J Heuristics (2007) 13: 471–503
DOI 10.1007/s10732-007-9021-7

Pivot, Cut, and Dive: a heuristic for 0-1 mixed integer programming

Jonathan Eckstein · Mikhail Nediak

Received: 8 July 2004 / Revised: 14 July 2006 / Accepted: 21 August 2006 / Published online: 7 April 2007

Abstract This paper describes a heuristic for 0-1 mixed-integer linear programming problems, focusing on "stand-alone" implementation. Our approach is built around concave "merit functions" measuring solution integrality, and consists of four layers: gradient-based pivoting, probing pivoting, convexity/intersection cutting, and diving on blocks of variables. The concavity of the merit function plays an important role in the first and third layers, as well as in connecting the four layers. We present both the mathematical and software details of a test implementation, along with computational results for several variants.

Keywords Integer programming · Simplex pivot · Convexity cut

This paper presents a heuristic method for finding feasible solutions to 0-1 mixed-integer linear programming (MIP) problems of the form

$$\min \mathbf{c}^\top \mathbf{x} \tag{1}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \tag{3}$$

$$x_i \in \{0, 1\}, \quad \forall i \in I, \tag{4}$$

J. Eckstein
Business School and RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854, USA
e-mail: jeckstei@rci.rutgers.edu

M. Nediak (✉)
Queen's School of Business, Queen's University, Goodes Hall, 143 Main St., Kingston, K7L 3N6, Ontario, Canada
e-mail: mnediak@business.queensu.ca

where $\mathbf{A}$ is an $m \times n$ matrix, $I \subseteq \{1, \ldots, n\}$ is the set of integer variables, and l_i, $u_i \in \{0, 1\}$ for all $i \in I$. For future reference, let P denote this problem's linear programming (LP) relaxation feasible region, that is, the set of all $\mathbf{x} \in \mathbb{R}^n$ satisfying (2) and (3). Further, let Y denote the set of all points satisfying (3) and (4); the set of feasible solutions is thus $P \cap Y$.

The majority of the extensive literature for this very general and widely-applied class of $\mathcal{NP}$-hard problems concerns exact solution, with the greatest practical success being achieved via branch-and-bound methods, augmented with cutting plane techniques. In this paper, we instead focus on heuristic methods. Heuristic methods may be of interest for the usual reasons—for example, it may not be possible to obtain exact solutions to some instances in a realistic or justifiable amount of time. In addition, heuristics may also be usefully combined with branch-and-bound methods by providing good incumbent solutions early in the development of the search tree. Such early incumbents can help limit the amount of memory needed to store the tree, or accelerate branch-and-bound search via techniques such as reduced-cost fixing; see Beale (1979). In parallel implementations of branch and bound—see for example Anbil et al. (1999), Eckstein (1994), Eckstein et al. (2001), Linderoth (1998), Ralphs and Ladányi (2000)—early, high-quality incumbent solutions can be particularly important in preventing exploration of noncritical portions of the tree. Indeed, it was parallel implementation of branch and bound that initially motivated our interest in MIP heuristics. Conversely, branch-and-bound methods can assist heuristics by intensifying the search in promising regions, and by providing guarantees of solution quality.

One can thus categorize applications of MIP heuristics as either "stand-alone", where the heuristic is run by itself, or in conjunction with branch and bound. This paper focuses on the stand-alone case. Upcoming work will discuss a streamlined adaptation of some of the ideas presented here that is more suitable for combination with branch-and-bound search.

By comparison to the literature of exact methods, the literature of heuristic methods for problems in the general form (1–4) is fairly limited. The earliest publications we have discovered present rounding heuristics for MIPs (in inequality form) whose LP relaxations' feasible regions have nonempty interior; see Faaland and Hillier (1979), Hillier (1969), Ibaraki et al. (1974). Starting from an LP relaxation solution, these methods construct a piecewise-linear path leading into this interior and then move parametrically along the path, attempting to find nearby integer-feasible solutions. We initially attempted to generalize these ideas to problems lacking an interior, but found the results disappointing.

An extensively tested heuristic for 0-1 MIP is the *pivot-and-complement* method of Balas and Martin (1980), which is based on an optimal solution being attained at a vertex where all binary variables are nonbasic. It uses individual pivots to try to remove integer variables from the basis.

