^{(2)}\|_2 (0, 1, 0)$, but it is impossible to make Q a continuous function of d. (Choosing $Q(d)$ to be the negative of the obvious Householder transformation would give us a discontinuity at $d = (0,-1,0)$.)

I have tried to rotate the remaining figures sensibly. This generally means an extra 180° rotation for figures with view direction $d = (0,0,1)$ and no extra rotation for other directions.

I choose a cost vector (c in (1a) or b in (2a)) to make the simplex method (as implemented by XMP [17]) trace out a picturesque path, one with several zig-zags. Figure 10 shows figure 6 with the path followed by the simplex method superimposed; the dotted lines connect vertices visited in phase 1 (while XMP sought a feasible point), and the heavy solid lines (19 of them) connect the vertices visited in phase 2. The first phase-1 step is off screen, and the last three phase-1 steps coalesce into what looks like a piece of solid line outside the polytope.

Pictures of the Dual Affine Algorithm

The next several figures deal with a dual affine version of Karmarkar's LP algorithm. This version chooses its step lengths as in [1]: $x^{k+1} = x^k + 0.99\alpha_k^{\max} p^k$ for the first 10 steps and $x^{k+1} = x^k + 0.9\alpha_k^{\max} p^k$ for the remaining iterations, where $\alpha_k^{\max} = \max\{\alpha\colon x^k + \alpha p^k \ge \mathbf{0}\}$ is the maximum feasible step. It stops when it succeeds in finding a point at which the Karush-Kuhn-Tucker optimality conditions hold. (This side computation is an interesting detail to be discussed elsewhere). For the problem at hand, it finds an optimal point during the 19th iteration.

Figure 11 is similar to figure 10: it shows figure 6 with the dual affine algorithm's path superimposed. But whereas the path in figure 10 runs along edges of the polytope, the path in figure 11 lies strictly inside the polytope; to understand figure 11, imagine that the path is glowing and the polytope's faces are translucent but dark enough that you do not see more polytope edges.

Notice that the latter half of the solution path in figure 11 appears to be following polytope edges. This accords with the theory in [18], which says that when an affine iterate lies close to a facet, the corresponding search direction should be nearly parallel to the facet.

Figures 12–14 are similar to figure 11, but are drawn from other viewpoints and with a view angle of 30°. (Except where otherwise noted, all the remaining figures use this view angle.) Like figure 11, figures 12 and 13 are from outside the polytope; figure 14, on the other hand, is a view from inside.

The view changes rapidly when one sits at the *center* (1,1,1) of the affinely transformed polytope $\{x \in \mathbb{R}^n\colon ADx = b,\ x \ge \mathbf{0}\}$, i.e., at the image of the current iterate. View direction $d = (-1,0,0)$ proves interesting in this context. Figures 15–17 show iterations 2–4. For iteration 6, the corresponding view, figure 18, looks much simpler. But if we turn the magnification up sharply by changing the view angle from 30° to 0.005°, then we get the view in figure 19. The situation is similar for iteration 8: figure 20 shows view angle 30°, whereas figure 21 shows much more detail with view angle 0.01°.

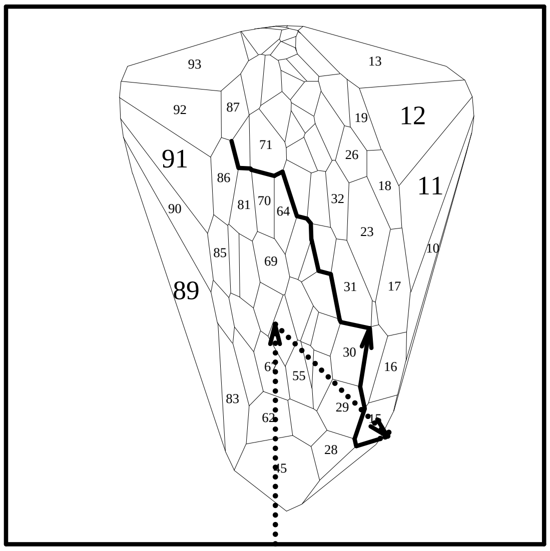

Fig. 10: simplex (XMP) path

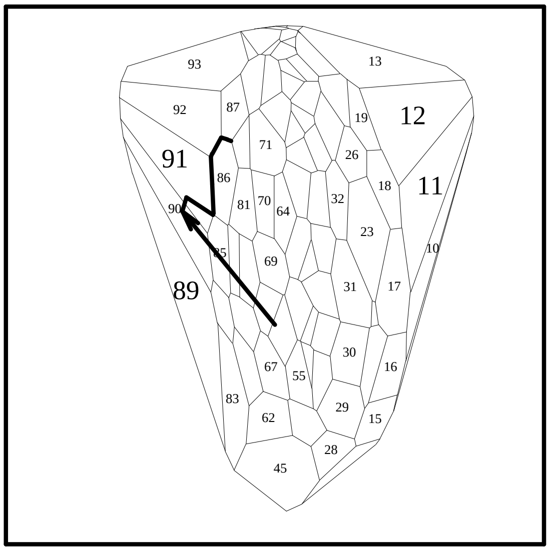

Fig. 11: dual affine path

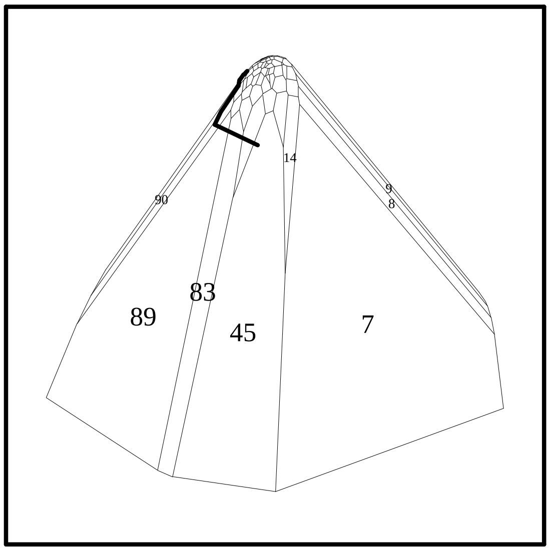

Fig. 12: d = (0,0,1), 30° outside view

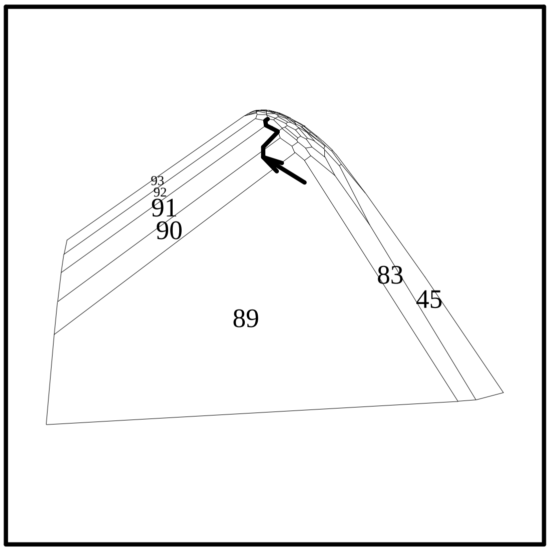

Fig. 13: d = (–1,0,0), outside view

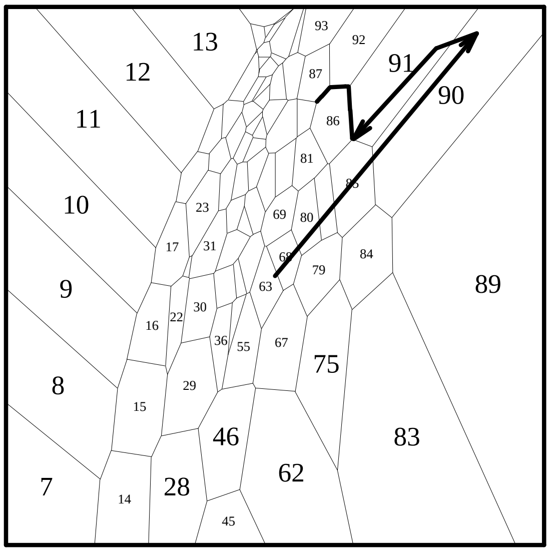

Fig. 14: (inside) v = (0,–7,0), d = (0,1,0)

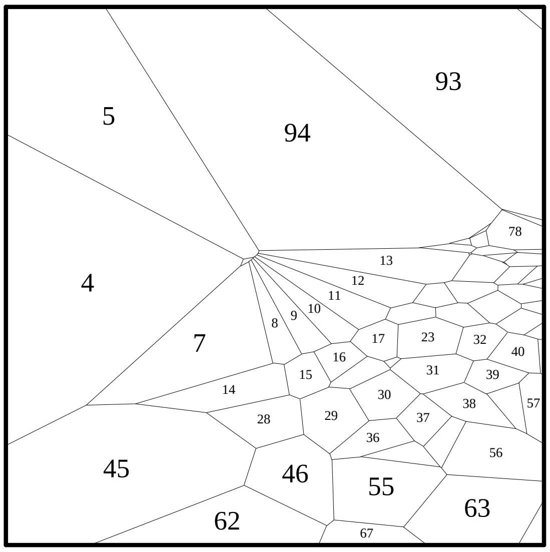

Fig. 15: iter 2, v = (1,1,1), d = (–1,0,0)

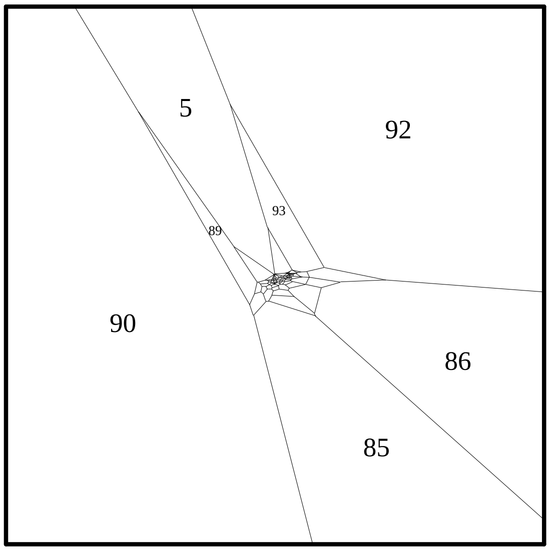

Fig. 16: dual affine iter 3

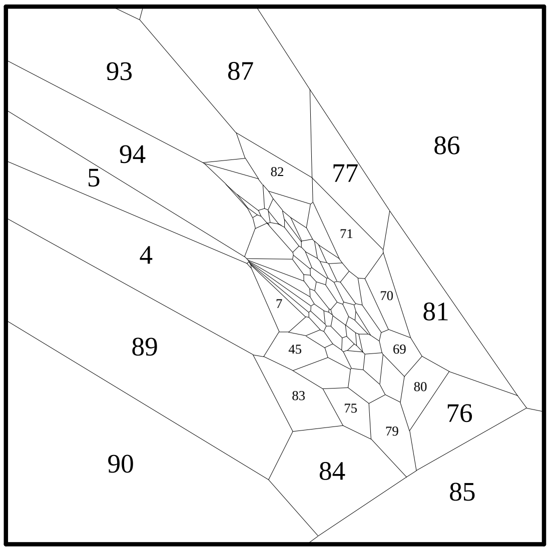

Fig. 17: dual affine iter 4

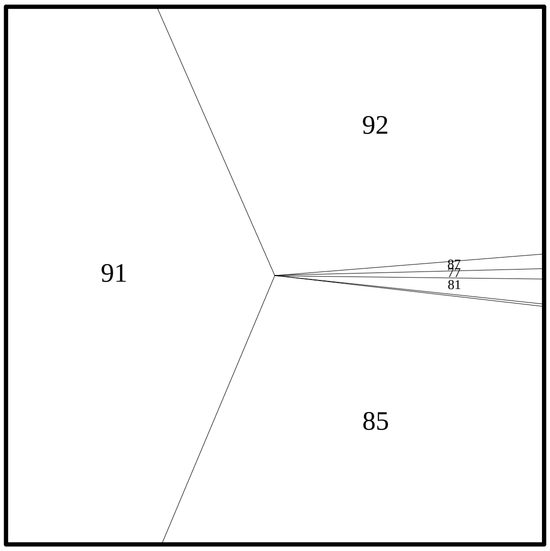

Fig. 18: dual affine iter 6, 30° view

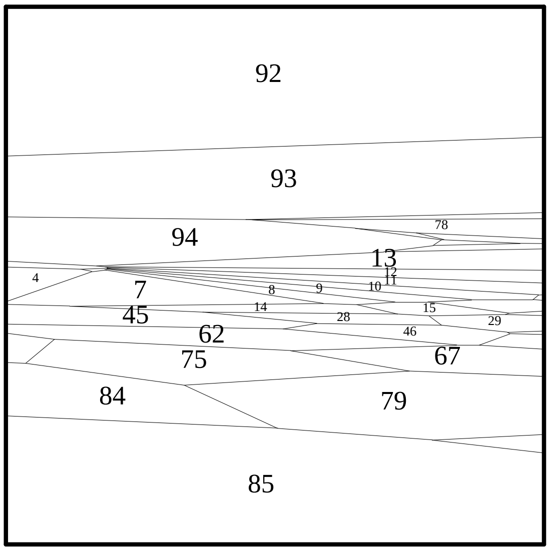

Fig. 19: dual affine iter 6, 0.005° view

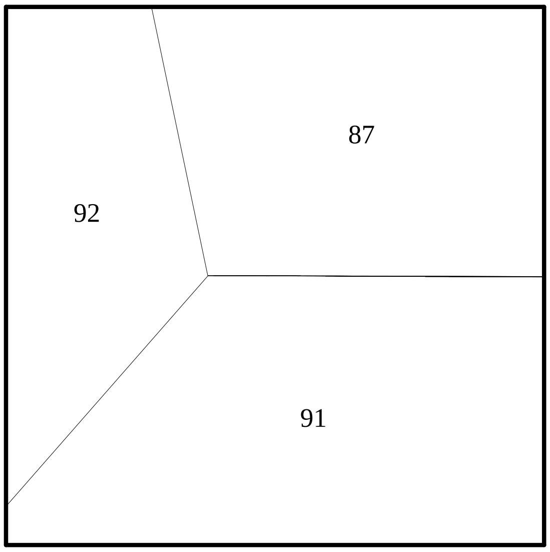

Fig. 20: dual affine iter 8, 30° view

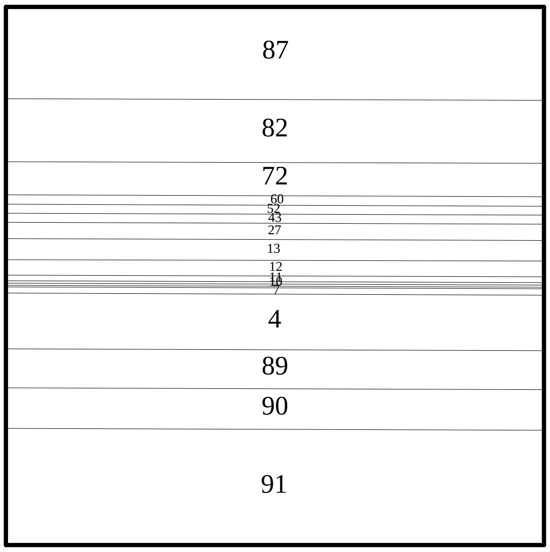

Fig. 21: dual affine iter 8, 0.01° view

By backing away from the center to (10,0,0) or (20,0,0), we can see the steps the algorithm takes. Moving to (10,0,0) in the affinely transformed space of iteration 2 (and retaining the view direction (–1,0,0) used in figures 15–21), for example, we obtain figures 22 and 23. Figure 22 is analogous to figures 11–13, in that you see the step shining through from inside the polytope. Figure 23, on the other hand, shows a *cutaway* view: it is drawn, from outside the polytope, as though the polytope faces were one-way mirrors, visible only from the inside. Figures 24–27 show similar cutaway views for some other iterations; to make the step fit, figure 24 is from (20,0,0), while the others are from (10,0,0). Despite the extreme early stretching shown in figures 18–21, figures 26 and 27 show that later iterations may bring some of the polytope detail back into view.

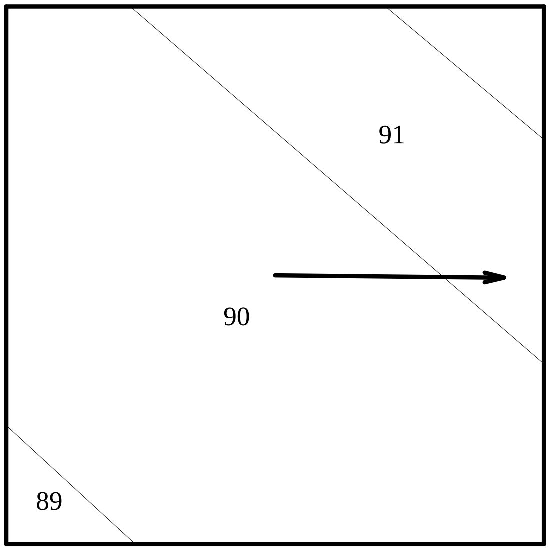

Fig. 22: dual affine iter 2, $v = (10,0,0)$

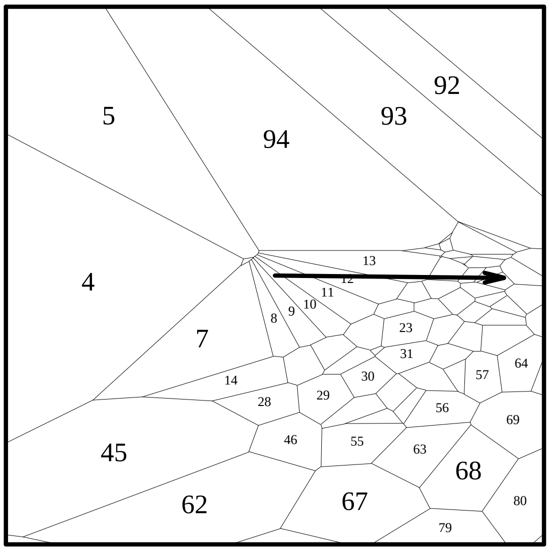

Fig. 23: cutaway view of fig. 22

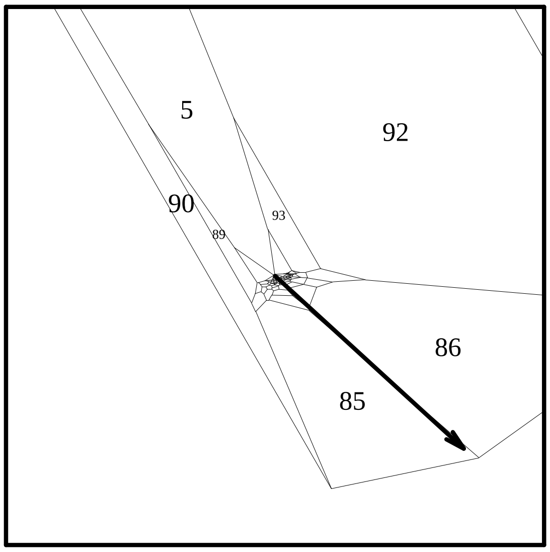

Fig. 24: iter 3, cutaway, $v = (20,0,0)$

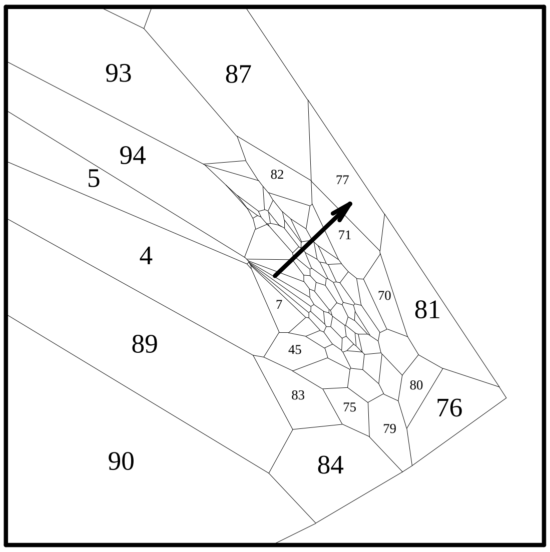

Fig. 25: iter 4, cutaway, $v = (10,0,0)$

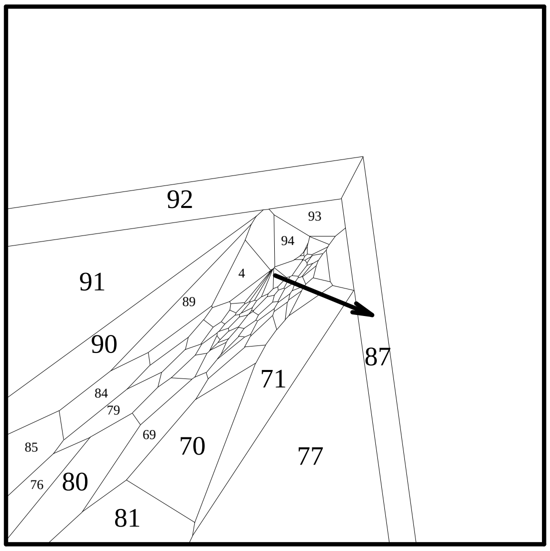

Fig. 26: dual affine iter 10, cutaway

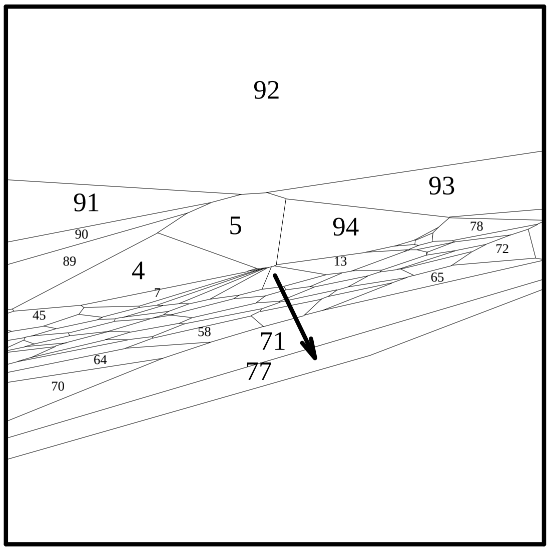

Fig. 27: dual affine iter 18, cutaway

Figures 28 and 29 show two outside views of the polytope as affinely transformed in iteration 1; visible inside it is the step taken. In subsequent iterations, the transformed polytope gets so long and thin that there would be little point in showing outside views of it.

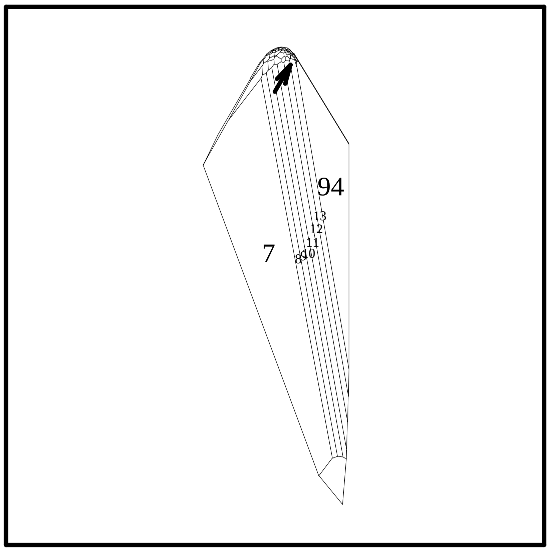

Fig. 28: iter 1, outside, $d = (0,-1,0)$

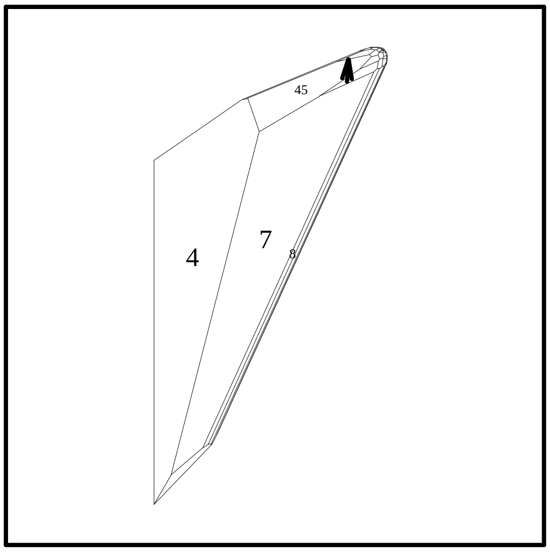

Fig. 29: iter 1, outside, $d = (0,0,1)$

Pictures of the Dual Projective Algorithm

The next several figures illustrate a dual projective version of Karmarkar's LP algorithm. This version chooses its step lengths by doing a line search on the potential function (6) (using quadratic interpolation until it reduces the directional derivative's absolute value to at most 0.01 times its initial value). It uses the same stopping tests as the dual affine algorithm considered above. In fact, it starts out running the dual affine algorithm and switches to the dual projective algorithm only after it has computed a primal feasible iterate x (and therewith a valid upper bound on the optimal dual objective function value). For the problem at hand, it computes a primal feasible x on the first iteration and runs a total of 14 iterations before finding an optimal solution. The linear objective $b^{\mathrm{T}} y$ increases in all but the second and third iterations. (It is easy to make $b^{\mathrm{T}} y$ monotonic by adding the constraint that it remain constant on iterations where it would otherwise decrease; for the problem at hand this does not change the number of iterations, but does require computing two extra projections.)

In analogy with figure 11, figure 30 shows figure 6 with the path followed by the dual projective algorithm showing through from inside. Figures 31 and 32 show the same outside views as figures 12 and 13, but with the dual projective path visible from inside. Figure 33 is similar to figure 14, but is drawn from a bit further back and with a wider view angle to bring the whole path into view.

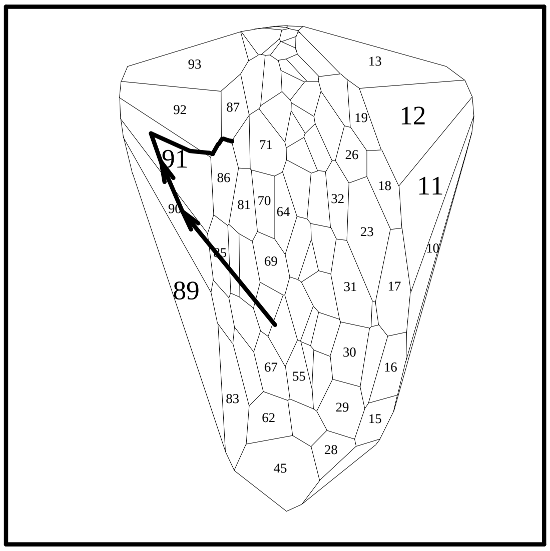

Fig. 30: dual projective path

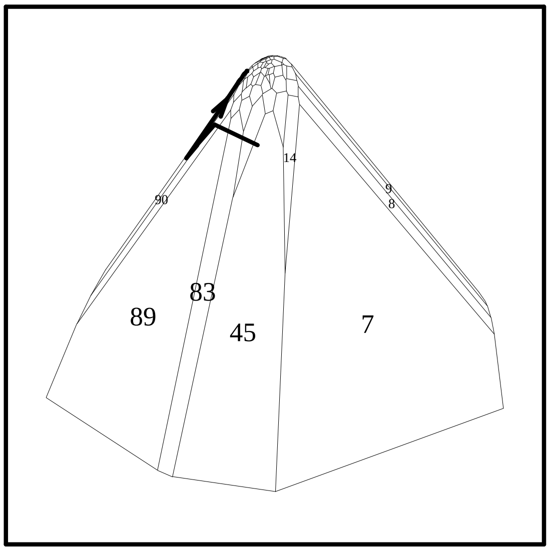

Fig. 31: d = (0,0,1), 30° outside view

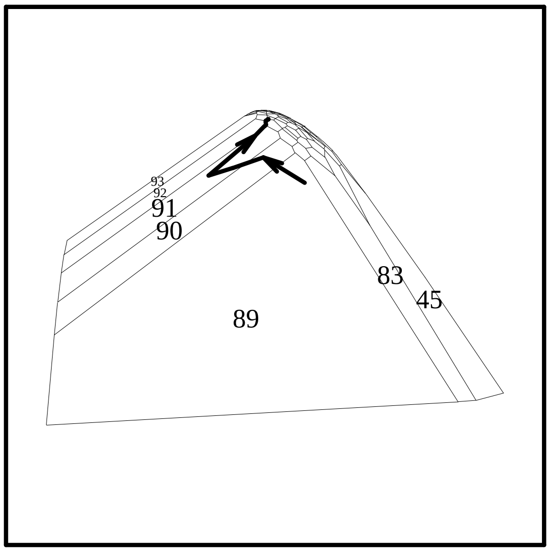

Fig. 32: d = (–1,0,0), outside view

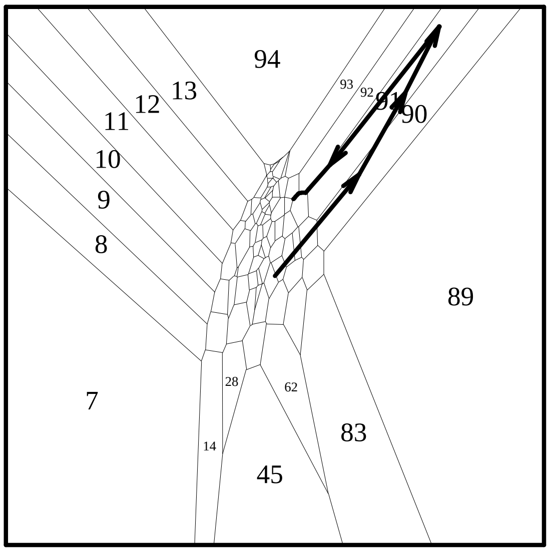

Fig. 33: 50° view, v = (0,–10,0)

In stark contrast with the dual affine algorithm, the transformed polytopes of the dual projective algorithm stay reasonably round. Indeed, since the image of the current iterate is the center $e/(n+1)$ of the projectively transformed polytope, whose only inequality constraints are those of the unit simplex, the center is at least a distance of $1/\sqrt{n(n+1)}$ and at most a distance of $\sqrt{n/(n+1)}$ from the boundary of the transformed polytope. Figures 34–46 give an outside view of the projectively transformed polytope in iterations 2–14, all from view direction $d = (0,0,1)$. Also showing through from inside the polytopes in these figures and depicted by heavy arrows are the steps taken. Usually these steps are short (somewhere near $1/\sqrt{n(n+1)}$) or are nearly parallel to the direction of view, so the arrowhead (which is of fixed size) is about all that appears. In going from one iteration to the next, the polytope often appears to be rotated a bit and mildly massaged. The view in figure 46 is not very informative; figure 47 shows the same polytope from a different view direction, $d = (0,-1,0)$. (Since iteration 1 is an affine step, figures 28 and 29 illustrate its transformed polytope.)

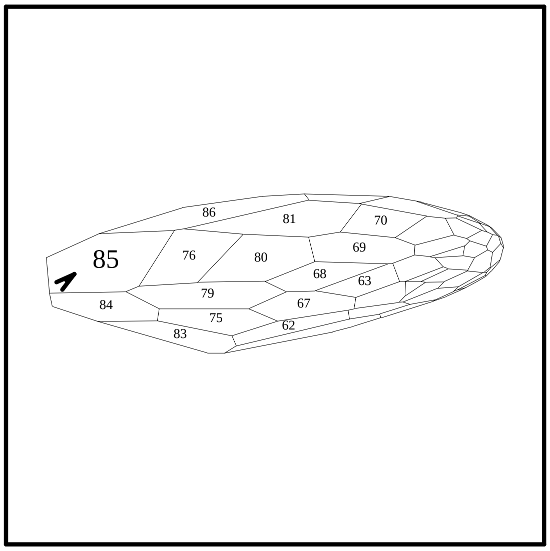

Fig. 34: dual projective iter 2, outside

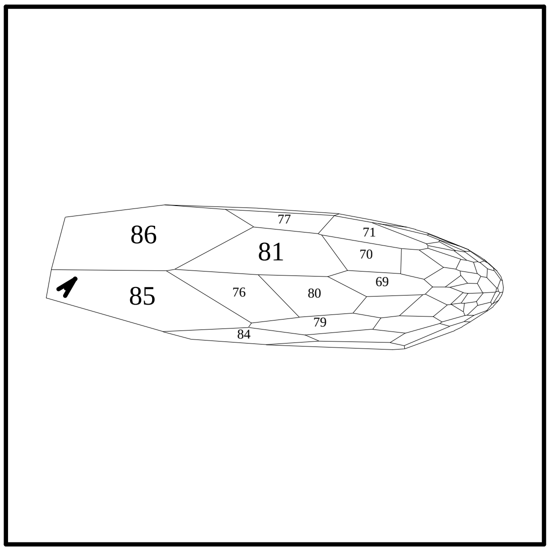

Fig. 35: dual projective iter 3, outside

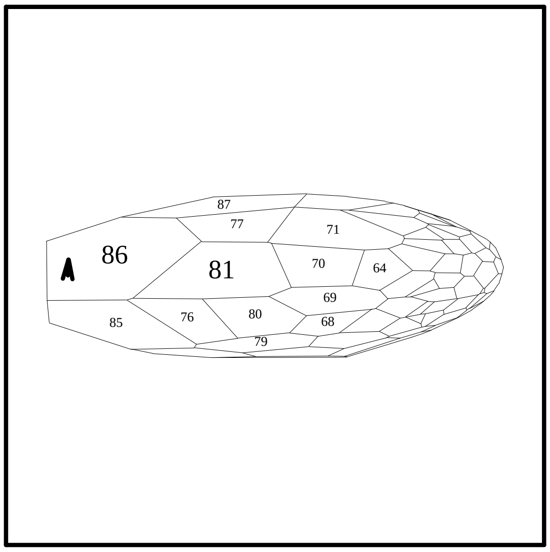

Fig. 36: dual projective iter 4, outside

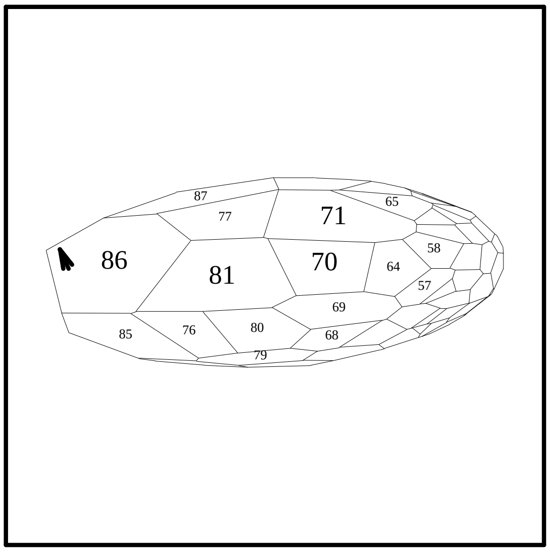

Fig. 37: dual projective iter 5, outside

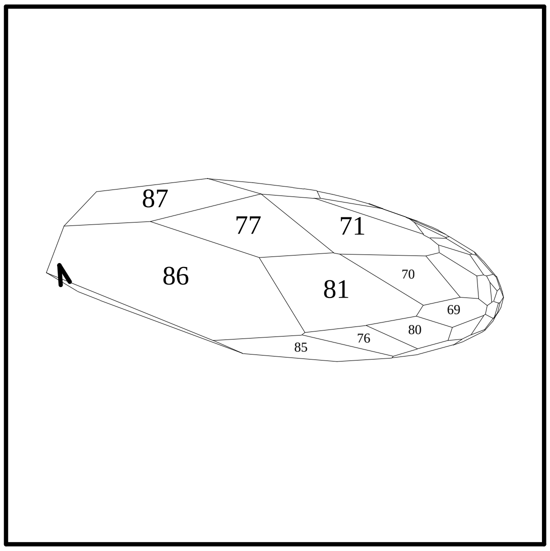

Fig. 38: dual projective iter 6, outside

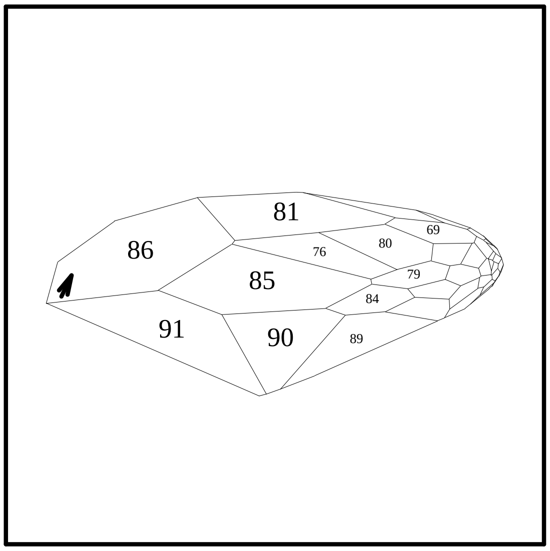

Fig. 39: dual projective iter 7, outside

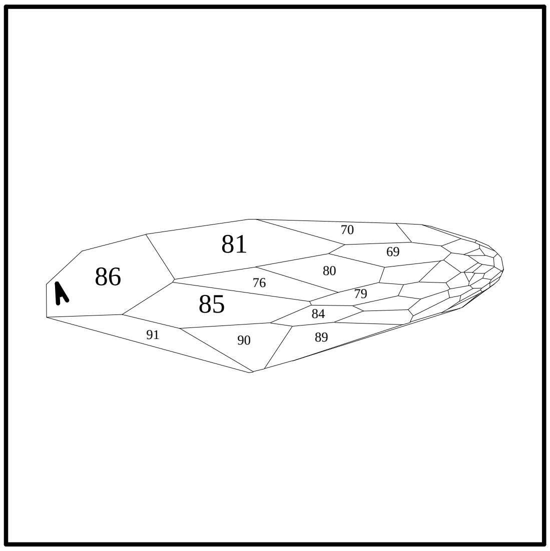

Fig. 40: dual projective iter 8, outside

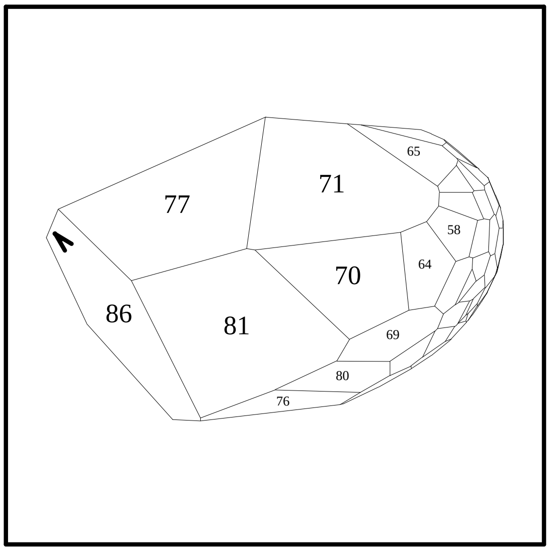

Fig. 41: dual projective iter 9, outside

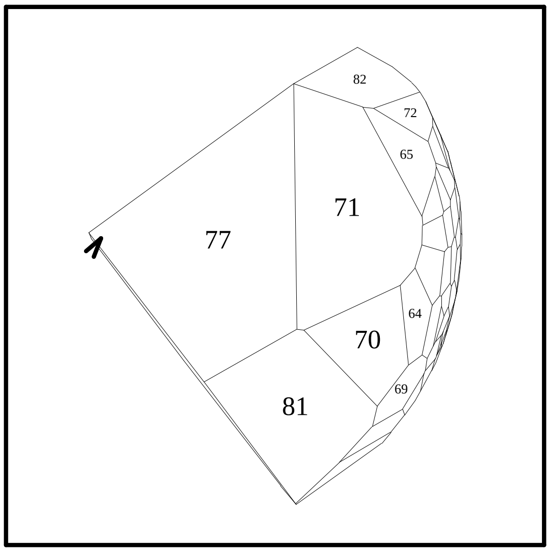

Fig. 42: dual projective iter 10, outside

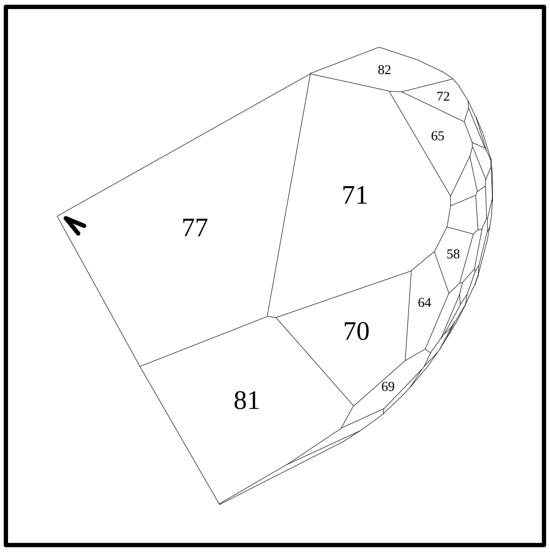

Fig. 43: dual projective iter 11, outside

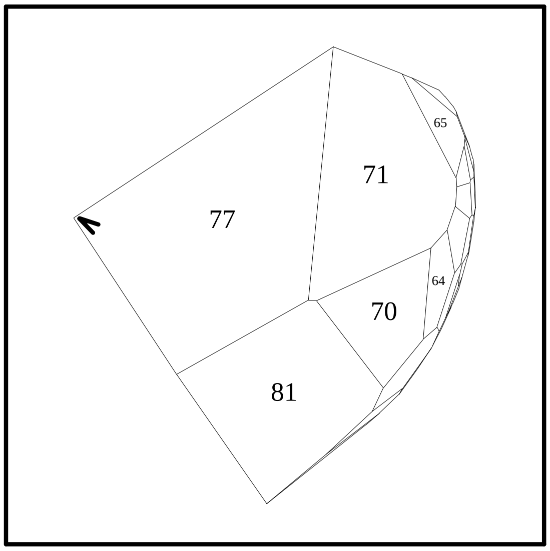

Fig. 44: dual projective iter 12, outside

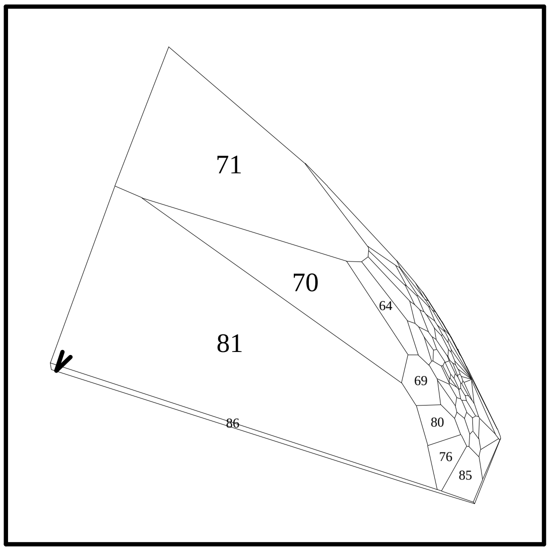

Fig. 45: dual projective iter 13, outside

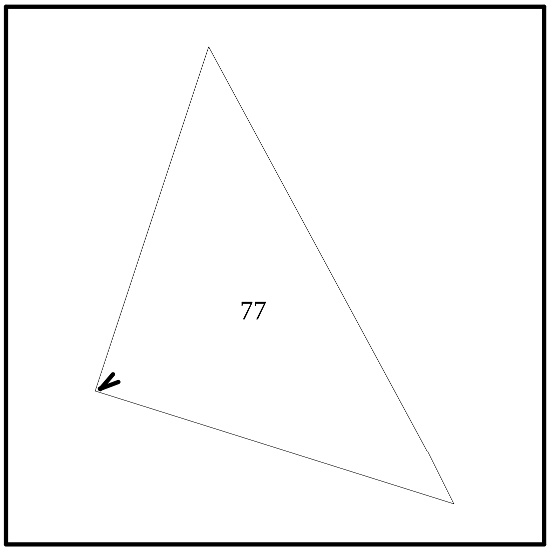

Fig. 46: dual projective iter 14, outside

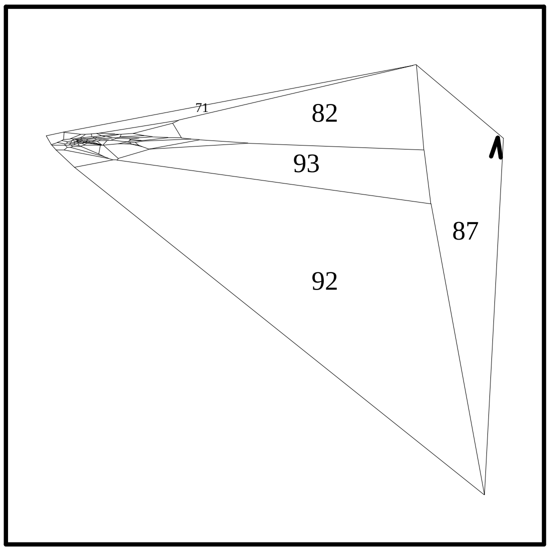

Fig. 47: iter 14, outside, $d = (0,-1,0)$

Figures 48–73 show views in direction $d = (-1,0,0)$ of the projectively transformed polytope. The even numbered figures in this range are views from the center of the transformed polytope, and the odd numbered figures are cutaway views from the center plus (0.2,0,0), i.e., from outside of the polytope, showing the step taken in the current iteration (analogously to figures 23–27). Again from this perspective the picture changes less wildly than for the dual affine algorithm.