The tabu-search-based method for 0-1 mixed integer programming of Løkketangen and Glover (1998) uses local search over the pivot neighborhood of the current vertex. This work includes extensive computational results, but limited to relatively small multi-constraint knapsack problems. This method uses a *merit function* to measure solution integrality, and treats the rounding of an LP solution as a bicriterion optimization involving both this merit function and the original objective function.

The more recent "OCTANE" heuristic of Balas et al. (2001) applies only to *pure* 0-1 integer programming (the case $I = \{1, \ldots, n\}$). It is based on enumerative intersection cuts; see Balas (1971), Burdet (1972), Glover (1972). The numerical results reported in this work are very encouraging.

A very general MIP heuristic approach is based on the notion of cut search proposed by Glover (1972); see Glover and Laguna (1997a, 1997b) and Chap. 6 of Glover and Laguna (1997c). However, numerical results have not been reported in this work.

In the past few years, research on MIP heuristics has intensified, in particular heuristics which take an already integer-feasible solution, and attempt to find better nearby solutions. Such methods include *local branching* by Fischetti and Lodi (2003), as well as *guided dives* and *relaxation induced neighborhood search* (RINS); see Danna et al. (2005). In principle, these methods could be combined with ours: our method could find an initial feasible solution, and one of these alternative methods could then improve upon it.

Recent work by Fischetti et al. (2005), appearing while this paper was under review, proposes a *feasibility pump* heuristic for MIP problems. At iteration $n + 1$, this method solves a linear program in which the original objective function is replaced by the L_1-distance between $\mathbf{x}_I$ and $[\mathbf{x}^n_I]$, where the subscript I denotes a vector of just the integer variables, $\mathbf{x}^n$ denotes the result of the prior iteration, and $[\cdot]$ denotes rounding to the nearest integer vector. The heuristic repeatedly solves such linear programs until $\mathbf{x}^n$ is integer feasible. One may regard this method either as an extreme form of diving—see Sect. 1.5—or as a nonsmooth Frank-Wolfe merit function procedure; see Remark 1.7 below. Fischetti et al. (2005) report a favorable comparison of their method with commercial software on a number of hard 0-1 instances.

Like the method of Løkketangen and Glover (1998), our approach to MIP heuristics is built around a merit function that is zero at integer-feasible points and positive elsewhere in the unit cube. The fundamental layer of our method uses either individual pivots or Frank-Wolfe blocks of pivots to reduce the value of the merit function, while trying to avoid excessive deterioration of the original objective. Thus, the most closely-related prior methods are those of Løkketangen and Glover (1998) and to a lesser extent of Balas and Martin (1980), which is also pivot-based.

Unlike Løkketangen and Glover (1998), we require that the merit function be concave. This restriction allows for quick selection of pivots based on local gradient information, leading relatively rapidly to local minima of the merit function over P. The basic search moves of Løkketangen and Glover (1998) are more akin to the second layer of our method, which we call "probing", in which we explicitly test a list of possible pivots. We invoke this probing layer only at local merit function minima that are not integer-feasible.

Merit function concavity is also critical in the third layer of our method. Thanks to concavity, failure of the probing layer of our method immediately gives rise to a *convexity cut*, also called an intersection cut—see Balas (1971, 1972), Balas et al. (1971), Glover (1973), Raghavachari (1969), Tuy (1964), Zwart (1973), Eckstein and Nediak (2005)—violated by the current vertex and (barring degeneracy) all adjacent vertices.

If probing fails, and the resulting convexity cut appears excessively shallow, we resort to the fourth and final layer of our heuristic: a recursive, depth-first "diving"

technique. In this method, we attempt to fix a (usually large) group of variables simultaneously. Typically, this operation results in a linearly infeasible problem, in which case we backtrack and instead apply a complementary "vertex" cut. This modification also makes our heuristic a provably exact, finitely terminating algorithm when it is allowed to run beyond the initial feasible solution.

Section 1 below develops the main technical ideas of our approach. Section 2 then discusses a range of less mathematically involved implementation issues ranging from starting point and merit function selection to software design. Finally, Sect. 3 presents computational results and analysis. Coupling of the method with branch and bound and treatment of problems with general integer variables are reserved for subsequent work.