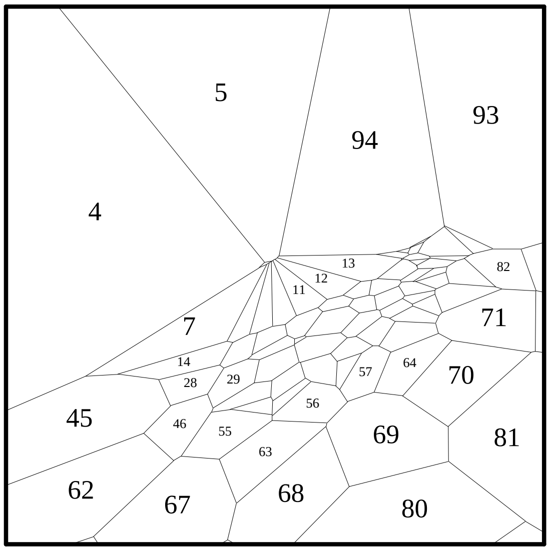

Fig. 48: dual proj. it. 2, *v* = *center*

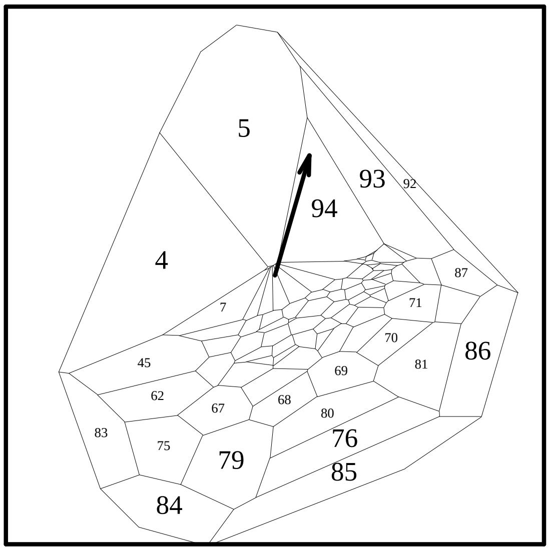

Fig. 49: dual proj. it. 2, *v* = *center*+(.2,0,0)

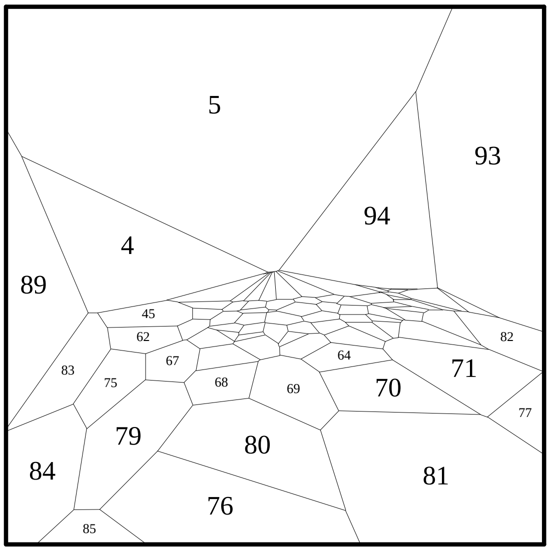

Fig. 50: dual proj. it. 3, *v* = *center*

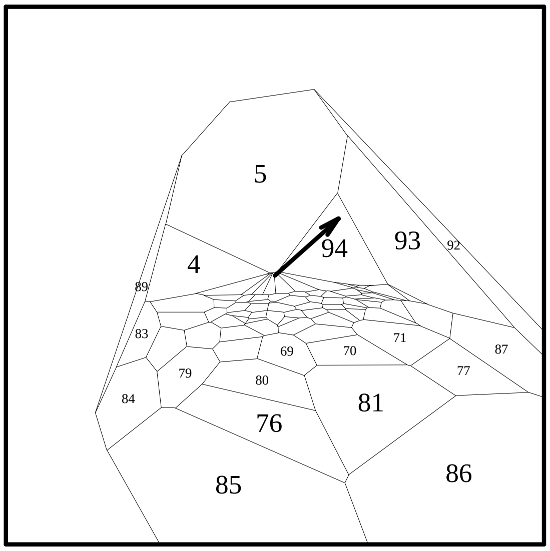

Fig. 51: dual proj. it. 3, *v* = *center*+(.2,0,0)

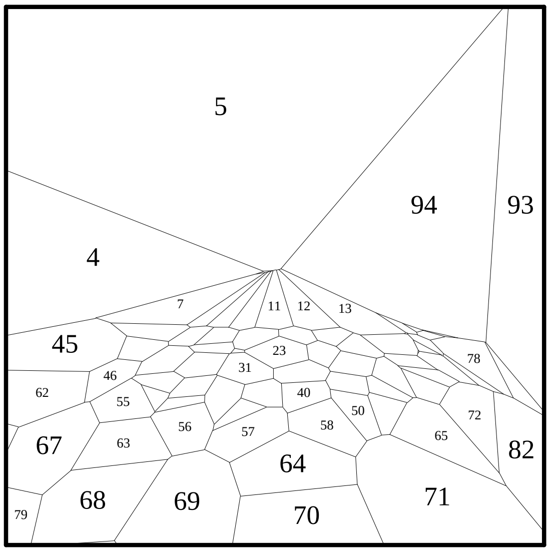

Fig. 52: dual proj. it. 4, *v* = *center*

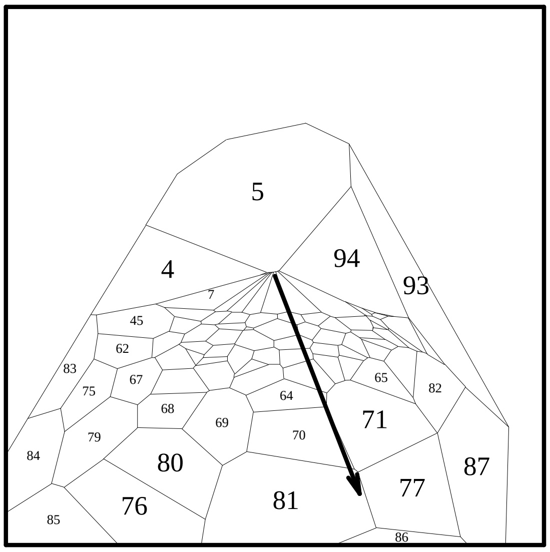

Fig. 53: dual proj. it. 4, *v* = *center*+(.2,0,0)

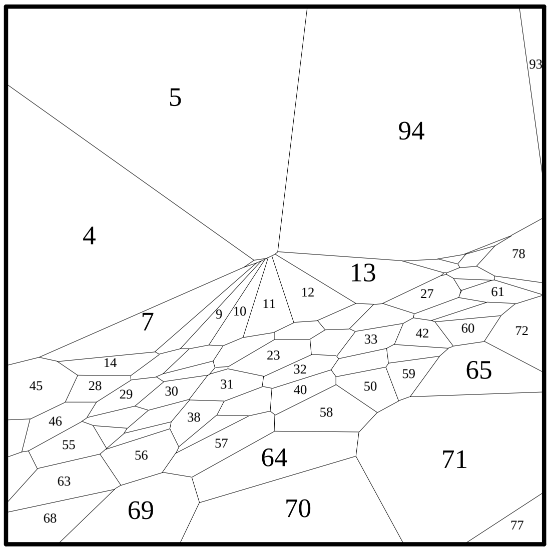

Fig. 54: dual proj. it. 5, v = *center*

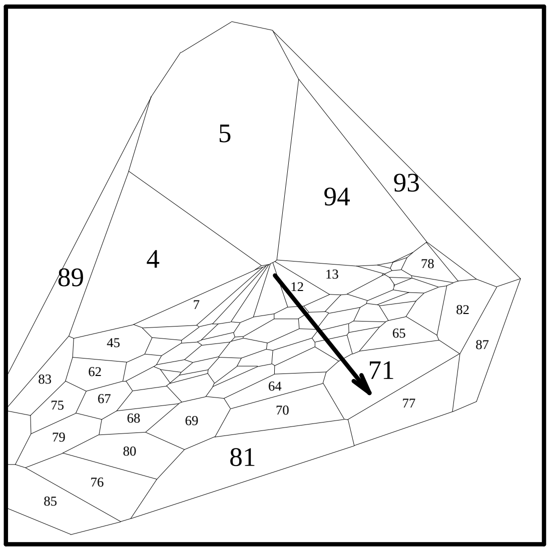

Fig. 55: dual proj. it. 5, v = *center*+(.2,0,0)

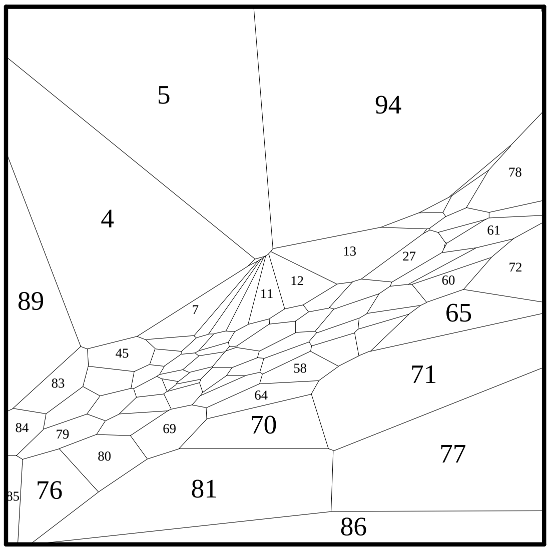

Fig. 56: dual proj. it. 6, v = *center*

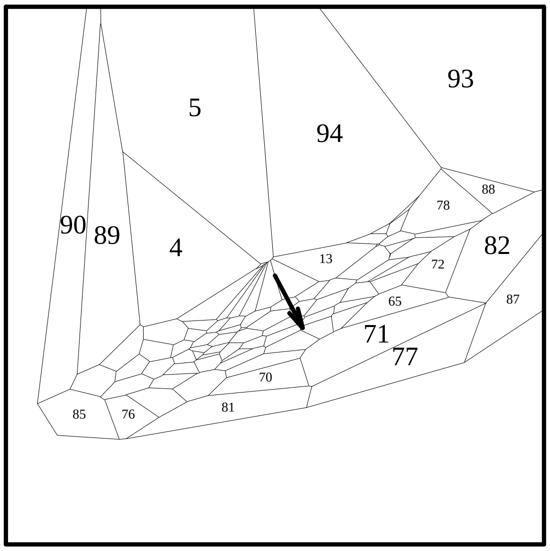

Fig. 57: dual proj. it. 6, v = *center*+(.2,0,0)

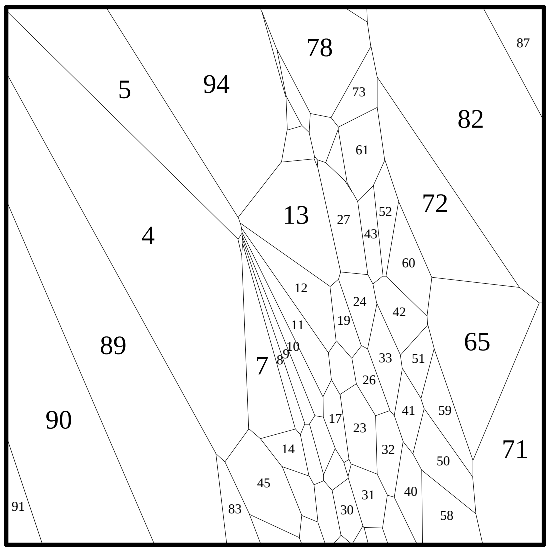

Fig. 58: dual proj. it. 7, v = *center*

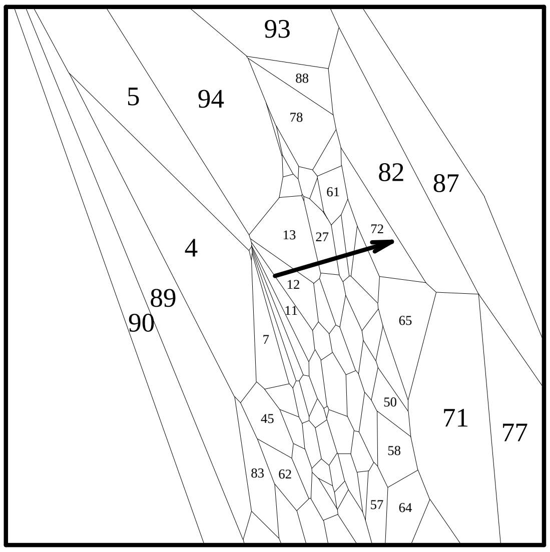

Fig. 59: dual proj. it. 7, v = *center*+(.2,0,0)

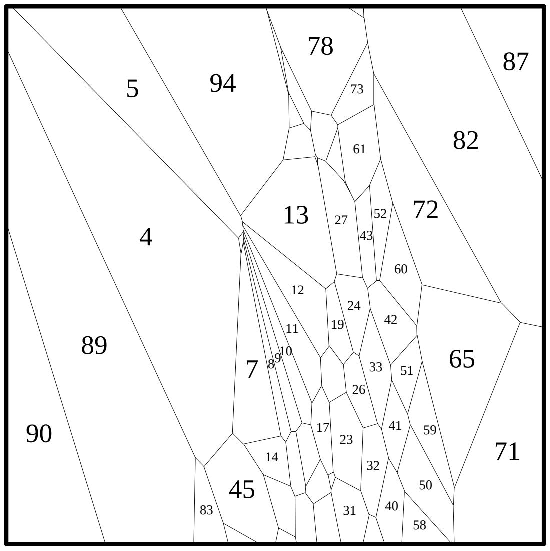

Fig. 60: dual proj. it. 8, *v* = *center*

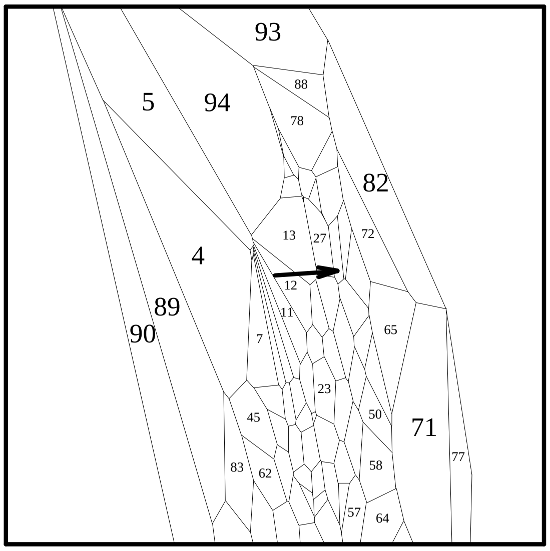

Fig. 61: dual proj. it. 8, *v* = *center*+(.2,0,0)

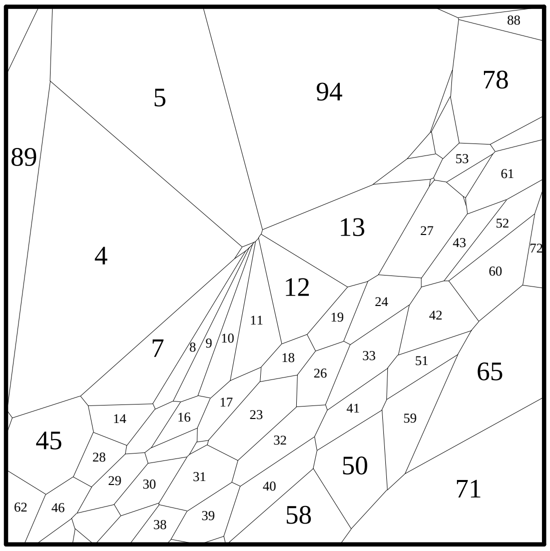

Fig. 62: dual proj. it. 9, *v* = *center*

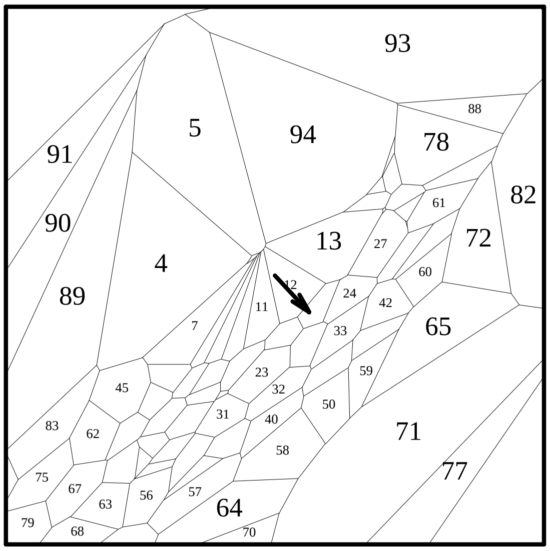

Fig. 63: dual proj. it. 9, *v* = *center*+(.2,0,0)

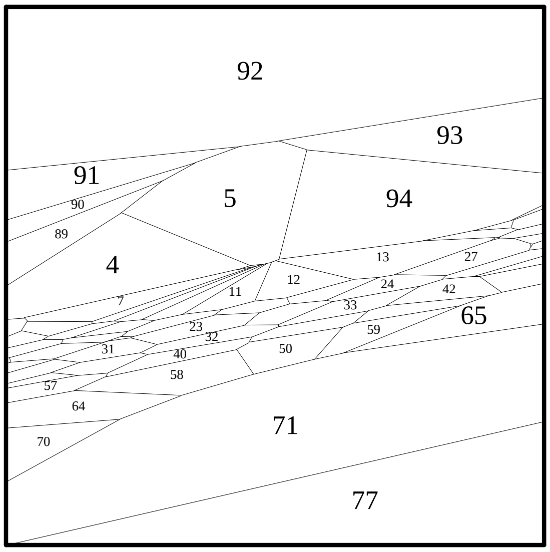

Fig. 64: dual proj. it. 10, *v* = *center*

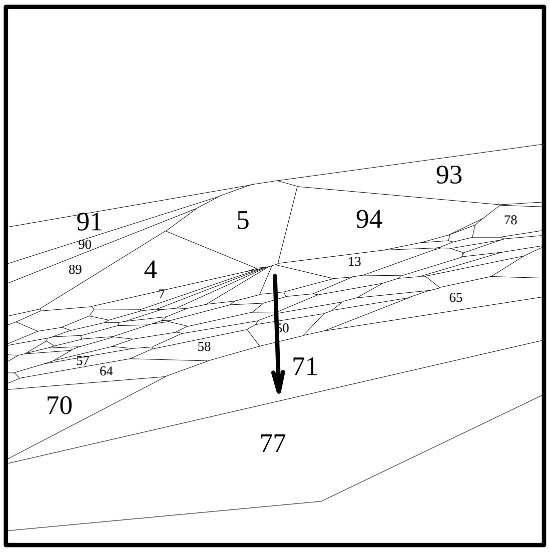

Fig. 65: dual proj. it. 10, *v* = *center*+(.2,0,0)

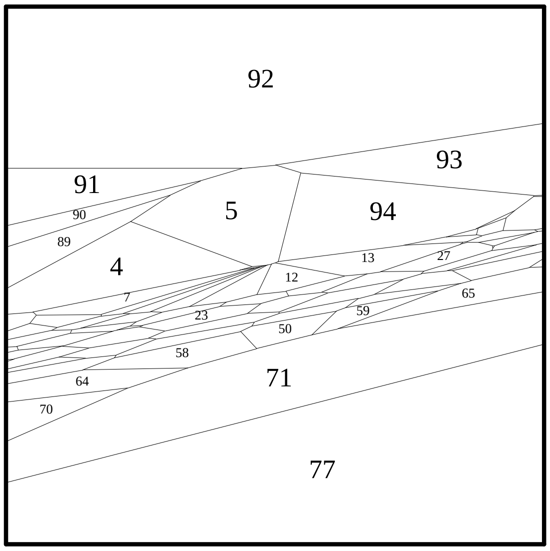

Fig. 66: dual proj. it. 11, *v* = *center*

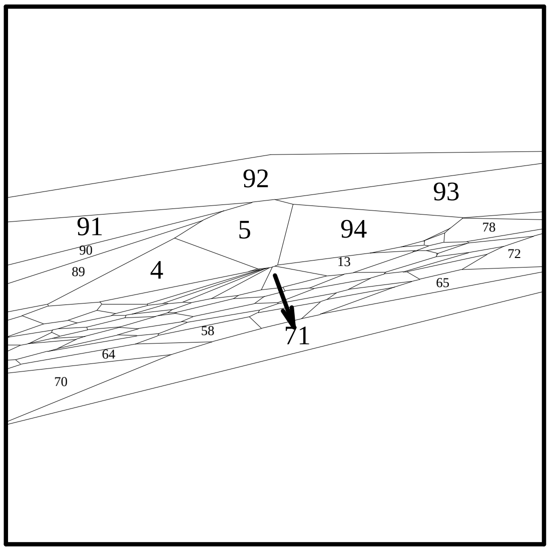

Fig. 67: dual proj. it. 11, *v* = *center*+(.2,0,0)

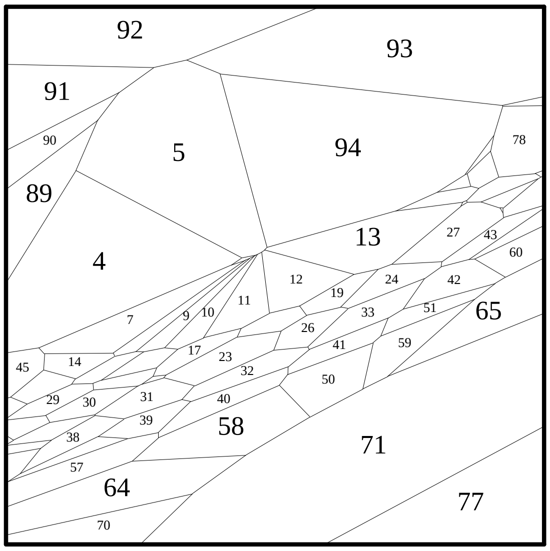

Fig. 68: dual proj. it. 12, *v* = *center*

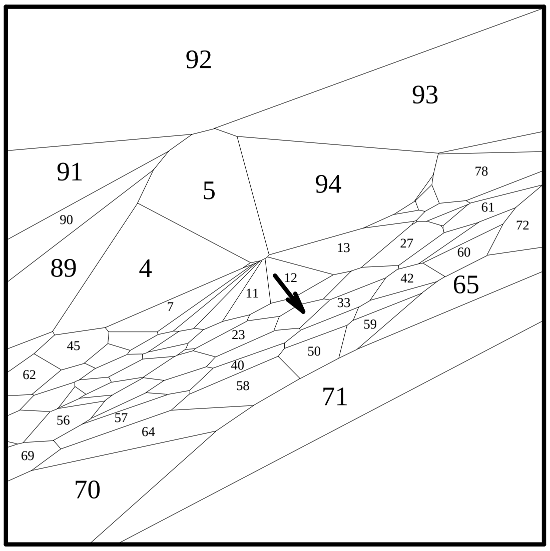

Fig. 69: dual proj. it. 12, *v* = *center*+(.2,0,0)

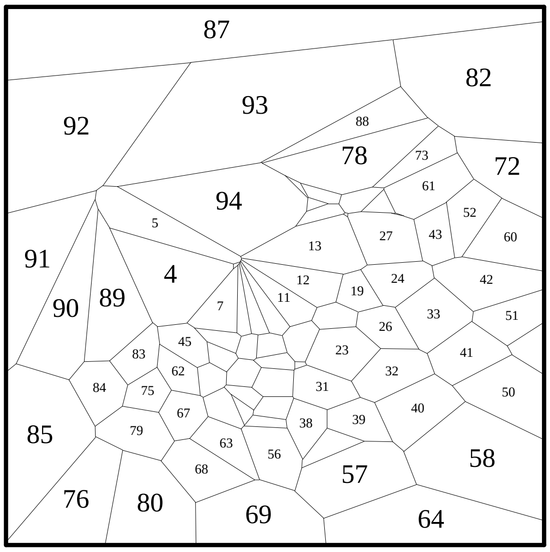

Fig. 70: dual proj. it. 13, *v* = *center*

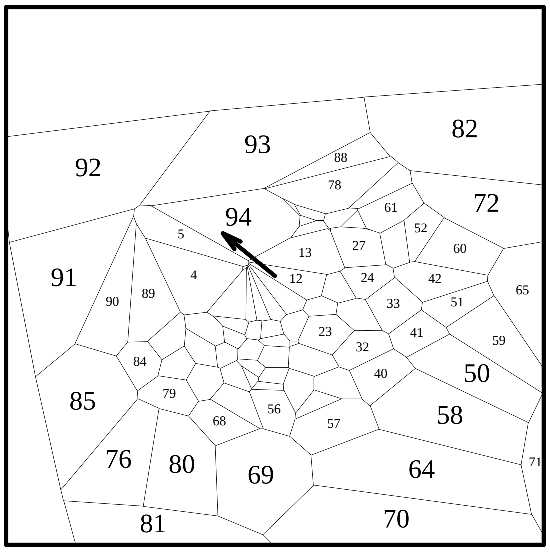

Fig. 71: dual proj. it. 13, *v* = *center*+(.2,0,0)

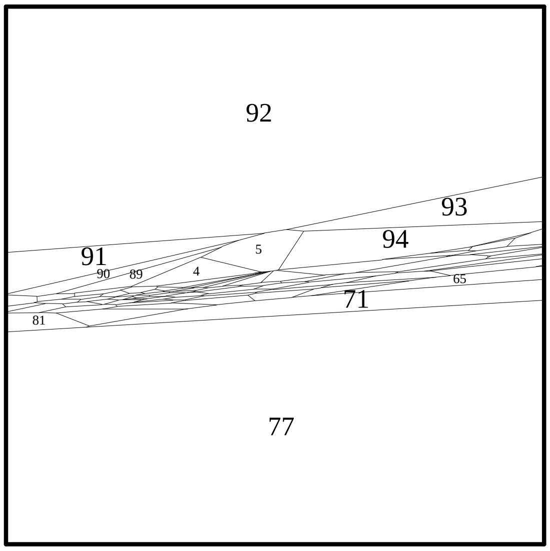

Fig. 72: dual proj. it. 14, v = *center*

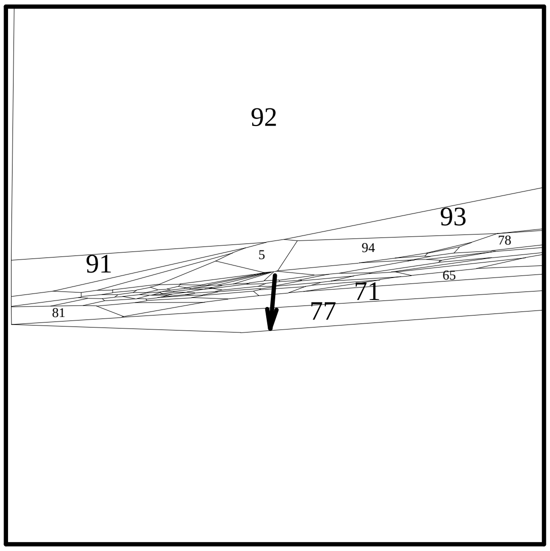

Fig. 73: dual proj. it. 14, v = *center*+(.2,0,0)

Pictures of the Primal Projective Algorithm

The next group of figures is for a primal projective version of Karmarkar's LP algorithm. This version uses the same step-length choice as the dual projective algorithm. It uses a less satisfactory termination test, stopping once $c^{\mathrm{T}} x^k + \beta^k \le 10^{-7} | c^{\mathrm{T}} x^k |$. It also runs a separate phase 1 to find a feasible point. (The dual algorithms add a large bound on the primal variables, which makes finding an initial dual-feasible point straightforward.) The LP problem of concern here is easy in this regard: only one phase-1 iteration is required. In what follows we restrict our attention to the phase-2 iterations. Of these there are 21, and the linear objective $c^{\mathrm{T}} x$ increases on the first two.

The LP problem is the one presented to the dual algorithms, but with slack variables explicitly added. Thus if the dual constraints (2b) are $A^\circ y \le c$, then the primal constraints (1b), ignoring bounds, are $Ax = b$ with $A = [A^{\circ\mathrm{T}}, I]$; an obvious F with $AF = \mathbf{0}$ is then $F = \begin{bmatrix} I \\ -A^\circ \end{bmatrix}$. With simple bound constraints on all the variables, A becomes

$$A = \begin{bmatrix} A^{\circ\mathrm{T}} & I & \mathbf{0} & \mathbf{0} \\ I & \mathbf{0} & I & \mathbf{0} \\ \mathbf{0} & I & \mathbf{0} & I \end{bmatrix} \tag{18}$$

and F becomes $F = [I, -A^\circ, -I, A^\circ]^{\mathrm{T}}$, which we then orthogonalize. The simple bound constraints are superfluous, but do somewhat affect the solution path and pictures.

Figures 74–77 show some outside views of the original polytope, with the primal projective path showing through from inside. Except for this path, figures 74–76 are the same as figures 30–32; figure 77 shows the path from yet another view direction, $d = (-1, 0, 1)$, halfway between that of figures 75 and 76.

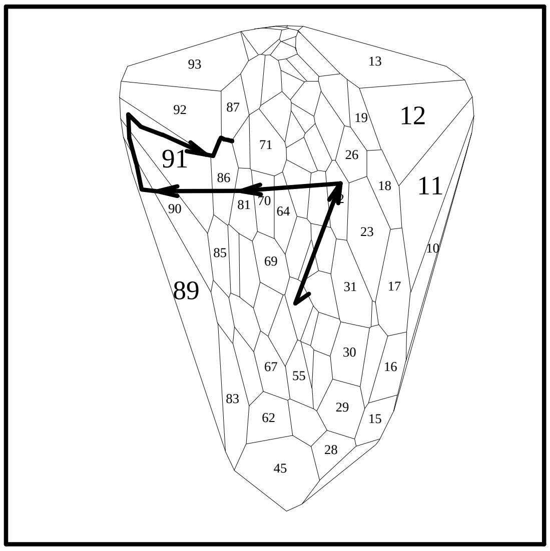

Fig. 74: primal projective path

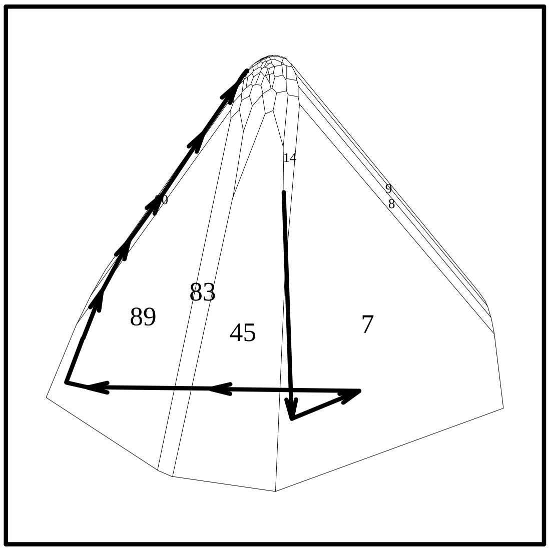

Fig. 75: d = (0,0,1), 30° outside view

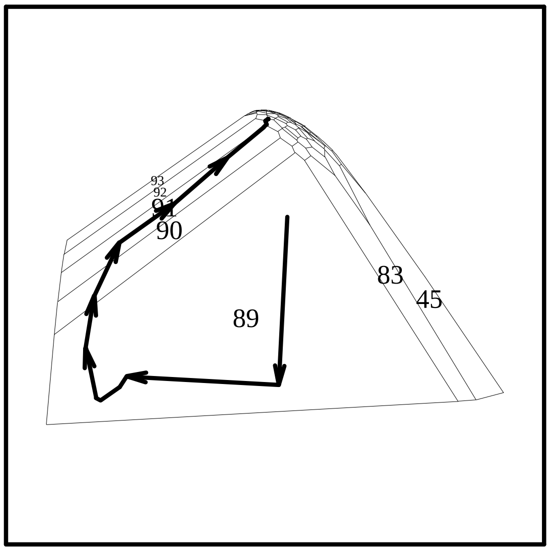

Fig. 76: d = (–1,0,0), outside view

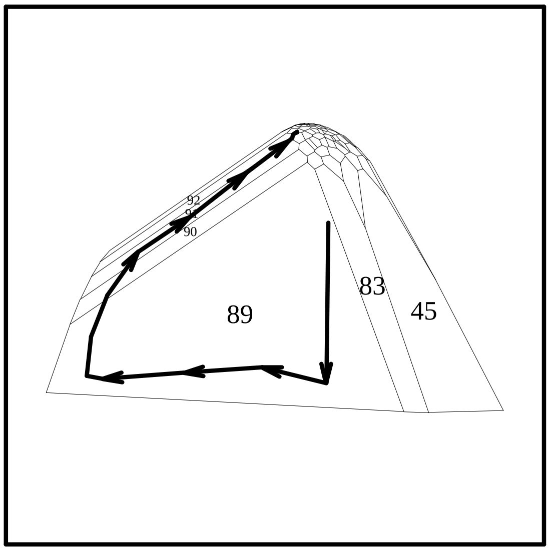

Fig. 77: d = (–1,0,1), outside view

Like the transformed polytopes of the dual projective algorithm, those of the primal projective algorithm stay reasonably round, but they change more rapidly, especially at first. Figures 78–98 show these polytopes on iterations 2–22 (the phase-2 iterations), with the steps taken showing through from inside; figure 99 is analogous to figure 47, showing the final polytope from view direction $d = (0, -1, 0)$.

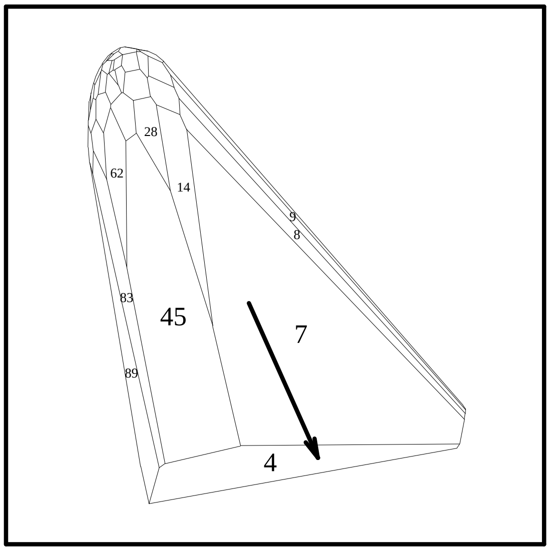

Fig. 78: primal projective iter 2, outside

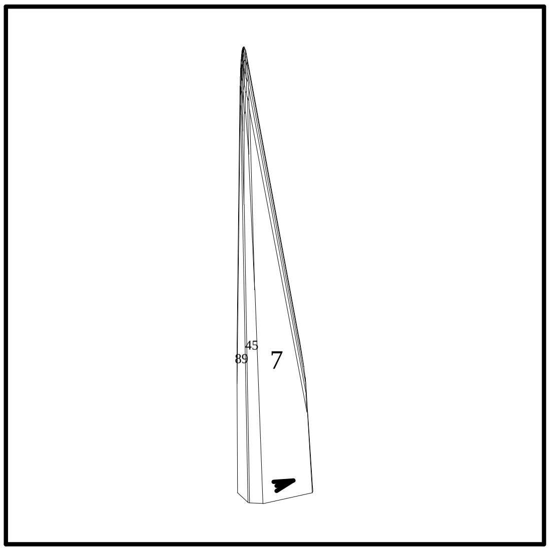

Fig. 79: primal projective iter 3, outside

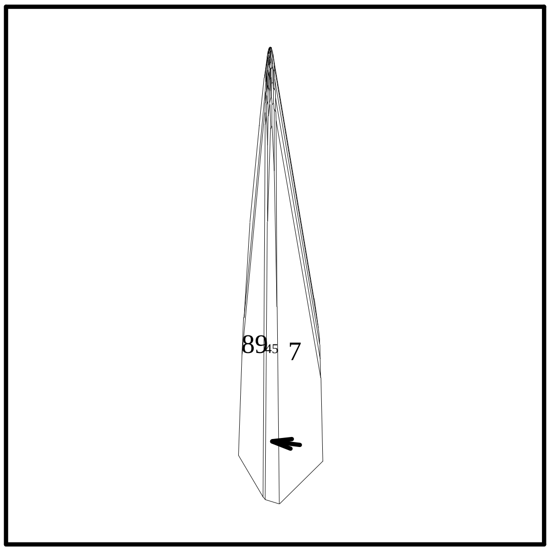

Fig. 80: primal projective iter 4, outside

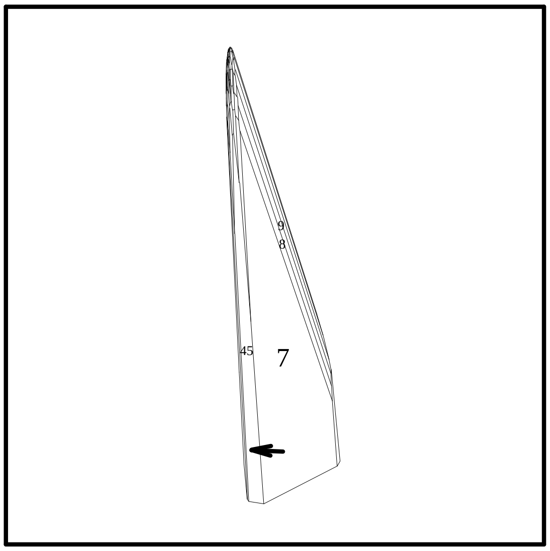

Fig. 81: primal projective iter 5, outside

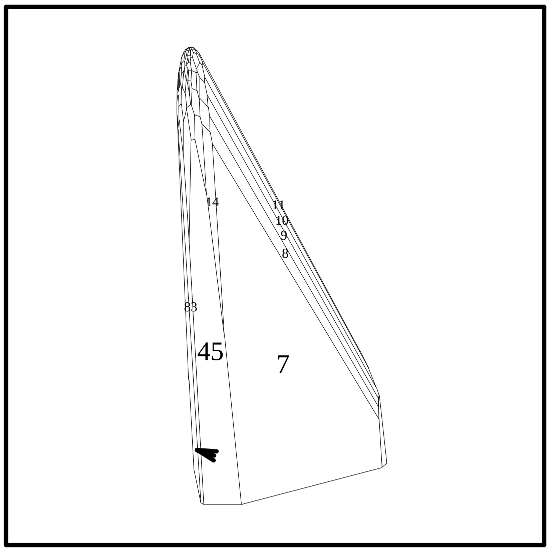

Fig. 82: primal projective iter 6, outside

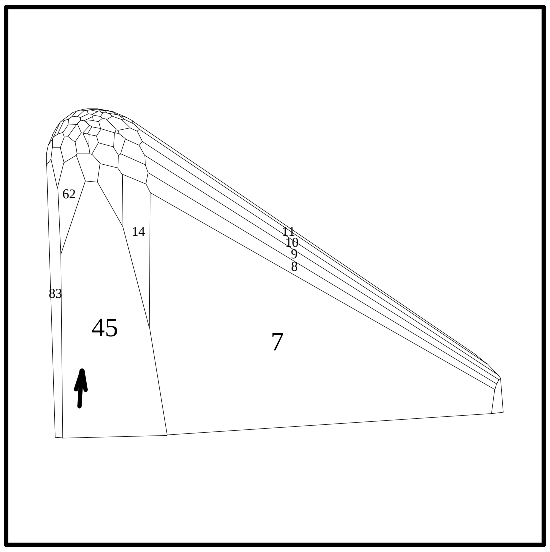

Fig. 83: primal projective iter 7, outside

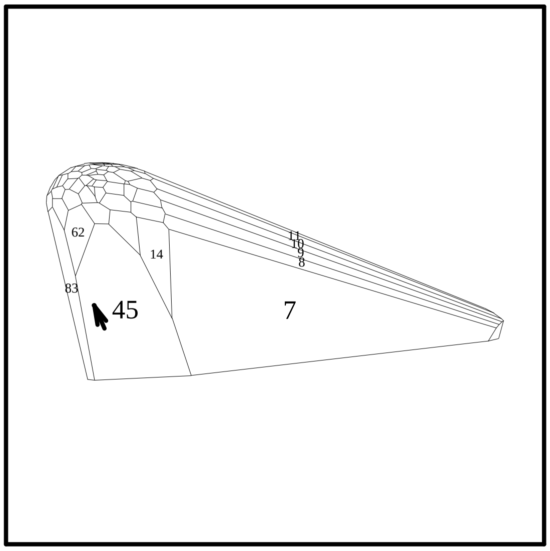

Fig. 84: primal projective iter 8, outside

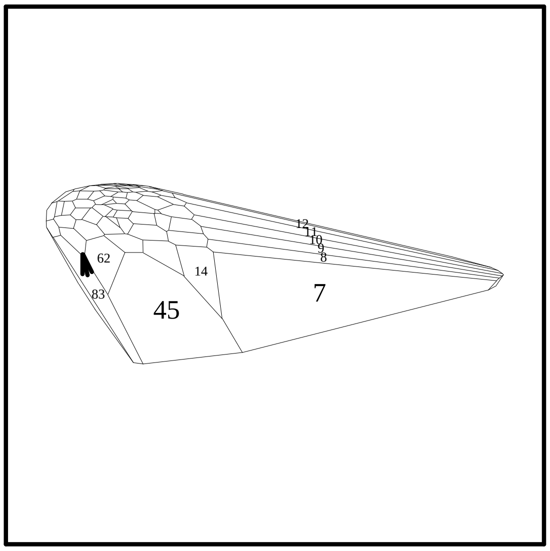

Fig. 85: primal projective iter 9, outside

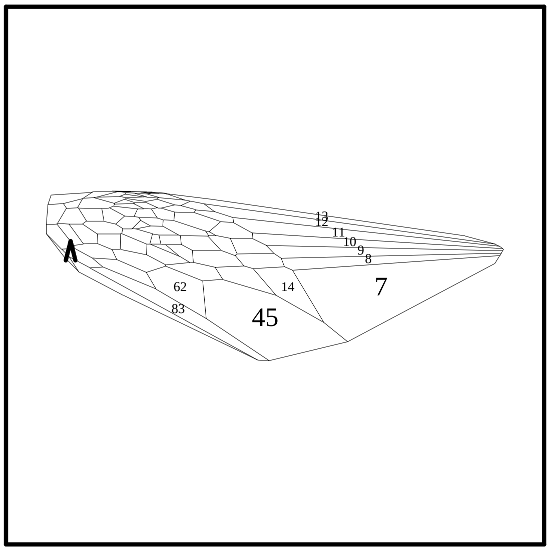

Fig. 86: primal projective iter 10, outside

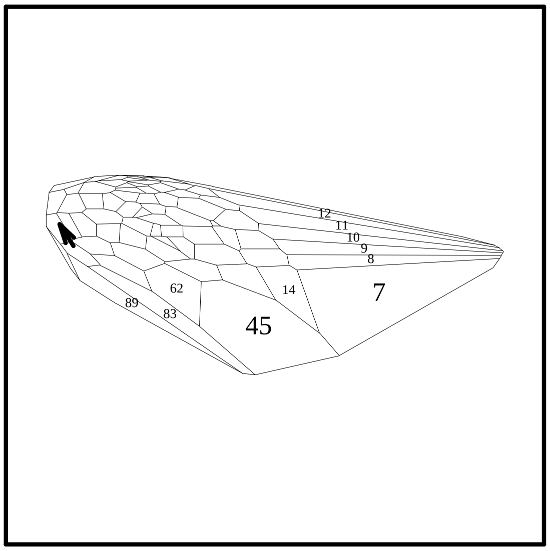

Fig. 87: primal projective iter 11, outside

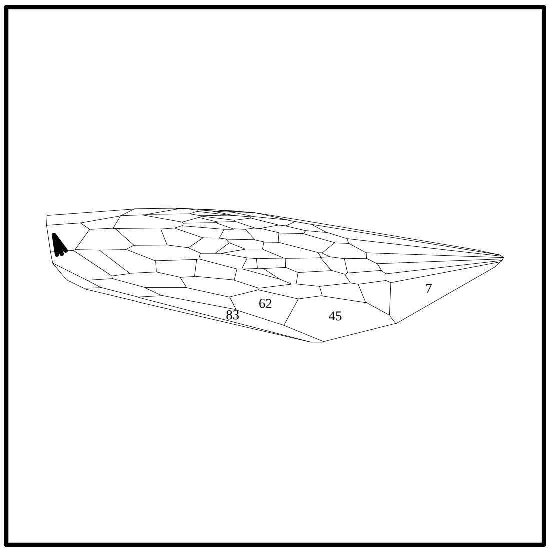

Fig. 88: primal projective iter 12, outside

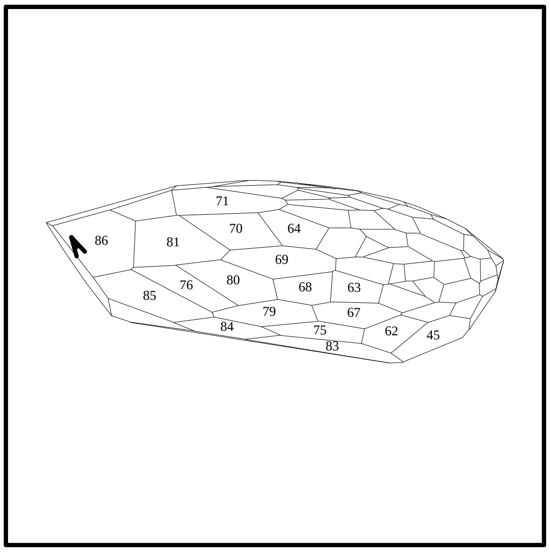

Fig. 89: primal projective iter 13, outside

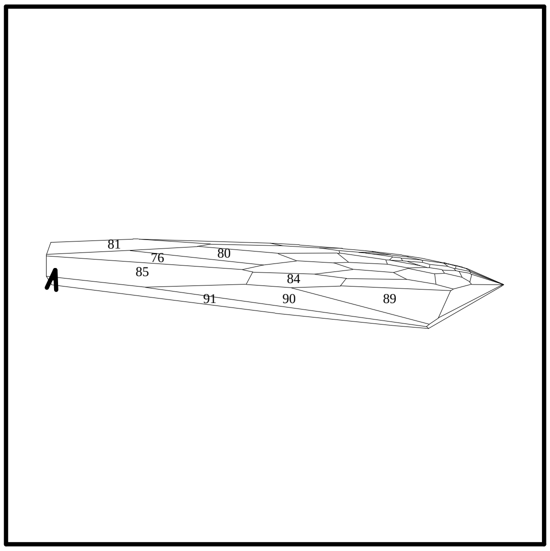

Fig. 90: primal projective iter 14, outside

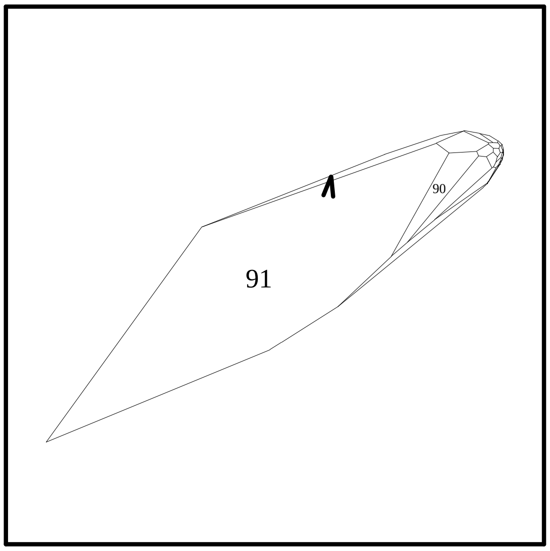

Fig. 91: primal projective iter 15, outside

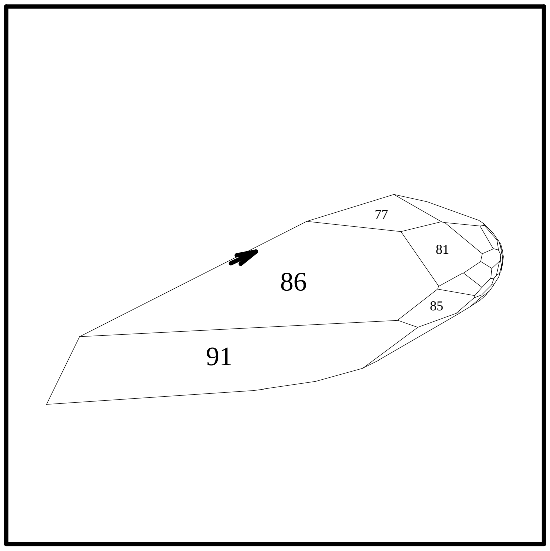

Fig. 92: primal projective iter 16, outside

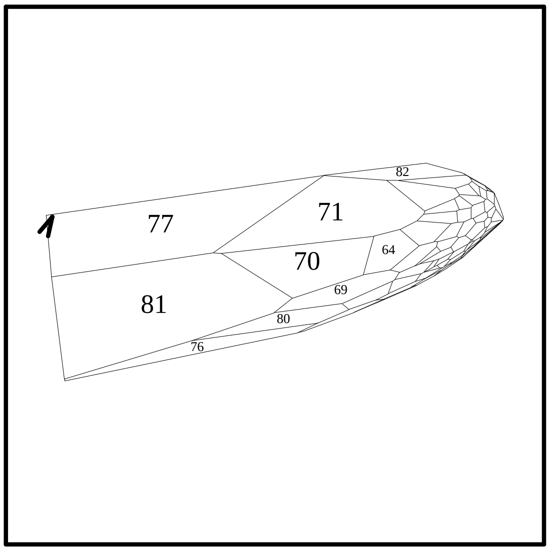

Fig. 93: primal projective iter 17, outside

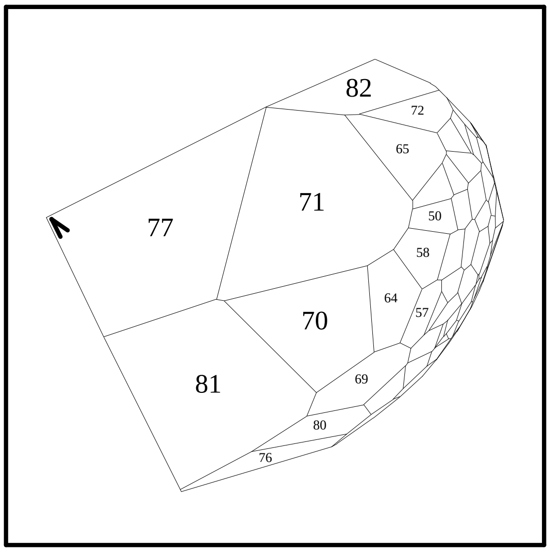

Fig. 94: primal projective iter 18, outside

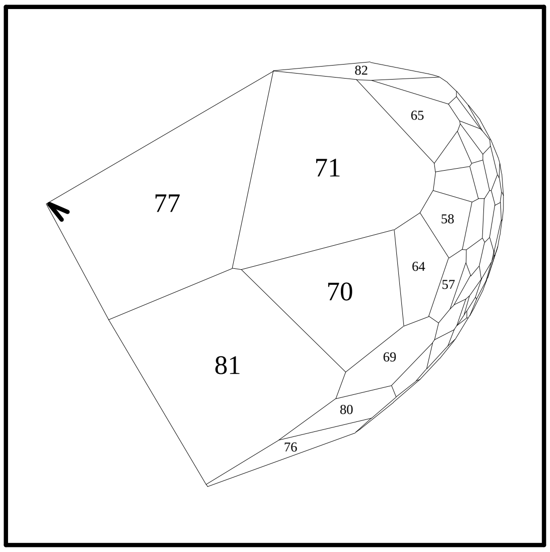

Fig. 95: primal projective iter 19, outside

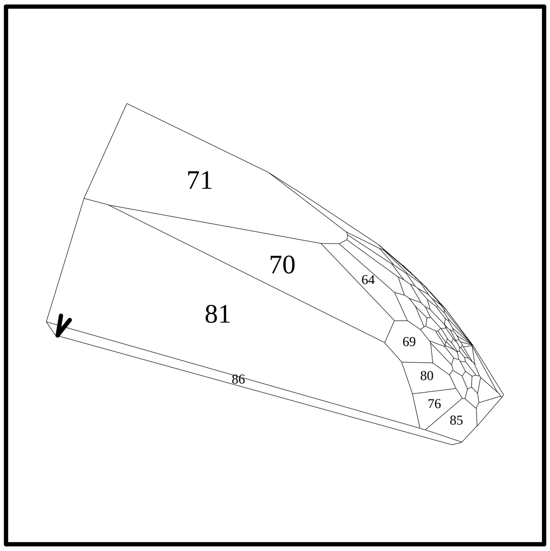

Fig. 96: primal projective iter 20, outside

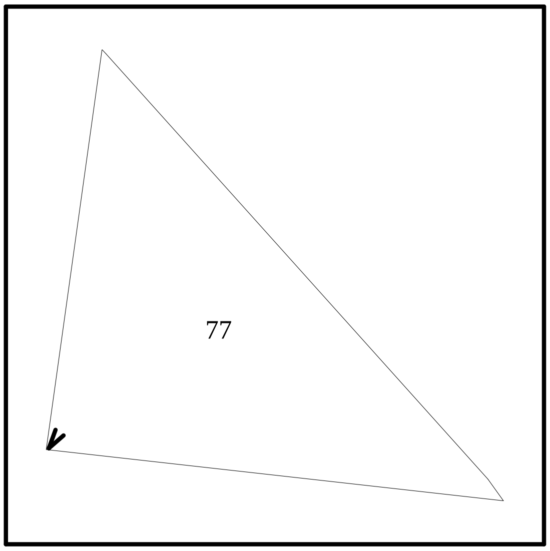

Fig. 97: primal projective iter 21, outside

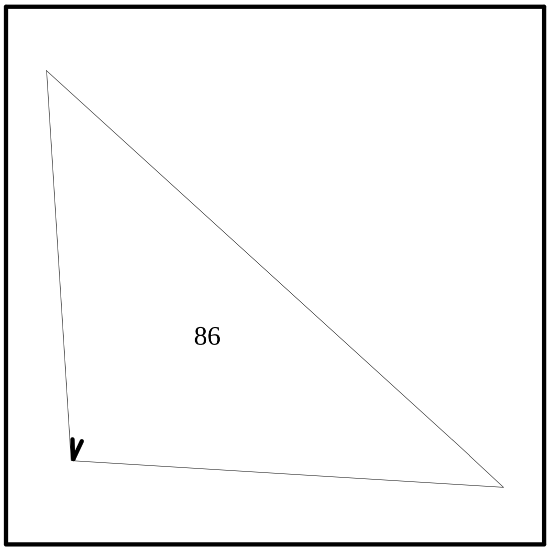

Fig. 98: primal projective iter 22, outside

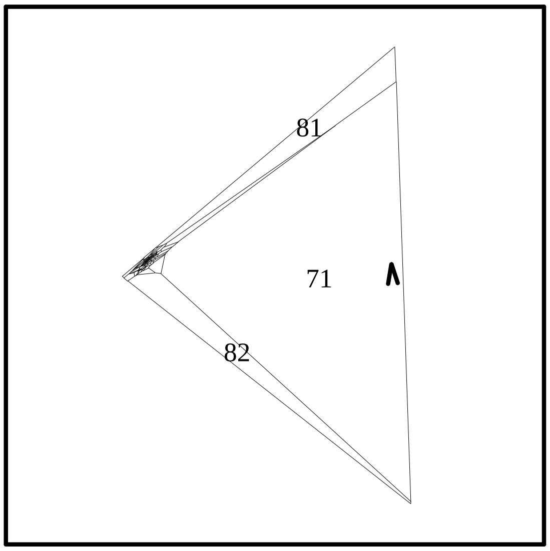

Fig. 99: iter 22, outside, $d = (0,-1,0)$

Figures 100–135 are analogous to figures 48–73: they show views of the projectively transformed polytope in direction $d = (-1,0,0)$; the even numbered figures are from the center, the odd numbered ones from the center plus (0.1,0,0) — cutaway views from outside the polytope.