1 Mathematical structure of the heuristic

The starting point of our approach is measuring integer infeasibility by a merit function constructed as follows: consider a collection of continuously differentiable concave functions $\phi_i : \mathbb{R} \to \mathbb{R}$, $i \in I$, such that $\phi_i(0) = \phi_i(1) = 0$ and $\phi_i(x) > 0$ for all $x \in (0, 1)$, and define

$$\psi(\mathbf{x}) = \sum_{i \in I} \phi_i(x_i). \tag{5}$$

Example 1.1 Given a parameter $\tilde{x} \in (0, 1)$, the following function satisfies the requirements on $\phi_i(\cdot)$:

$$\phi_{\tilde{x}}(x) = 1 - \begin{cases} \left(\dfrac{x - \tilde{x}}{\tilde{x}}\right)^2, & x \le \tilde{x}, \\ \left(\dfrac{x - \tilde{x}}{1 - \tilde{x}}\right)^2, & x \ge \tilde{x}. \end{cases} \tag{6}$$

Note that this function attains a maximum value of 1 at $x = \tilde{x}$, and that for $\tilde{x} = 1/2$, one obtains $\phi_{1/2}(x) = 4x(1 - x)$. We can select possibly different values $\tilde{x}_i \in (0, 1)$ for each $i \in I$ and obtain

$$\psi(\mathbf{x}) = \sum_{i \in I} \phi_{\tilde{x}_i}(x_i), \tag{7}$$

the class of functions used in the numerical experiments of Sect. 3.

1.1 Rounding via concave pivots and probing

The goal of the rounding portion of our heuristic is, given a fractional solution to (1–3), modify it in such a way that it becomes integer-feasible, while keeping the original objective function value as low as possible.

Rounding proceeds via primal simplex pivots starting with the basis corresponding to some optimal solution $\mathbf{x}^0$ of the relaxation (1–3). Consider a basic feasible solution $\bar{\mathbf{x}}$ to (1–3), denote the basic and non-basic matrices by B and N, and denote the

basic and non-basic index sets by J_B and J_N, respectively. The edges of P extending from $\bar{\mathbf{x}}$ will be denoted by $\mathbf{x}^{(i)}(t) = \bar{\mathbf{x}} + t\boldsymbol{\xi}^{(i)}$, where

$$\boldsymbol{\xi}^{(i)} = s_i \begin{pmatrix} -B^{-1}N\mathbf{e}^{(i)} \\ \mathbf{e}^{(i)} \end{pmatrix}$$

are the edge directions and

$$s_i = \begin{cases} +1, & x_i = l_i, \\ -1, & x_i = u_i \end{cases}$$

for all $i \in J_N$. Let K denote the affine cone spanned by $\{\boldsymbol{\xi}^{(i)} : i \in J_N\}$ with its vertex at $\bar{\mathbf{x}}$. Let the reduced costs for any cost vector $\mathbf{v}$ be denoted by $\mathbf{z}(\mathbf{v}) = (\operatorname{diag}\mathbf{s})(\mathbf{v}_N - (B^{-1}N)^\top \mathbf{v}_B)$. Given $\mathbf{v} \in \mathbb{R}^n$, define the following index sets:

$$I_+(\mathbf{v}) = \{i \in I : z_i(\mathbf{v}) > 0\},$$
$$I_-(\mathbf{v}) = \{i \in I : z_i(\mathbf{v}) < 0\},$$
$$I_0(\mathbf{v}) = \{i \in I : z_i(\mathbf{v}) = 0\}.$$

Similarly to the tabu-search method variants in Løkketangen and Glover (1998), we consider two kinds of rounding procedure we call the "ratio" and "sum" methods; see Sects. 1.2 and 1.3, respectively.

Both the "sum" and "ratio" methods rely heavily on the concavity of the merit function. Notice that for any concave differentiable function $\psi(\cdot)$ on $\mathbb{R}^n$ and edge direction $\boldsymbol{\xi}^{(i)}$,

$$z_i(\nabla\psi(\bar{\mathbf{x}})) = (\nabla\psi(\bar{\mathbf{x}}))^\top \boldsymbol{\xi}^{(i)}. \tag{8}$$

From the concavity of $\psi(\cdot)$ and its differentiability at $\bar{\mathbf{x}}$, one immediately obtains:

Lemma 1.2 *For any concave continuously differentiable function $\psi(\cdot)$ on $\mathbb{R}^n$, one has*

$$\psi(\mathbf{x}^{(i)}(t)) \le \psi(\bar{\mathbf{x}}) + t z_i(\nabla\psi(\bar{\mathbf{x}})).$$