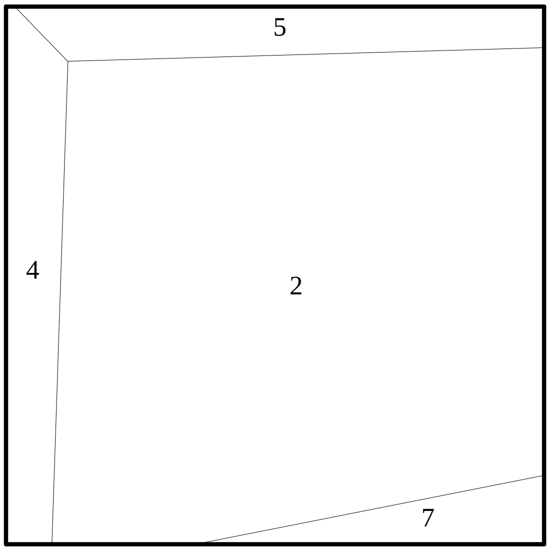

Fig. 100: pr. proj. it. 5, *v* = *center*

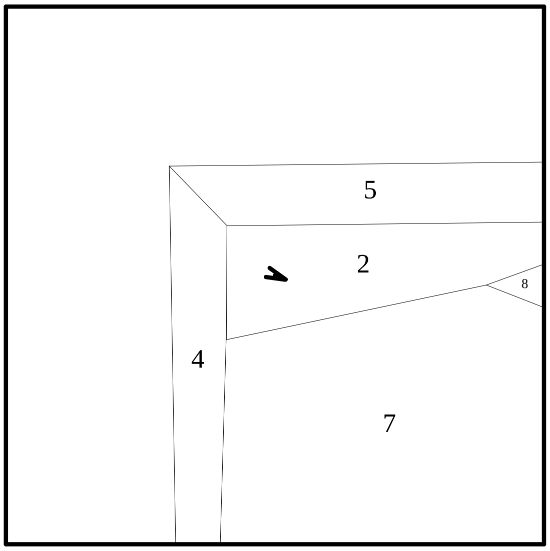

Fig. 101: pr. proj. it. 5, *v* = *center*+(.1,0,0)

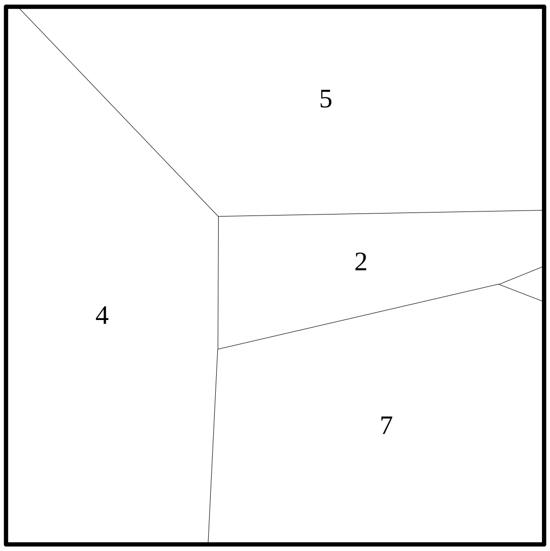

Fig. 102: pr. proj. it. 6, *v* = *center*

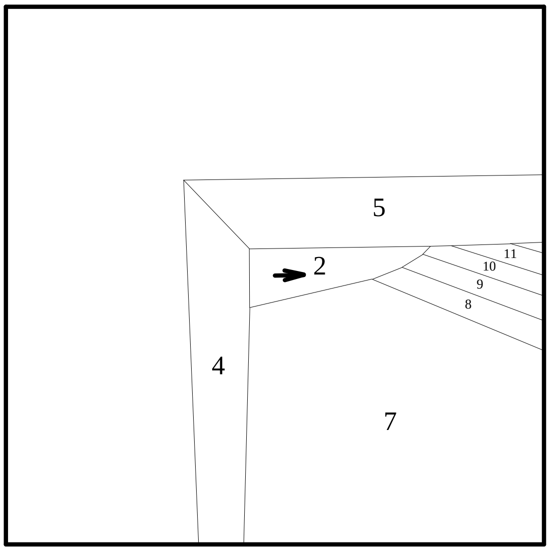

Fig. 103: pr. proj. it. 6, *v* = *center*+(.1,0,0)

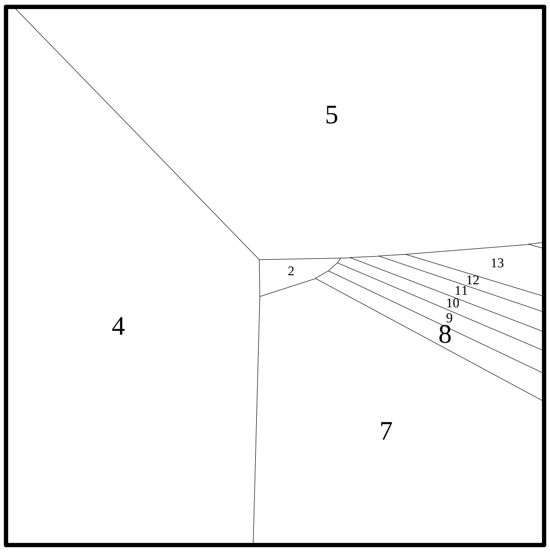

Fig. 104: pr. proj. it. 7, *v* = *center*

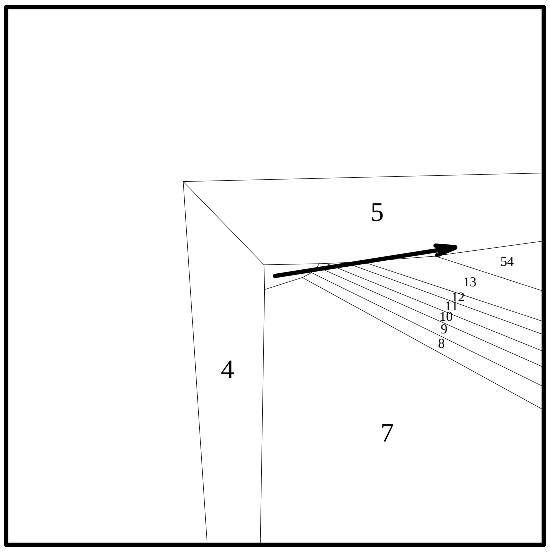

Fig. 105: pr. proj. it. 7, *v* = *center*+(.1,0,0)

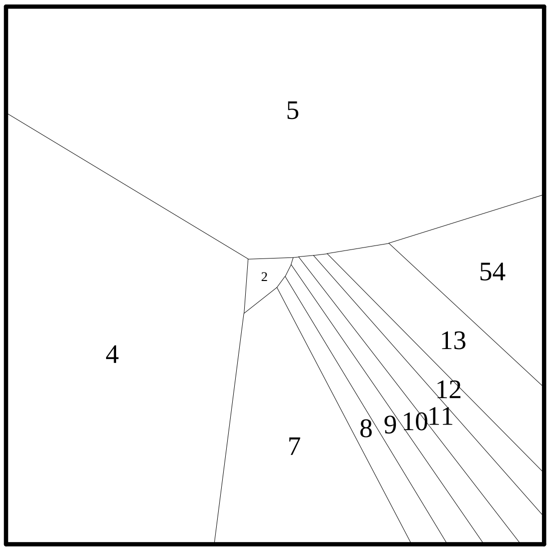

Fig. 106: pr. proj. it. 8, *v* = *center*

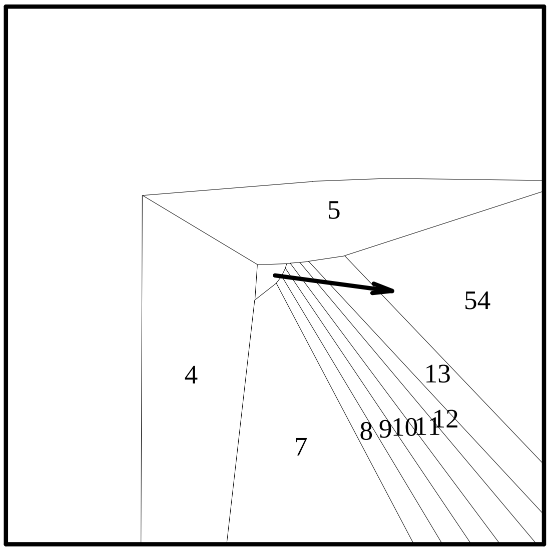

Fig. 107: pr. proj. it. 8, *v* = *center*+(.1,0,0)

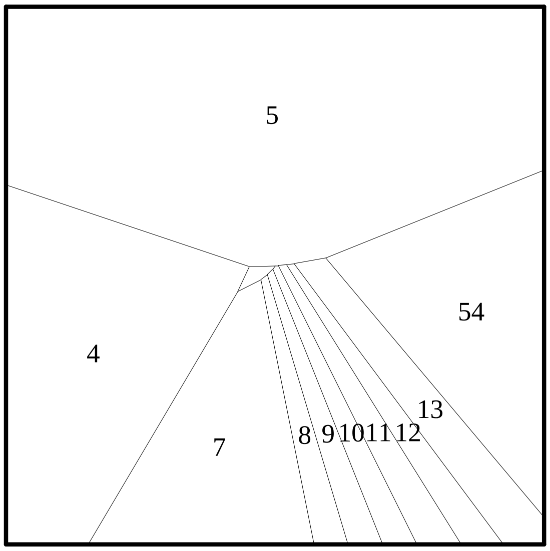

Fig. 108: pr. proj. it. 9, *v* = *center*

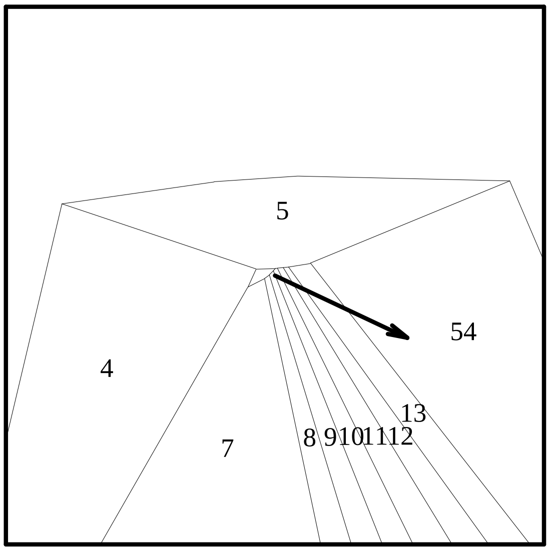

Fig. 109: pr. proj. it. 9, *v* = *center*+(.1,0,0)

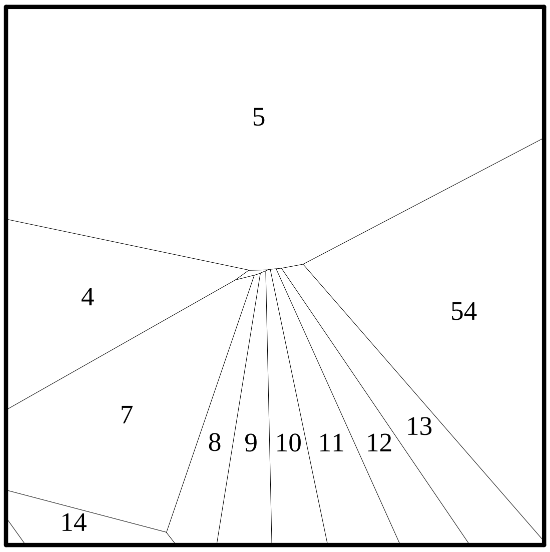

Fig. 110: pr. proj. it. 10, *v* = *center*

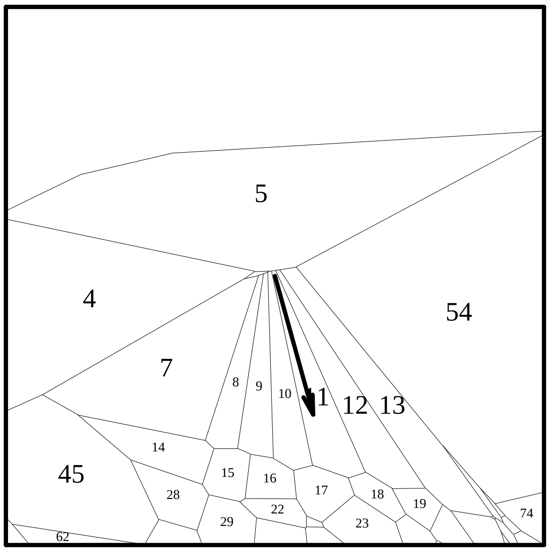

Fig. 111: pr. proj. it. 10, *v* = *center*+(.1,0,0)

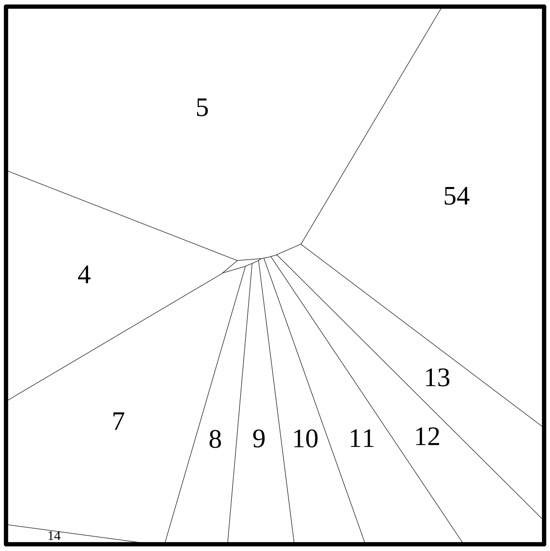

Fig. 112: pr. proj. it. 11, *v* = *center*

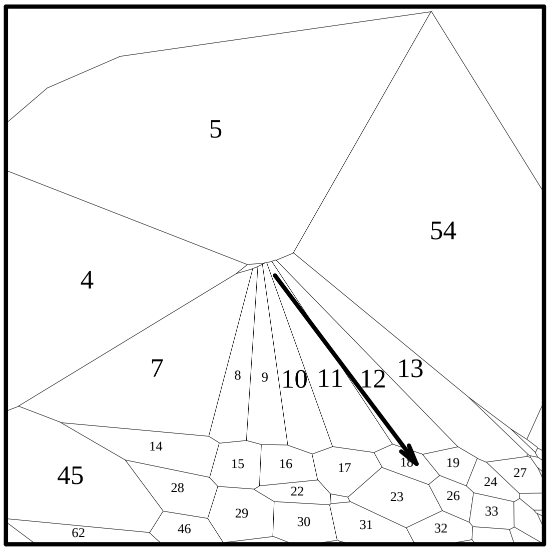

Fig. 113: pr. proj. it. 11, *v* = *center*+(.1,0,0)

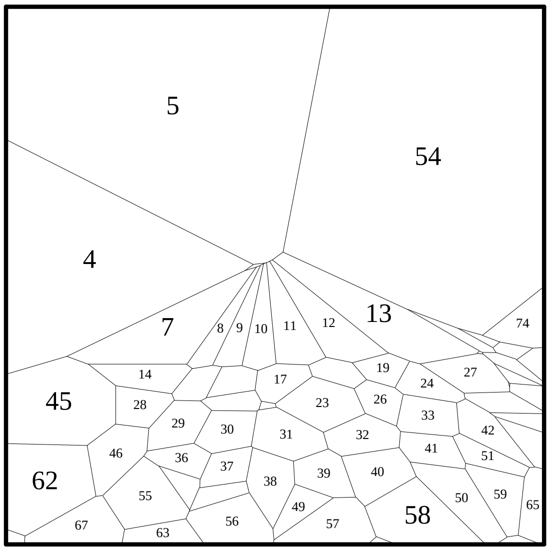

Fig. 114: pr. proj. it. 12, *v* = *center*

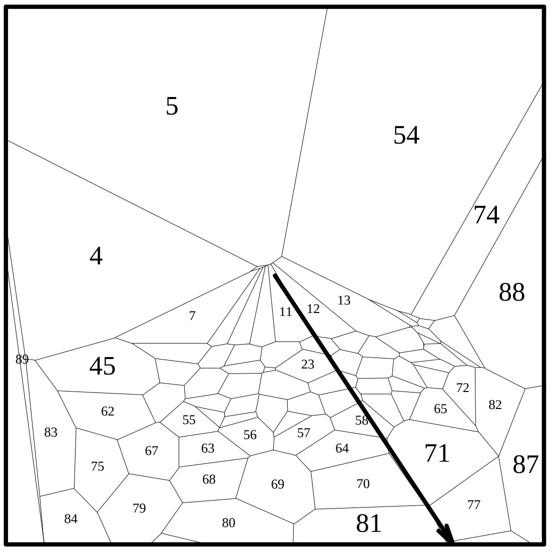

Fig. 115: pr. proj. it. 12, *v* = *center*+(.1,0,0)

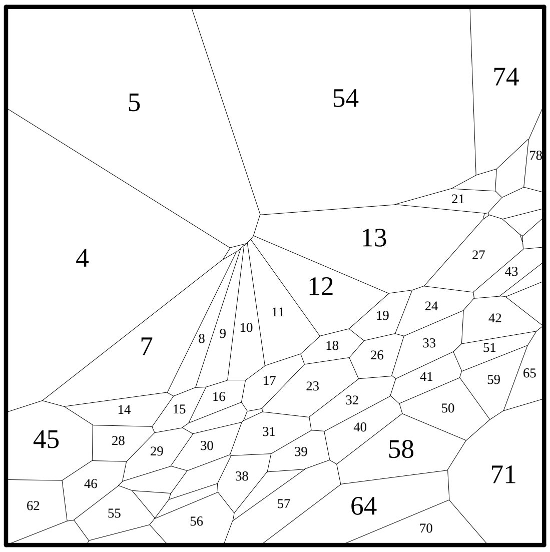

Fig. 116: pr. proj. it. 13, *v* = *center*

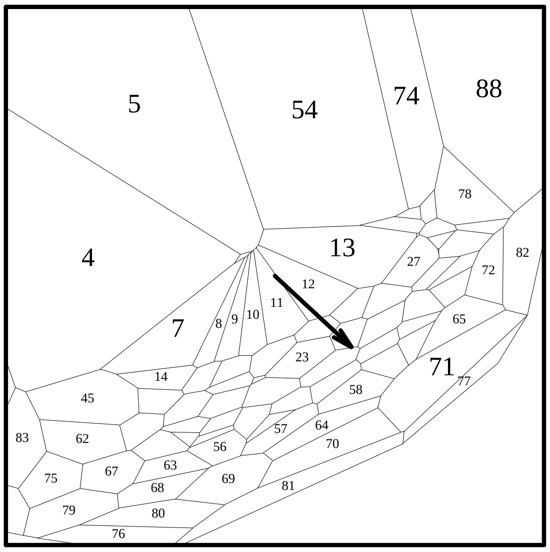

Fig. 117: pr. proj. it. 13, *v* = *center*+(.1,0,0)

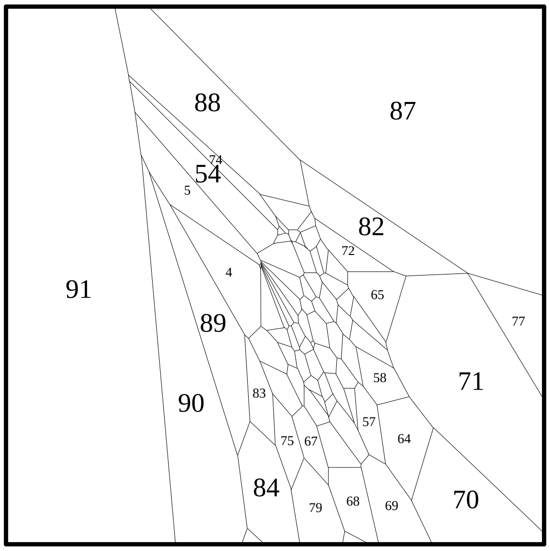

Fig. 118: pr. proj. it. 14, *v* = *center*

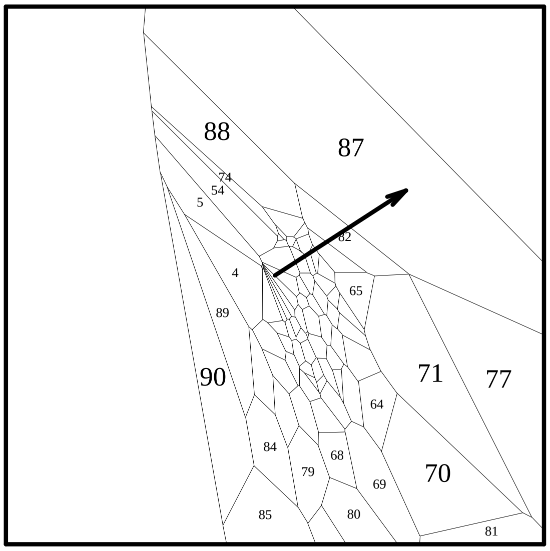

Fig. 119: pr. proj. it. 14, *v* = *center*+(.1,0,0)