Note that for *linear* ψ, the above relation holds with equality. Thus, one immediately obtains

Corollary 1.3 *For a concave continuously differentiable function $\psi(\cdot)$ on $\mathbb{R}^n$ and a fixed vector $\mathbf{c} \in \mathbb{R}^n$ with $\mathbf{c}^\top \boldsymbol{\xi}^{(i)} \neq 0$, one has*

$$\frac{\psi(\mathbf{x}^{(i)}(t)) - \psi(\bar{\mathbf{x}})}{\mathbf{c}^\top \mathbf{x}^{(i)}(t) - \mathbf{c}^\top \bar{\mathbf{x}}} \le \frac{z_i(\nabla\psi(\bar{\mathbf{x}}))}{z_i(\mathbf{c})}.$$

1.2 The "ratio" method

In the "ratio" method, we select pivots as follows: first, we look for pivots that decrease the merit function but do not increase the objective (Rule 1 below). If no such

pivot is possible, we try to pivot on a variable x_j that locally improves the merit function, that is, has $z_j(\nabla\psi(\bar{\mathbf{x}})) < 0$, at the least possible cost in terms of the objective function (Rule 2 below). Lemma 1.2 implies that the value of $\psi(\cdot)$ after pivoting will not be higher than predicted by the corresponding reduced cost $z_j(\nabla\psi(\bar{\mathbf{x}}))$. Corollary 1.3 also asserts that the ratio of reduced costs $z_j(\nabla\psi(\bar{\mathbf{x}}))/z_j(\mathbf{c})$ provides a lower bound on the relative change in the merit function to the change in the objective.

If no pivot locally improving $\psi(\cdot)$ exists, we are at a stationary point of $\psi(\cdot)$ on P. We then perform a "probing" operation in which we explicitly test possible pivots until a satisfactory improving pivot is found or the list of possible entering variables is exhausted (Rule 3 below). Although it is possible that probing will examine all the neighboring vertices of the current iterate $\bar{\mathbf{x}}$, we try to avoid this possibility by inspecting them in an order that gives priority to candidate entering variables x_i that have low values of $z_i(\nabla\psi(\bar{\mathbf{x}}))$ and $z_i(\mathbf{c})$; see (11) below. In inspecting a possible entering variable x_i, we compute the new iterate $\mathbf{x}$ that would result. If $\psi(\mathbf{x}) \geq \psi(\bar{\mathbf{x}})$, that is, $\mathbf{x}$ is farther from integrality than $\bar{\mathbf{x}}$, we abandon the pivot and move to the next pivot on the list; note that, due to the concavity of $\psi(\cdot)$, $\psi(\mathbf{x}) < \psi(\bar{\mathbf{x}})$ is possible even though $z_i(\nabla\psi(\bar{\mathbf{x}})) \geq 0$. If the pivot passes this test, we then consider its "objective sacrifice rate"

$$r(\bar{\mathbf{x}}, \mathbf{x}) = \frac{\mathbf{c}^\top\mathbf{x} - \mathbf{c}^\top\bar{\mathbf{x}}}{\psi(\bar{\mathbf{x}}) - \psi(\mathbf{x})}, \tag{9}$$

that is, the amount of objective value given up per unit improvement in the merit function. If this sacrifice rate is "acceptable" in relation to the prior history of the run—see (12) below—then $\mathbf{x}$ is accepted as the next iterate and no further pivots are probed. Otherwise, probing continues. If it is necessary to examine every possible pivot, we select, from among all the pivots with $\psi(\mathbf{x}) < \psi(\bar{\mathbf{x}})$, a pivot minimizing $r(\bar{\mathbf{x}}, \mathbf{x})$. If there was no pivot with $\psi(\mathbf{x}) < \psi(\bar{\mathbf{x}})$, we declare the rounding process to have failed.