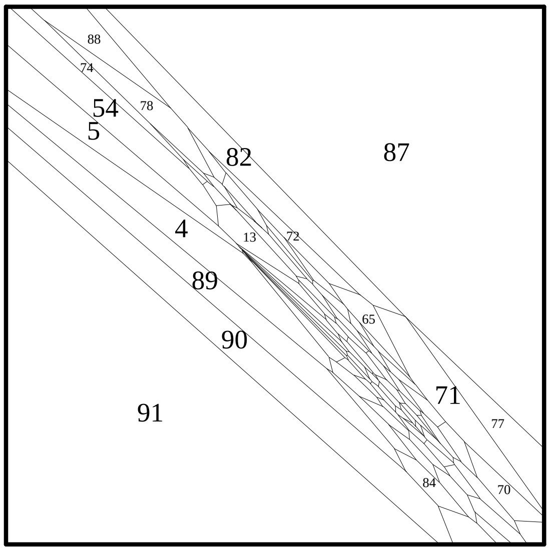

Fig. 120: pr. proj. it. 15, *v* = *center*

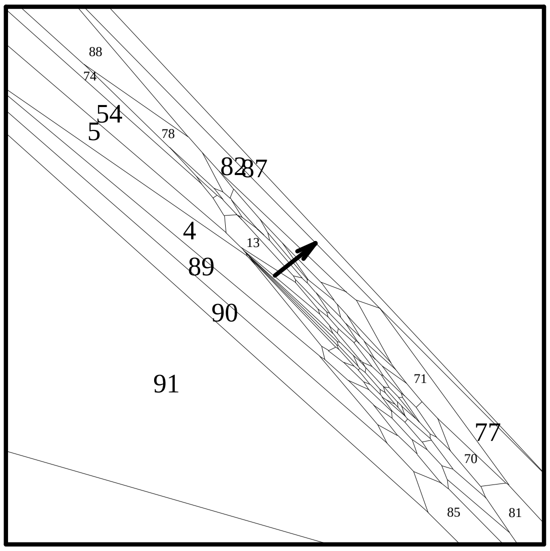

Fig. 121: pr. proj. it. 15, *v* = *center*+(.1,0,0)

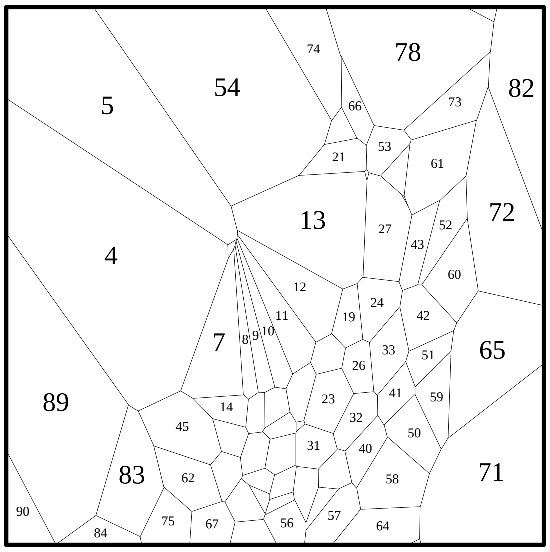

Fig. 122: pr. proj. it. 16, *v* = *center*

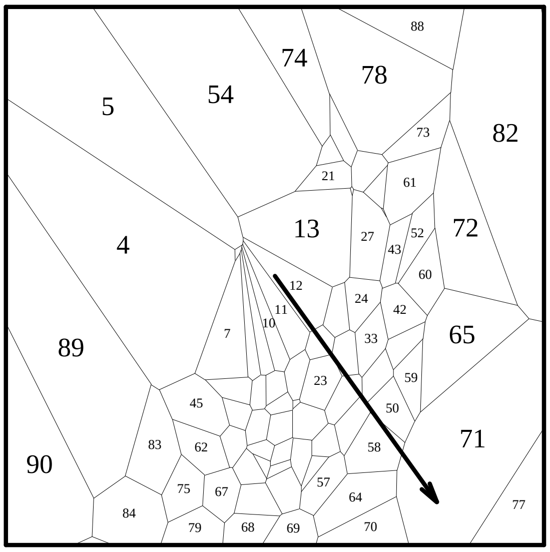

Fig. 123: pr. proj. it. 16, *v* = *center*+(.1,0,0)

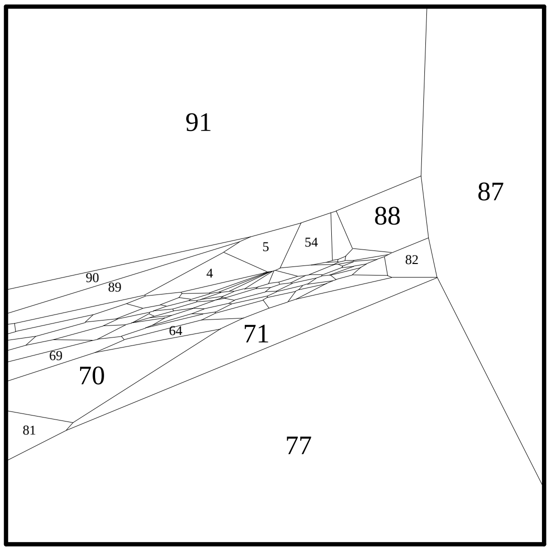

Fig. 124: pr. proj. it. 17, *v* = *center*

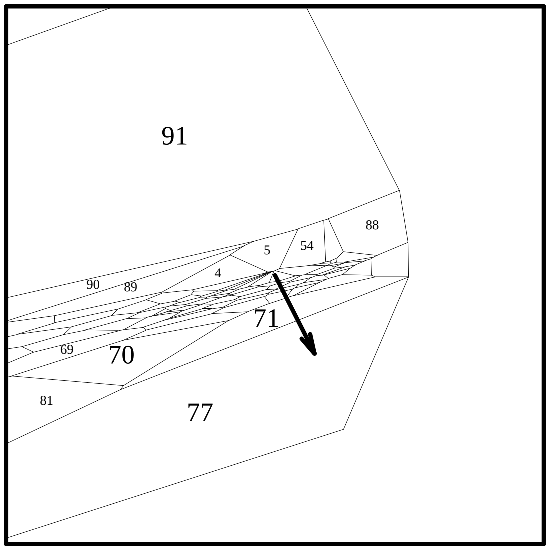

Fig. 125: pr. proj. it. 17, *v* = *center*+(.1,0,0)

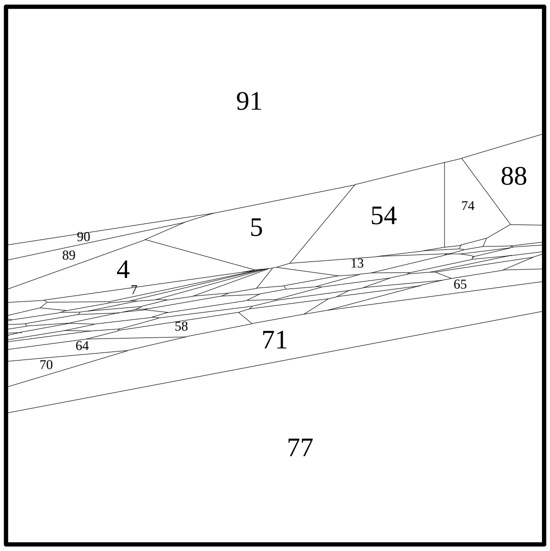

Fig. 126: pr. proj. it. 18, *v* = *center*

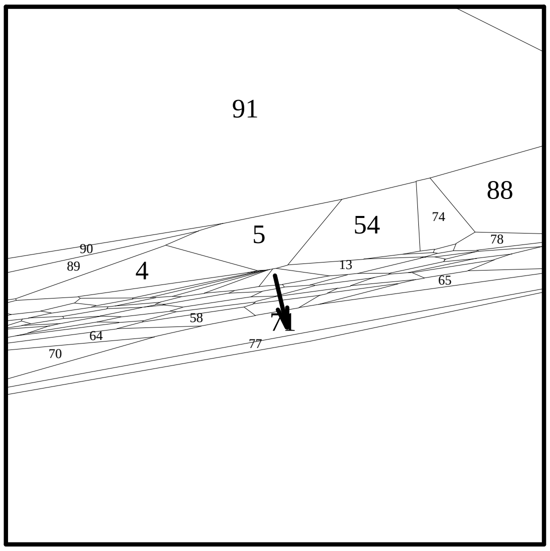

Fig. 127: pr. proj. it. 18, *v* = *center*+(.1,0,0)

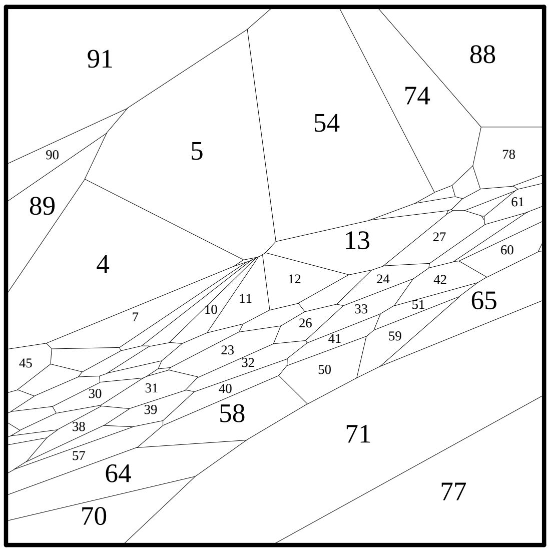

Fig. 128: pr. proj. it. 19, *v* = *center*

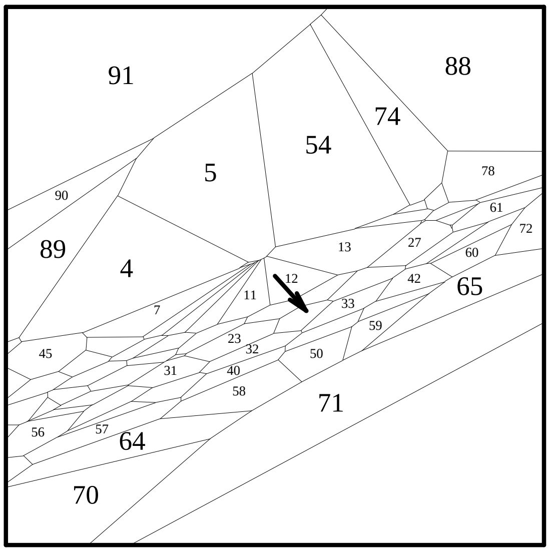

Fig. 129: pr. proj. it. 19, *v* = *center*+(.1,0,0)

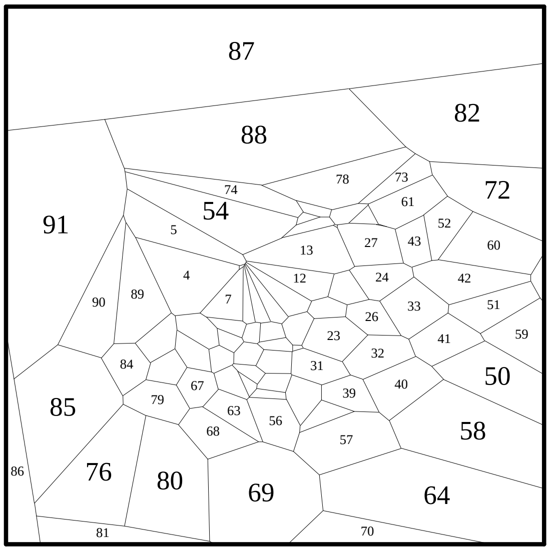

Fig. 130: pr. proj. it. 20, *v* = *center*

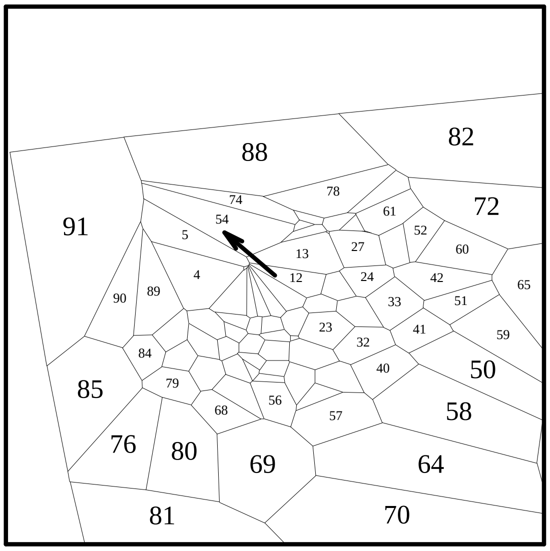

Fig. 131: pr. proj. it. 20, *v* = *center*+(.1,0,0)

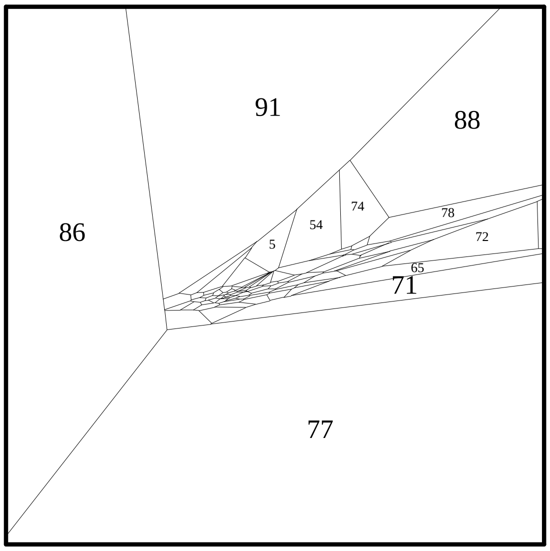

Fig. 132: pr. proj. it. 21, *v* = *center*

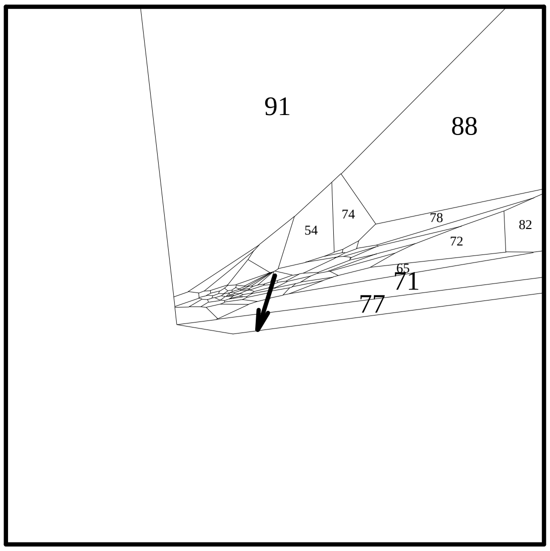

Fig. 133: pr. proj. it. 21, *v* = *center*+(.1,0,0)

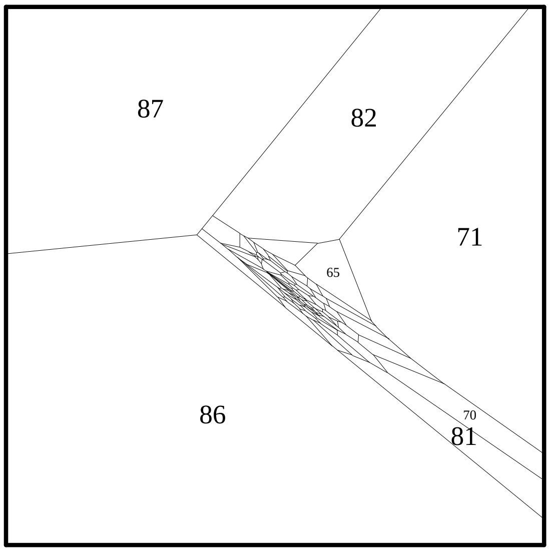

Fig. 134: pr. proj. it. 22, *v* = *center*

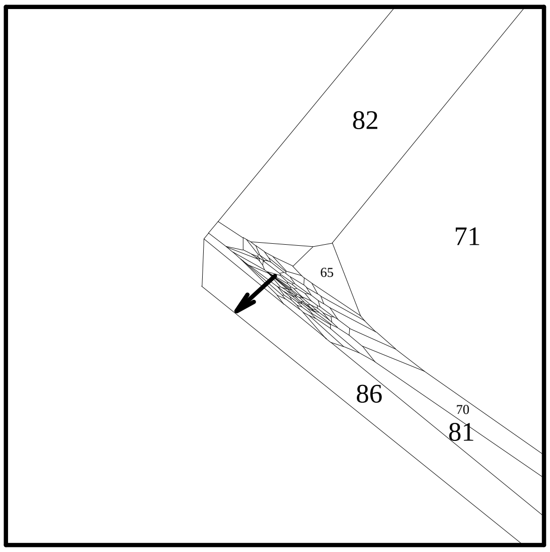

Fig. 135: pr. proj. it. 22, *v* = *center*+(.1,0,0)

Views from ''Natural Variables'' — the *Record* Pictures

In making the pictures on pages 6 and 7 of the *Record* article [9], I used what I thought to be a ''natural'' F in (16) — rather than one with orthonormal columns; I hadn't yet thought about the invariance properties that come with the latter F. (The *Record* pictures correspond to at least two different linear programming problems; I had asked that only one be used and had been told my request would be honored, but forgot to double check this point when the proofs came around.)

The remaining figures depict the same run of the primal projective algorithm as do figures 74–135, but from the perspective of the variables in which I originally conceived the problem (i.e., using the same F as for the *Record* pictures). My idea was to massage the constraints (18) and (7) into the form (16). In more detail, (7a) with A as in (18) is of the form $B\hat{x} = e_{m+1}$, where

$$B = \begin{bmatrix} A^{\circ\mathrm{T}} D_1 & D_2 & \mathbf{0} & \mathbf{0} & -b \\ D_1 & \mathbf{0} & D_3 & \mathbf{0} & -u_1 \\ \mathbf{0} & D_2 & \mathbf{0} & D_4 & -u_2 \\ e^{\mathrm{T}} & e^{\mathrm{T}} & e^{\mathrm{T}} & e^{\mathrm{T}} & 1 \end{bmatrix}. \tag{19}$$

Here D is the diagonal matrix $D = \operatorname{diag}(D_1, D_2, D_3, D_4)$, with $D_1, D_3 \in \mathbb{R}^{3\times 3}$ and $D_2, D_4 \in \mathbb{R}^{m\times m}$; $u_1 \in \mathbb{R}^3$ and $u_2 \in \mathbb{R}^m$ are upper bound vectors. By doing row operations on B, we may first eliminate the (3,2) block of (19), then the (4,2), (4,3), and (4,4) blocks, and finally the (1,5), (2,5), and (3,5) blocks to obtain F in (16); the f of (16) comes from doing the corresponding row operations to the right-hand side e_{m+1}. With this F and f, we get interesting views by looking forwards and backwards in the direction of the current step. (When F is orthogonal, such forward and backward view directions swing so wildly that it is hard to make sense out of the resulting pictures.) Figures 136–177 show these views for all 21 phase-2 iterations; the even-numbered figures are forward views, and the odd-numbered ones are corresponding backward views, all from the center of the projectively transformed polytope. In the first three iterations the view changes rapidly. Then the unimportant constraints — those not binding at the solution — are pushed further and further behind, until they are no longer visible in the final two iterations.

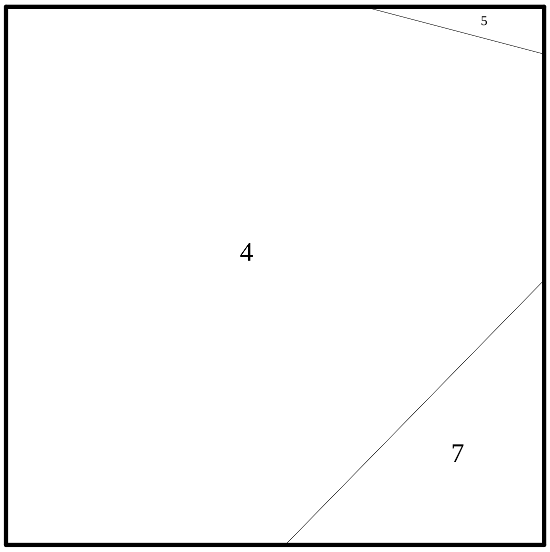

Fig. 136: pr.p.it. 2, d = *ahead*, nat. vars.

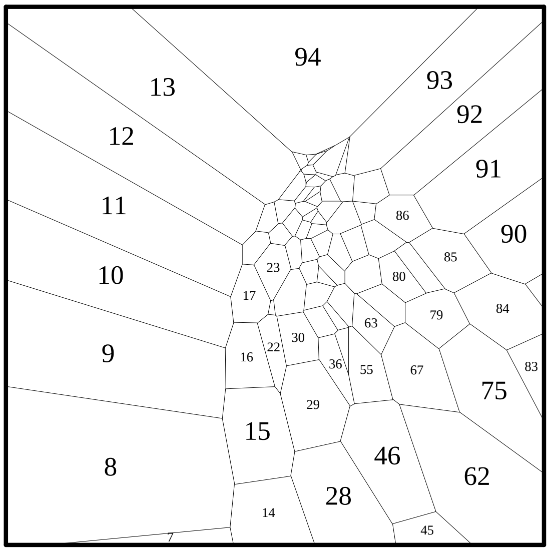

Fig. 137: pr.p.it. 2, d = *behind*, nat. vars.

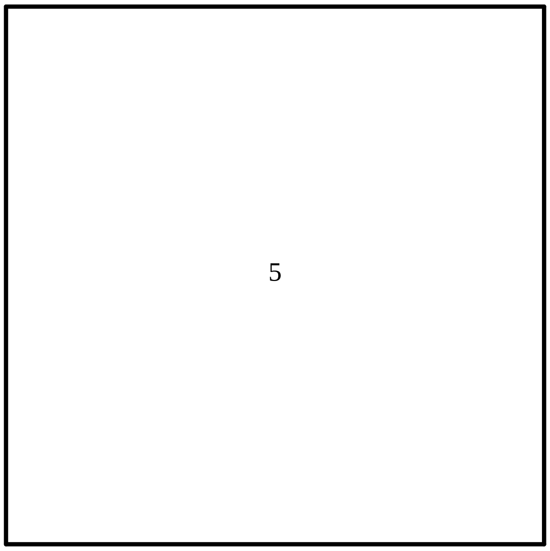

Fig. 138: pr.p.it. 3, d = *ahead*, nat. vars.

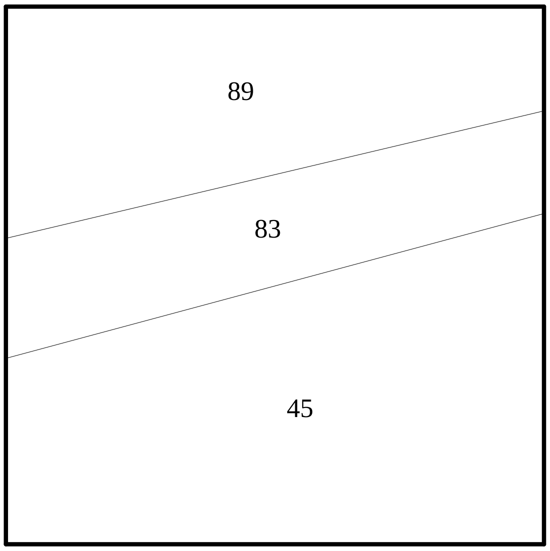

Fig. 139: pr.p.it. 3, d = *behind*, nat. vars.

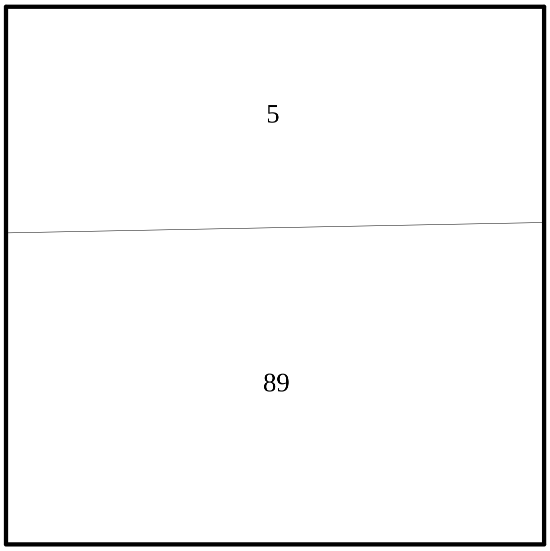

Fig. 140: pr.p.it. 4, d = *ahead*, nat. vars.

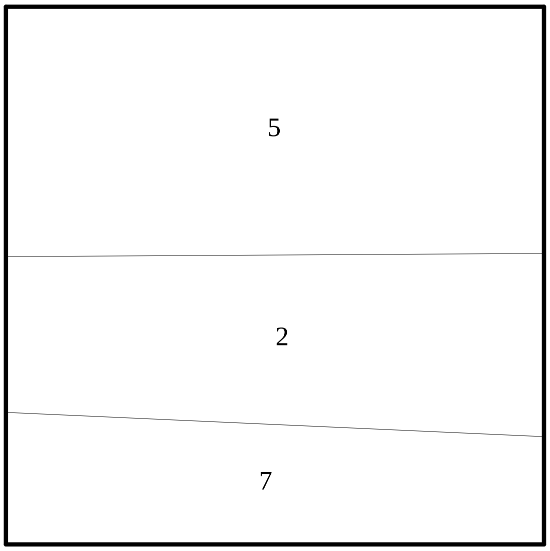

Fig. 141: pr.p.it. 4, d = *behind*, nat. vars.

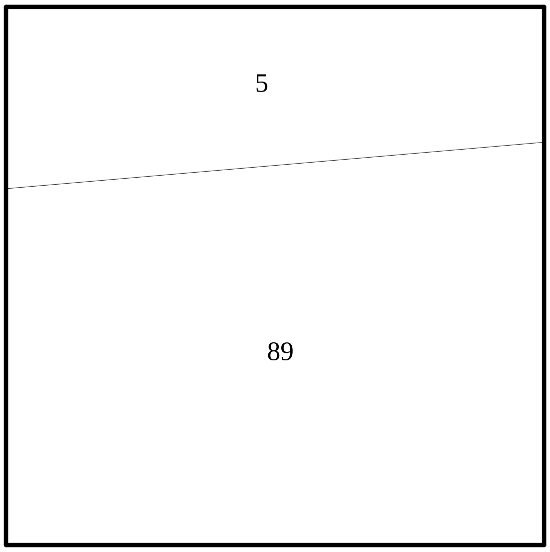

Fig. 142: pr.p.it. 5, d = *ahead*, nat. vars.

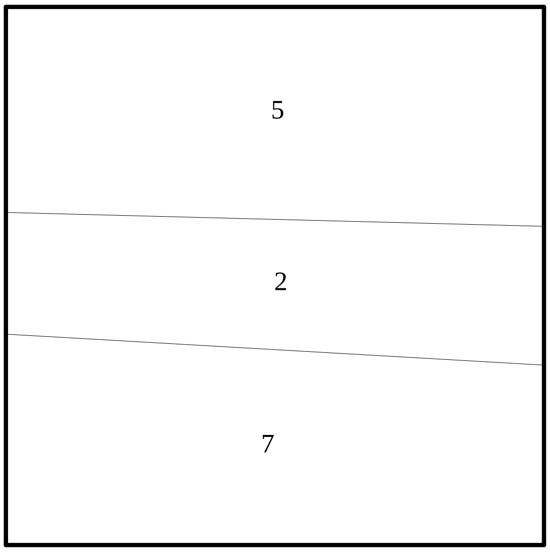

Fig. 143: pr.p.it. 5, d = *behind*, nat. vars.

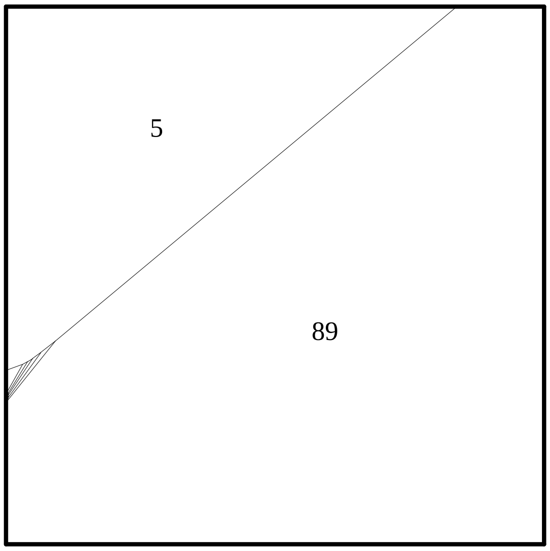

Fig. 144: pr.p.it. 6, d = *ahead*, nat. vars.

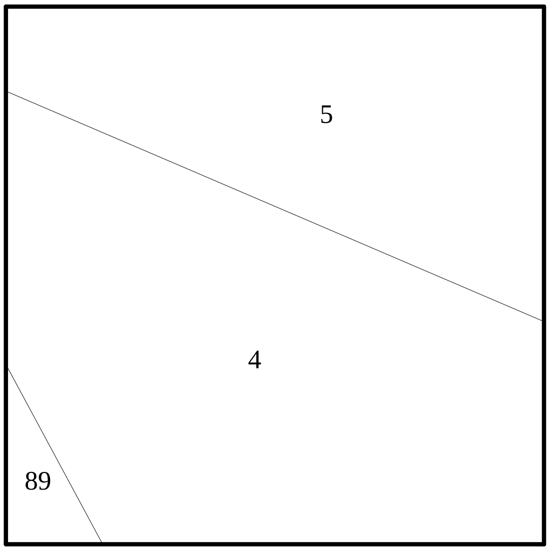

Fig. 145: pr.p.it. 6, d = *behind*, nat. vars.

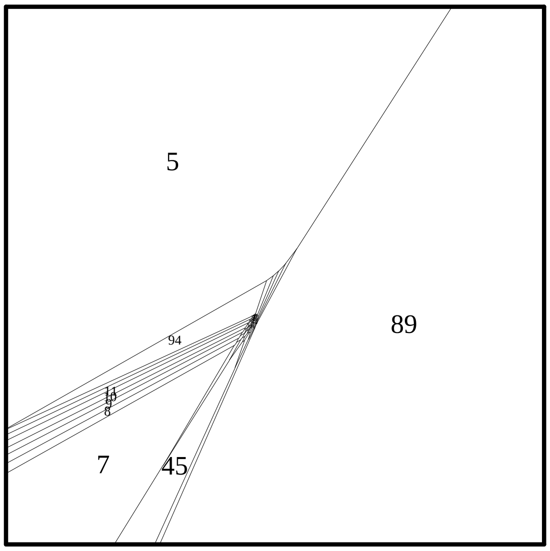

Fig. 146: pr.p.it. 7, d = *ahead*, nat. vars.

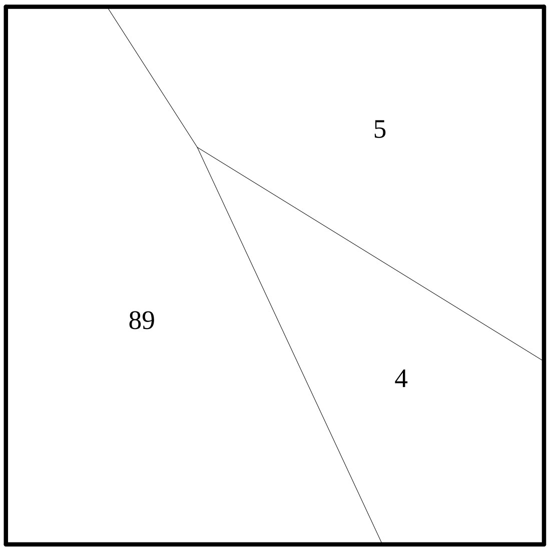

Fig. 147: pr.p.it. 7, d = *behind*, nat. vars.

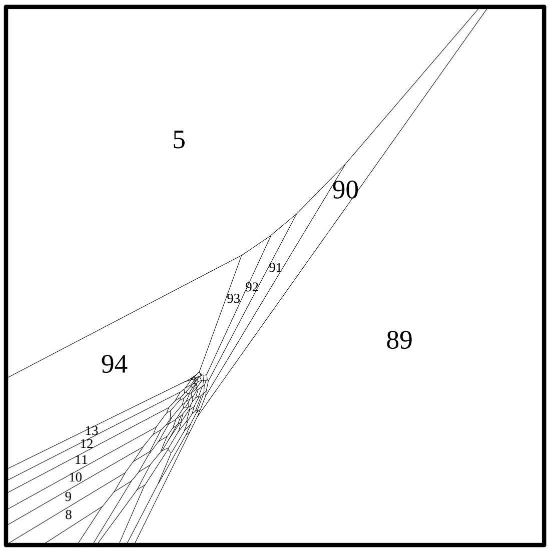

Fig. 148: pr.p.it. 8, d = *ahead*, nat. vars.

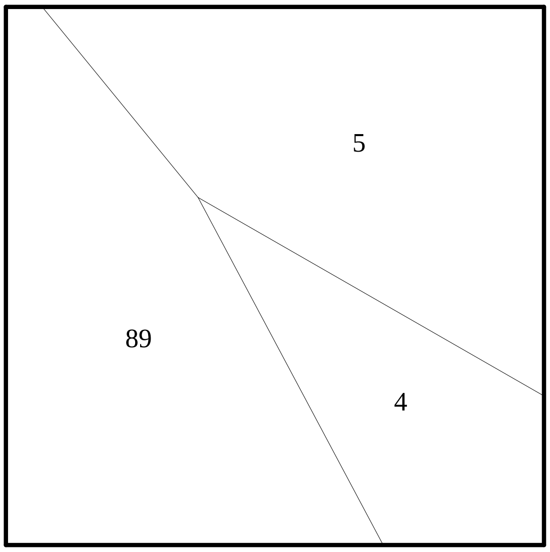

Fig. 149: pr.p.it. 8, d = *behind*, nat. vars.

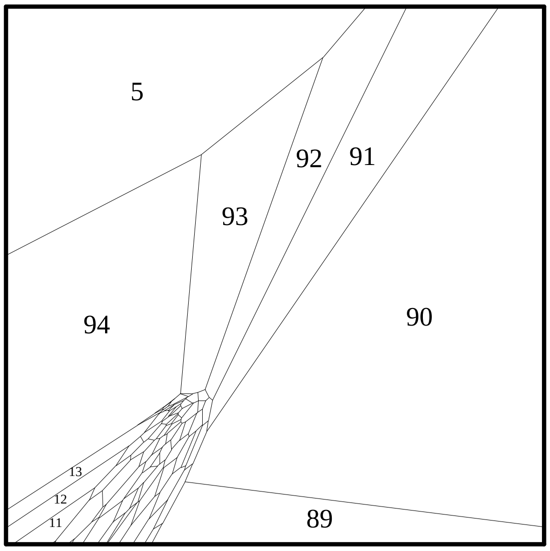

Fig. 150: pr.p.it. 9, d = *ahead*, nat. vars.

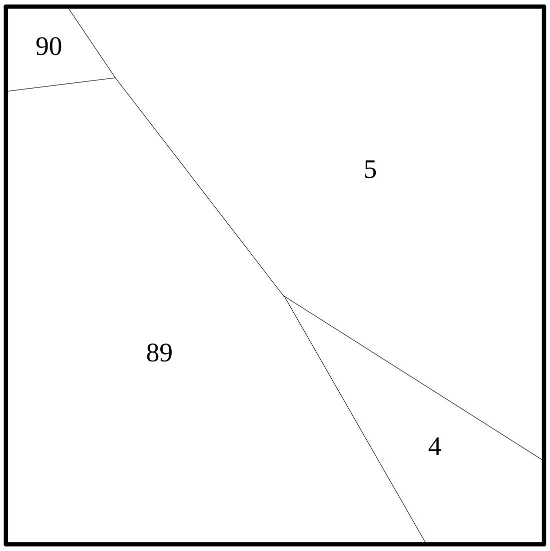

Fig. 151: pr.p.it. 9, d = *behind*, nat. vars.

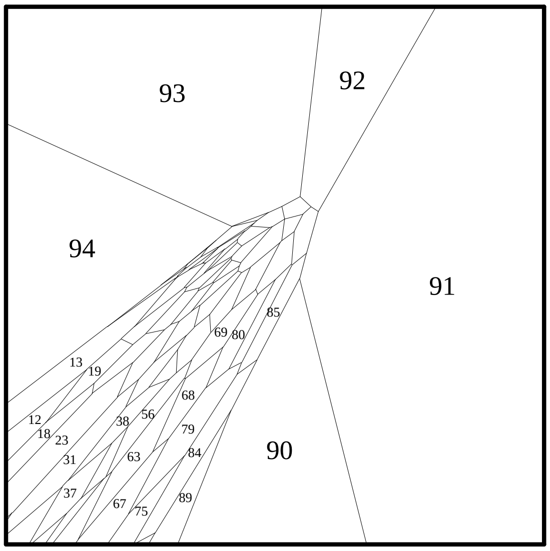

Fig. 152: pr.p.it. 10, d = *ahead*, nat. vars.

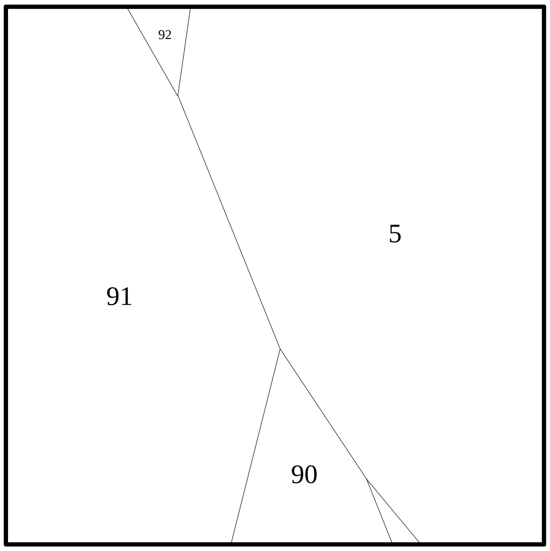

Fig. 153: pr.p.it. 10, d = *behind*, nat. vars.

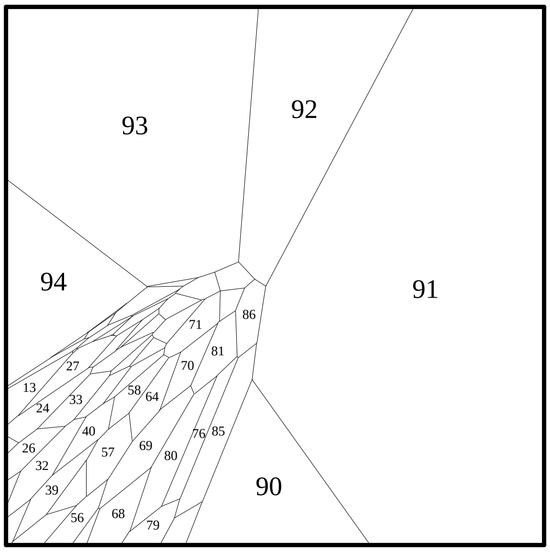

Fig. 154: pr.p.it. 11, d = *ahead*, nat. vars.

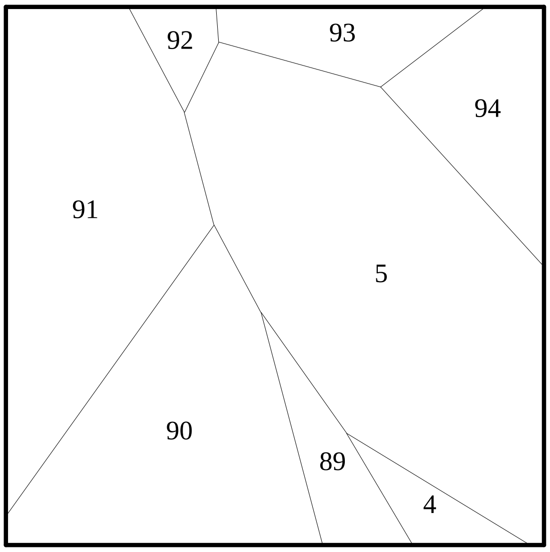

Fig. 155: pr.p.it. 11, d = *behind*, nat. vars.

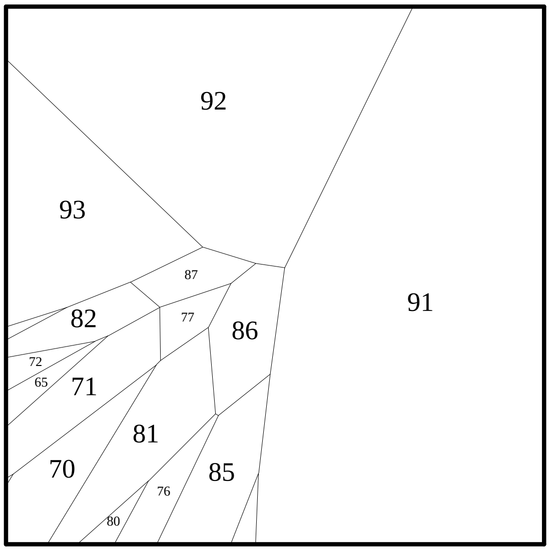

Fig. 154: pr.p.it. 12, *d* = *ahead*, nat. vars.

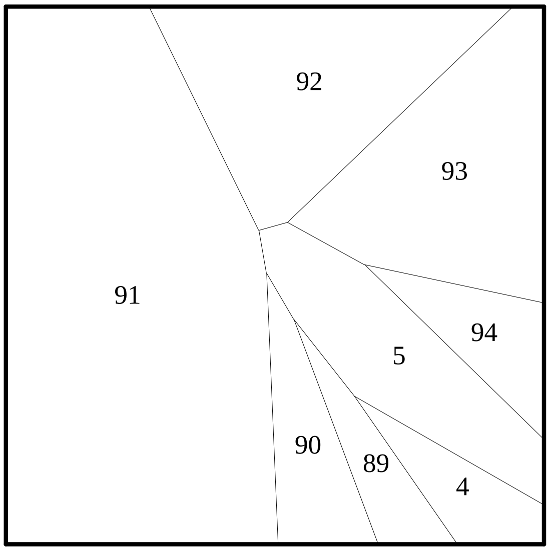

Fig. 155: pr.p.it. 12, *d* = *behind*, nat. vars.

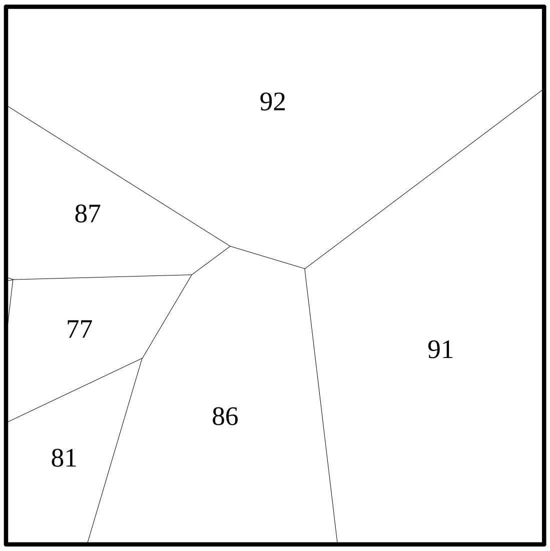

Fig. 158: pr.p.it. 13, *d* = *ahead*, nat. vars.

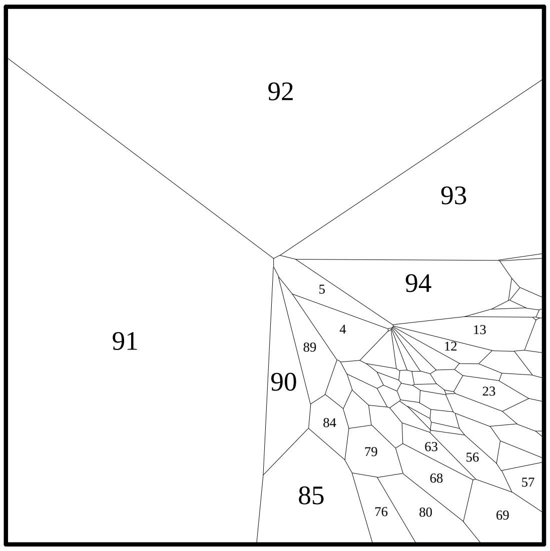

Fig. 159: pr.p.it. 13, *d* = *behind*, nat. vars.

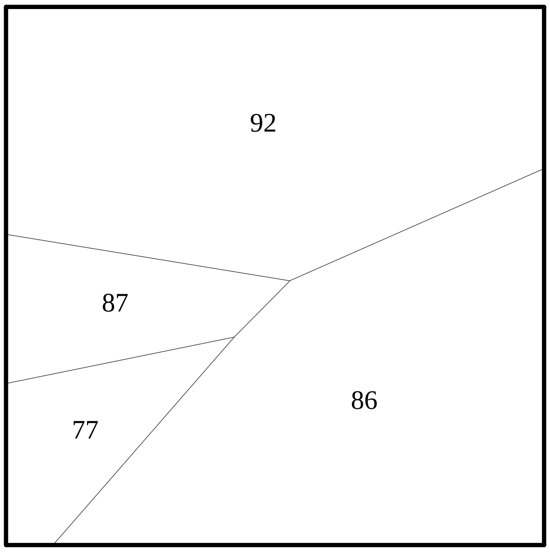

Fig. 160: pr.p.it. 14, *d* = *ahead*, nat. vars.

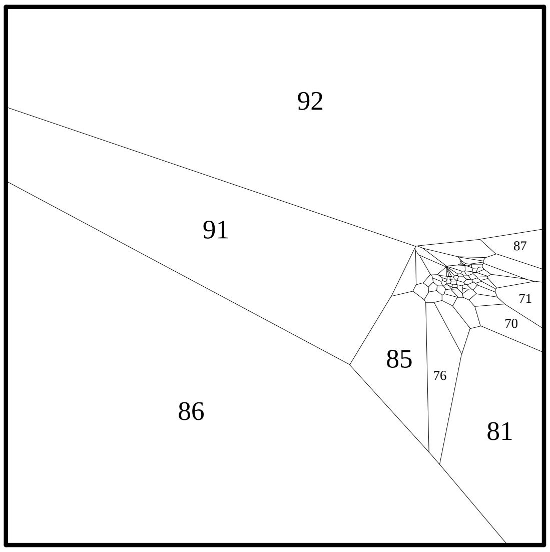

Fig. 161: pr.p.it. 14, *d* = *behind*, nat. vars.

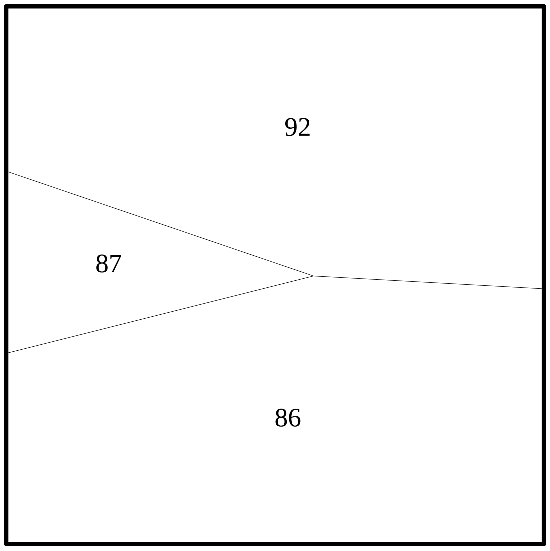

Fig. 162: pr.p.it. 15, d = *ahead*, nat. vars.

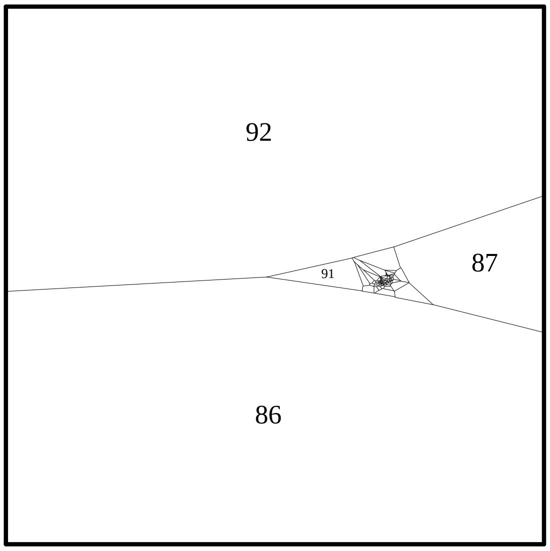

Fig. 163: pr.p.it. 15, d = *behind*, nat. vars.

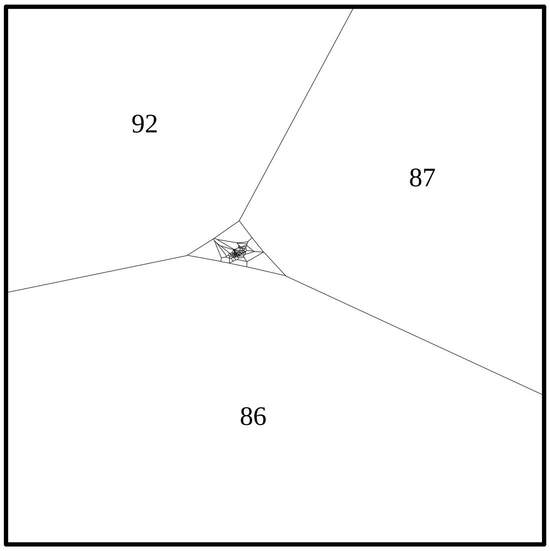

Fig. 164: pr.p.it. 16, d = *ahead*, nat. vars.

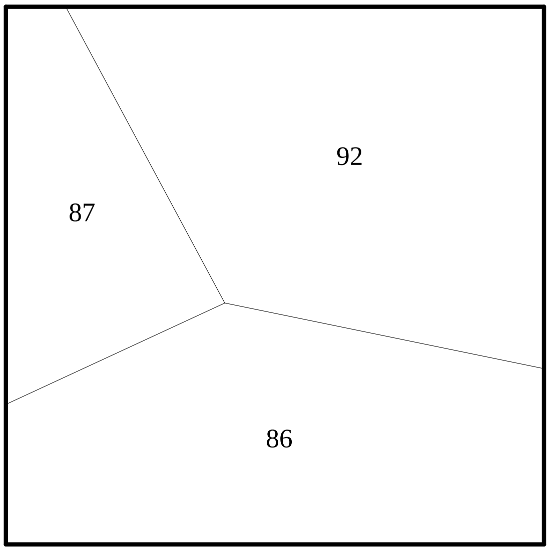

Fig. 165: pr.p.it. 16, d = *behind*, nat. vars.

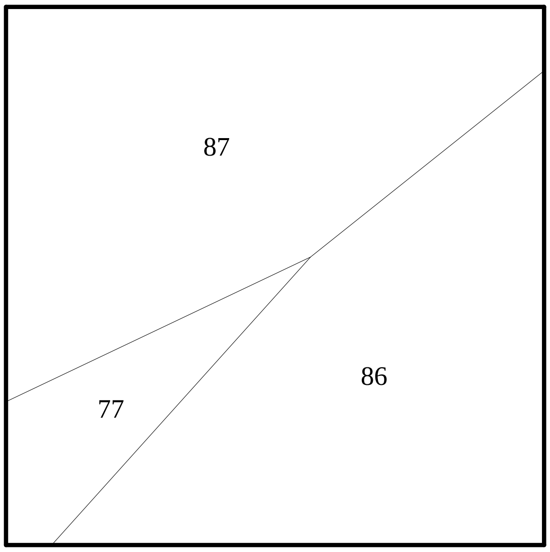

Fig. 166: pr.p.it. 17, d = *ahead*, nat. vars.

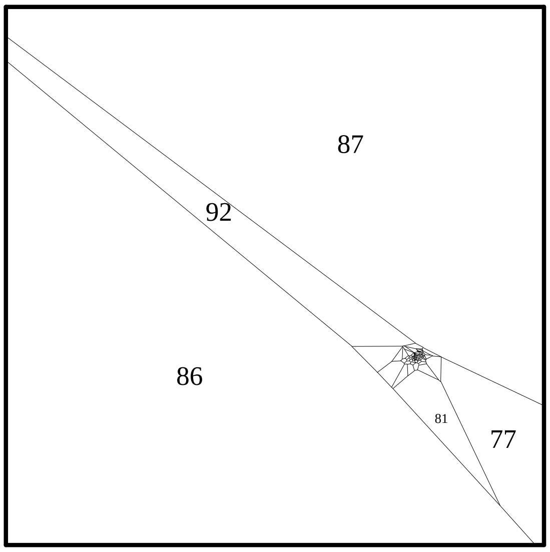

Fig. 167: pr.p.it. 17, d = *behind*, nat. vars.

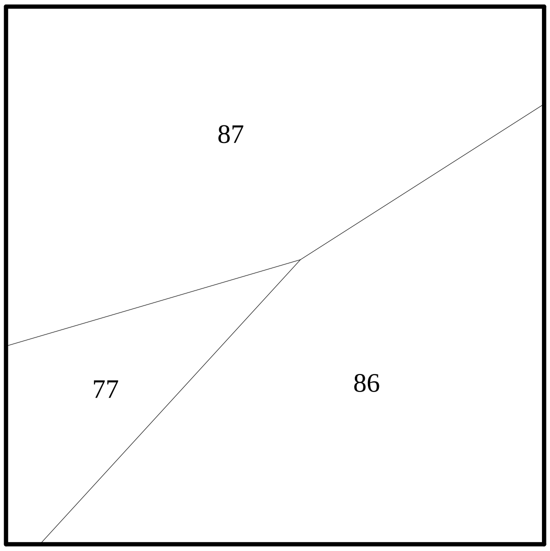

Fig. 168: pr.p.it. 18, d = *ahead*, nat. vars.

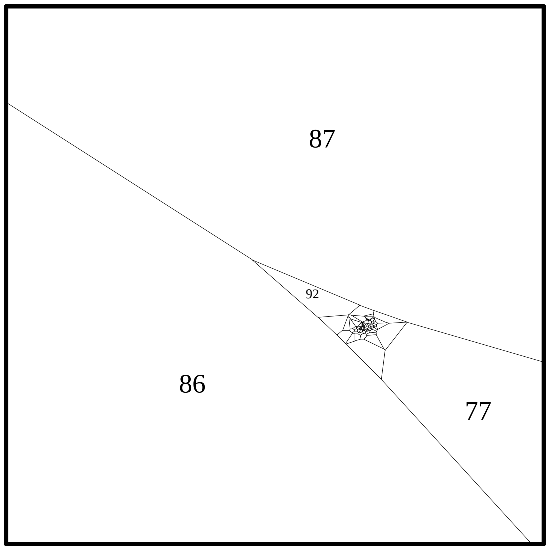

Fig. 169: pr.p.it. 18, d = *behind*, nat. vars.

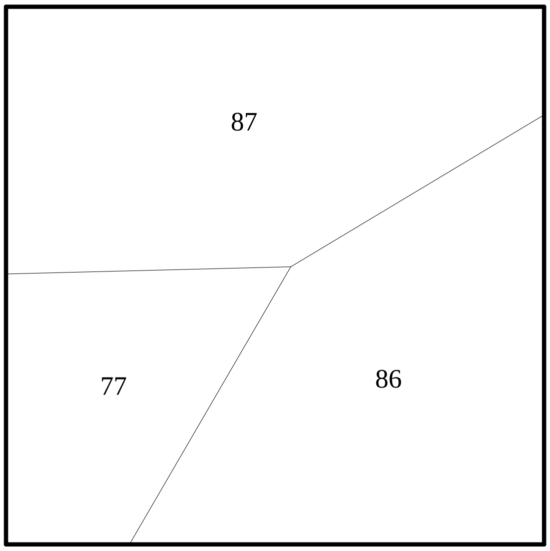

Fig. 170: pr.p.it. 19, d = *ahead*, nat. vars.

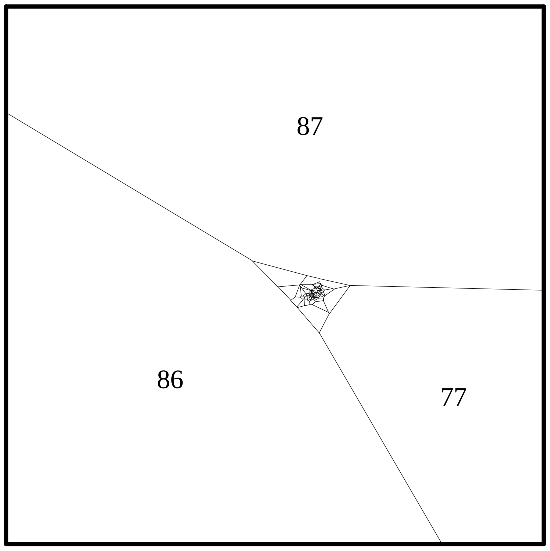

Fig. 171: pr.p.it. 19, d = *behind*, nat. vars.

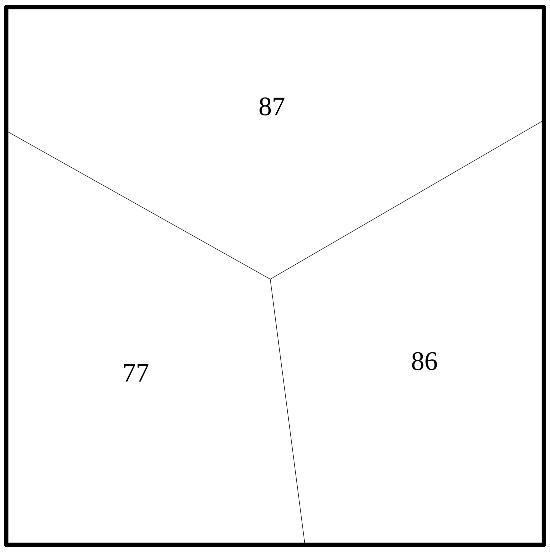

Fig. 172: pr.p.it. 20, d = *ahead*, nat. vars.

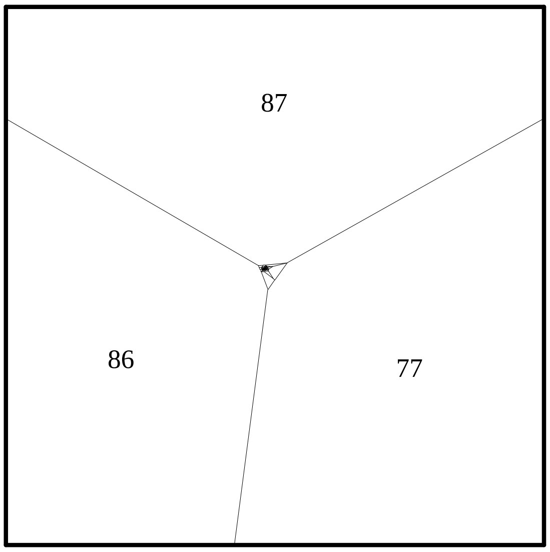

Fig. 173: pr.p.it. 20, d = *behind*, nat. vars.

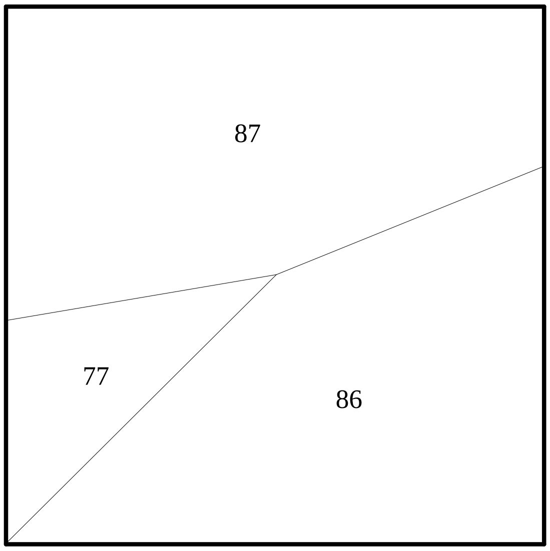

Fig. 174: pr.p.it. 21, d = *ahead*, nat. vars.

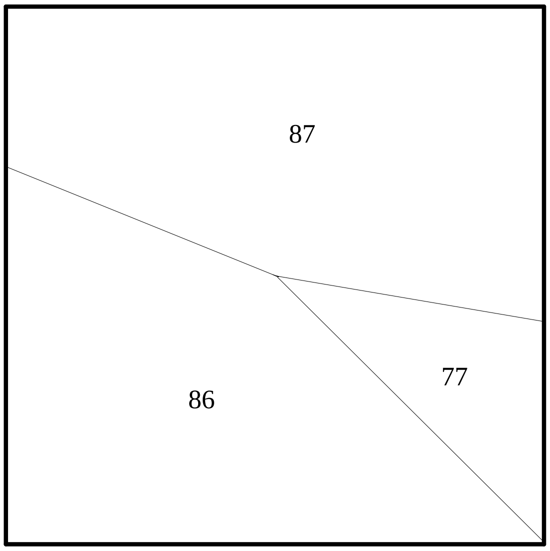

Fig. 175: pr.p.it. 21, d = *behind*, nat. vars.

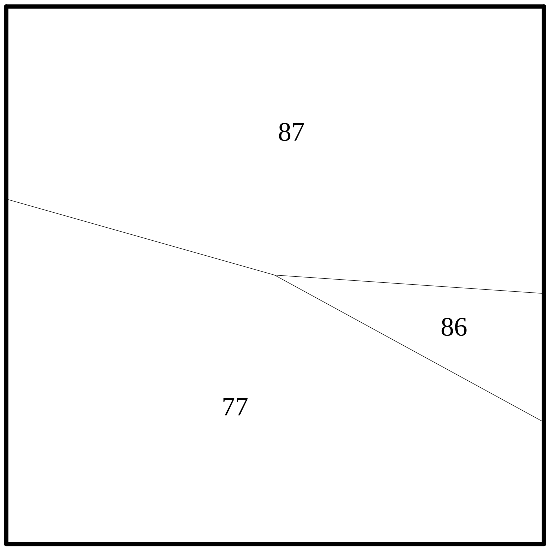

Fig. 176: pr.p.it. 22, d = *ahead*, nat. vars.

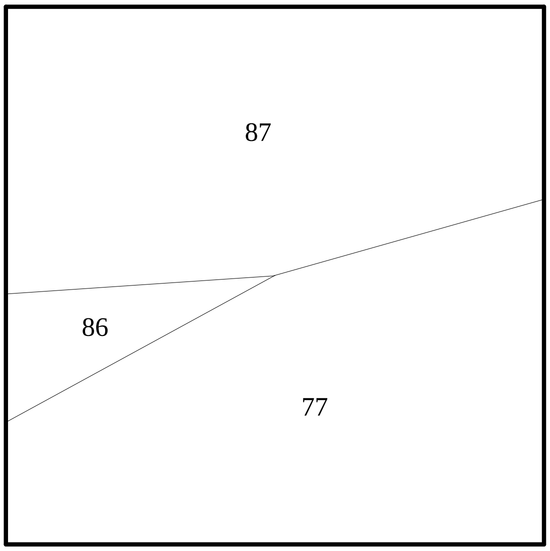

Fig. 177: pr.p.it. 22, d = *behind*, nat. vars.

The view direction in iteration 16 (figures 164–165) differs substantially from that in iterations 15 and 17, despite monotonic reductions of both the linear objective $c^T x$ and the potential function.

Notice how the forward and backward views of the last two iterations appear to be mirror images. This is so because the transformed polytope (as seen by the ‘‘natural’’ variables) is much longer in the (0,1,0) direction than in the (1,0,0) or (0,0,1) directions. Indeed, an outside view, analogous to figures 101–135, of the the transformed polytope from iteration 22 is simply a horizontal line. By turning up the magnification in figures 175 and 177, we can see where all the nonbinding constraints have gone: figure 178 is like figure 175, but with view angle 0.5°, and figure 179 is like figure 177, but with view angle 0.15°.

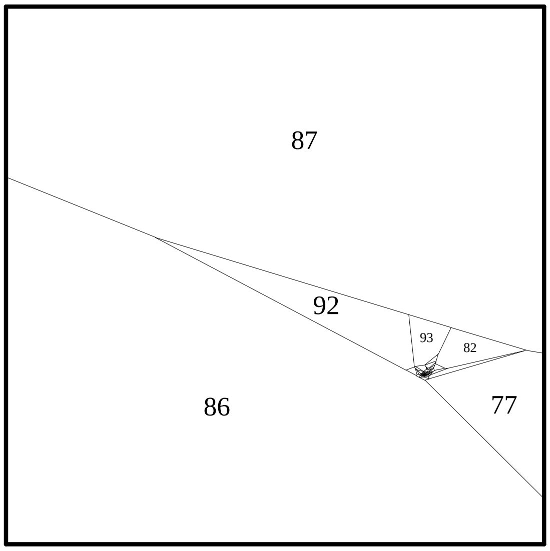

Fig. 178: It. 21, $d = behind$, $v = 0.5°$

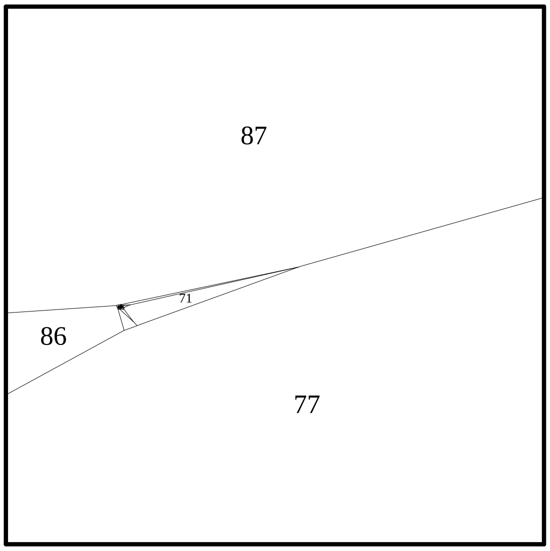

Fig. 179: It. 22, $d = behind$, $v = 0.15°$

Concluding Remarks

The pictures shown above of the dual affine and dual projective algorithms come from solving the same single problem, while those of the primal projective algorithm come from a closely related problem. I have not yet looked at enough other problems to make any claims about how typical the above pictures are (of problems having three degrees of freedom).

The algorithm's eye views shown above — at least those corresponding to an orthogonal F in (16) — change much faster for the affine dual variant than for the projective ones; views of the primal affine variant presumably would behave similarly to those of the dual affine variant, just as the primal and dual projective views are qualitatively similar. (For F orthogonal, the search direction has a nice interpretation: it is the projection, by FF^{T}, of a gradient vector into the viewing space.) The final figures show that the projective views can change more rapidly from the vantage point of a nonorthogonal F.

For any continuous choice of F, we can make the pictures change as slowly as we want by taking small enough steps. This leads naturally to considering differential forms of the algorithms — see [1, 4] — and to making movies in which the transformed polytope deforms smoothly. (Nearly everyone who sees the above pictures suggests making a movie. I would like to do so when time permits. And I look forward to some future time when we'll all have workstations powerful enough to display such a movie in real time.)

Outside views of the transformed feasible regions for the primal and dual projective variants visually support the well-known fact that the transformed polytopes have inscribed and circumscribed spheres whose diameters have a bounded ratio.

Only the first few steps are long enough to be seen on the plots of the solution paths (figures 11–14, 30–33, 74–77). This gives hope for the possibility of devising fast stopping tests.

Many authors like to consider problems in Karmarkar's canonical form, i.e., to have $b = e_m$ and $A^{\mathrm{T}} e_m = e$ in (1b). Explicit conversion to this form should have little effect on the pictures considered in this paper, as it generally adds as many new variables as new equality constraints (e.g. two of each in [8]); the main effect would come from perturbing the orthogonalization of F in (16).

We could also make pictures depicting Anstreicher's extension [2] of Karmarkar's algorithm to fractional linear programming problems. Anstreicher's step computation is similar to the primal projective step computation described above and should yield similar pictures.

For starting, some authors also like to arrange that e/n be feasible, usually by adding a new variable (at high cost), with no corresponding additional constraint, as in [24]. This is cleaner than doing a separate phase-1 calculation to find a feasible starting point $x°$, as it is not troubled by ''null variables'', i.e., components of x that vanish in all feasible solutions. But it adds a degree of freedom, which makes drawing pictures of the transformed feasible regions much harder.

One can regard Karmarkar's linear programming algorithm as an interior penalty-function method with a special choice of the penalty parameter. One may then be tempted to make other choices of the penalty parameter; see [11] and [12] for details. It is unclear how useful the approach of this paper might be in presenting an algorithm's eye view of such penalty methods. For problems like those considered above having three degrees of freedom, we might make a linear change of variables to turn the Hessian of the penalty function into the identity matrix I; pictures of the resulting transformed feasible region would then show the constraints as the algorithm sees them. Of course, for problems having two degrees of freedom, conventional level contour plots would also be informative.

Acknowledgements

I thank Bob Fourer, Eric Grosse, Norm Schryer, and Mike Todd for helpful comments on the manuscript.

References

[1] I. Adler, N. K. Karmarkar, M. G. C. Resende, and G. Veiga, ''An Implementation of Karmarkar's Algorithm for Linear Programming,'' draft manuscript, Dept. of Indust. Engin. & Oper. Res., Univ. of California, Berkeley, CA 94720 (May 1986).

[2] K. M. Anstreicher, ''A Monotonic Projective Algorithm for Fractional Linear Programming for Linear Programming,'' *Algorithmica* **1**(4), pp. 483–498 (1986).

[3] E. R. Barnes, ''A Variation on Karmarkar's Algorithm for Solving Linear Programming Problems,'' *Mathematical Programming* **36**, pp. 174–182 (1986).

[4] D. A. Bayer and J. C. Lagarias, ''The Nonlinear Geometry of Linear Programming I. Affine and Projective Scaling Trajectories,'' manuscript, AT&T Bell Labs, Murray Hill, NJ 07974 (Sept. 1986).

[5] T. M. Cavalier and A. L. Soyster, ''Some Computational Experience and a Modification of the Karmarkar Algorithm,'' manuscript (Feb. 1985).

[6] V. Chandru and B. S. Kochar, ''Exploiting Special Structures Using a Variant of Karmarkar's Algorithm,'' Research Memorandum No. 86–10, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 (June 1986).

[7] V. Chandru and B. S. Kochar, ''A Class of Algorithms for Linear Programming,'' Research Memorandum No. 85–14, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 (Nov. 1985; revised June 1986).

[8] J. E. Dennis, A. M. Morshedi, and K. Turner, ''A Variable-Metric Variant of the Karmarkar Algorithm for Linear Programming,'' *Math. Programming* **39**, pp. 1–20 (1987).

[9] D. M. Gay, N. K. Karmarkar, and K. G. Ramakrishnan, ''The Karmarkar Algorithm: Adding Wings to Linear Programming,'' *The AT&T Bell Laboratories Record*, pp. 4–10 (March 1986).

[10] D. M. Gay, ''A Variant of Karmarkar's Linear Programming Algorithm for Problems in Standard Form,'' *Mathematical Programming* **37**(1), pp. 81–90 (1987).

[11] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright, ''On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method,'' *Mathematical Programming* **36**, pp. 183–209 (1986).

[12] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, ''A Note on Nonlinear Approaches to Linear Programming,'' Technical Report SOL 86–7, Systems Optimization Lab., Dept. of Oper. Res., Stanford Univ., Stanford, CA 94305 (April 1986).

[13] D. Goldfarb and S. Mehrotra, ''A Relaxed Version of Karmarkar's Method,'' manuscript, Dept. of Indust. Engin. & Oper. Res., Columbia Univ., New York, NY 10027 (Dec. 1985, revised March 1986 and Dec. 1986).

[14] D. Goldfarb and S. Mehrotra, ‘‘Relaxed Variants of Karmarkar’s Algorithm for Linear Programs with Unknown Optimal Objective Value,’’ *Math. Programming* **40**(2), pp. 183–195 (1988).

[15] N. Karmarkar, ‘‘A New Polynomial-time Algorithm for Linear Programming,’’ *Combinatorica* **4**, pp. 373–395 (1984).

[16] I. J. Lustig, ‘‘A Practical Approach to Karmarkar’s Algorithm,’’ Technical Report SOL 85–5, Systems Optimization Lab., Dept. of Oper. Res., Stanford Univ., Stanford, CA 94305 (June 1985).

[17] R. E. Marsten, ‘‘The Design of the XMP Linear Programming Library,’’ *ACM Trans. Math. Software* **7**, pp. 481–497 (1981).

[18] N. Megiddo and M. Shub, ‘‘Boundary Behavior of Interior Point Algorithms in Linear Programming,’’ Report RJ 5319 (54679), IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 (Sept. 1986).

[19] S. Mehrotra, ‘‘A Self Correcting Version of Karmarkar’s Algorithm,’’ manuscript, Dept. of Indust. Engin. & Oper. Res., Columbia Univ., New York, NY 10027 (March 1986).

[20] A. Salamanca, ‘‘An Inner Ellipsoid Algorithm for Linear Programming,’’ Technical Report 1–ABR/86, Dept. of Math., E.T.S. de Ingenieros Industriales, Universidad Politécnica de Madrid, Madrid, Spain (1986).

[21] D. F. Shanno and R. E. Marsten, ‘‘On Implementing Karmarkar’s Method,’’ Working Paper 85–01 [sic], Graduate School of Administration, Univ. of California at Davis, Davis, CA 95616 (Sept. 1985).

[22] M. J. Todd and B. P. Burrell, ‘‘An Extension of Karmarkar’s Algorithm for Linear Programming Using Dual Variables,’’ *Algorithmica* **1**(4), pp. 409–424 (1986).

[23] M. J. Todd, ‘‘Exploiting Special Structure in Karmarkar’s Linear Programming Algorithm,’’ Technical Report 720, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853 (Oct. 1986).

[24] J. A. Tomlin, ‘‘An Experimental Approach to Karmarkar’s Projective Method for Linear Programming,’’ manuscript, Ketron, Inc., Mountain View, CA 94040 (Jan. 1985).

[25] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman, ‘‘On a Modification of Karmarkar’s Linear Programming Algorithm,’’ *Algorithmica* **1**(4), pp. 395–407 (1986).