Formally, the rounding algorithm proceeds by applying the following pivoting rules until an integer-feasible point is found or the probing step fails:

Rule 1 If $J = (I_-(\nabla\psi(\bar{\mathbf{x}}))) \cap (I_-(\mathbf{c}) \cup I_0(\mathbf{c})) \neq \emptyset$, then pivot on a variable

$$i \in \operatorname{Argmin}\{z_j(\nabla\psi(\bar{\mathbf{x}})) : j \in J\}. \tag{10}$$

Rule 2 Otherwise, if $I_-(\nabla\psi(\bar{\mathbf{x}})) \neq \emptyset$, pivot on a variable

$$i \in \operatorname{Argmin}\left\{\frac{z_j(\nabla\psi(\bar{\mathbf{x}}))}{z_j(\mathbf{c})} : j \in I_-(\nabla\psi(\bar{\mathbf{x}}))\right\};$$

Rule 3 (Probing) Otherwise, $I_-(\nabla\psi(\bar{\mathbf{x}})) = \emptyset$, and the current iterate $\bar{\mathbf{x}}$ is a stationary point of $\psi(\cdot)$ with respect to P. We therefore start a probing operation. First, we order the indices i of the possible entering variables by increasing

$$(z_i(\nabla\psi(\bar{\mathbf{x}})) + \theta)(z_i(\mathbf{c}) - \min_j z_j(\mathbf{c}) + \theta) \tag{11}$$

for some constant $\theta > 0$. In this order, we then evaluate each i as follows:

1. Compute the new iterate $\mathbf{x}$ that would result from pivoting x_i into the basis.
2. If $\psi(\mathbf{x}) \geq \psi(\bar{\mathbf{x}})$, move to the next pivot on the list.
3. Otherwise, compute $r(\bar{\mathbf{x}}, \mathbf{x})$ as in (9). If

$$r(\bar{\mathbf{x}}, \mathbf{x}) \leq \mu \frac{c^\top \bar{\mathbf{x}} - c^\top \mathbf{x}^0}{\psi(\mathbf{x}^0) - \psi(\bar{\mathbf{x}})}, \tag{12}$$

that is, it is no worse than μ times the rate of objective sacrifice sustained so far, then accept the new iterate $\mathbf{x}$ and terminate the probing step. If the denominator of the right-hand side of (12) is zero, we take the ratio to be zero.

If the entire set of possible entering variables has been examined without one passing the test (12), we accept the one with the lowest $r(\bar{\mathbf{x}}, \mathbf{x})$; if no pivot with $\psi(\mathbf{x}) < \psi(\bar{\mathbf{x}})$ was encountered, we declare the probing step to have failed.

Remark 1.4 One can use other pivot selection rules instead of Danzig's rule in (10) (for example, steepest-edge selection).

Remark 1.5 Note that probing may result in a large number of pivots being attempted and successively rejected. To avoid this phenomenon, we may instead modify the function $\psi(\cdot)$ in such a way that $\mathbf{x}$ is no longer a local minimum. In this case, however, special care must be taken to avoid cycling.

1.3 The "sum" method

An alternative approach is to use a single function that combines integrality and objective improvement at some fixed ratio. For a (sufficiently small) positive constant w, consider a modified merit function

$$\widehat{\psi}(\mathbf{x}) = \psi(\mathbf{x}) + w\mathbf{c}^\top \mathbf{x}. \tag{13}$$

Then one can replace Rules 1 and 2 of the ratio method with the following:

Rule 1′ Select a pivot

$$i \in \operatorname{Argmin}\{z_j(\nabla \widehat{\psi}(\bar{\mathbf{x}})) : j \in I_-(\nabla \widehat{\psi}(\bar{\mathbf{x}}))\}$$

whenever $I_-(\nabla \widehat{\psi}(\bar{\mathbf{x}})) \neq \emptyset$.

Otherwise the sum method applies a probing step, analogous to the Rule 3:

Rule 2′ The natural ordering of probing pivots, in this case, is of increasing $z_j(\nabla \widehat{\psi}(\bar{\mathbf{x}}))$. Again, if the point $\mathbf{x}$ resulting from a pivot has $\widehat{\psi}(\mathbf{x}) \geq \widehat{\psi}(\bar{\mathbf{x}})$, we abandon the pivot and move to the next pivot on the list. If a pivot is improving, we test if the improvement is satisfactory, as compared to a multiple of a historical improvement in $\psi(\cdot)$, i.e. if

$$\widehat{\psi}(\mathbf{x}) - \widehat{\psi}(\bar{\mathbf{x}}) \leq \mu \frac{\widehat{\psi}(\bar{\mathbf{x}}) - \widehat{\psi}(\mathbf{x}^0)}{N},$$

where N is a number of simplex steps from the initial vertex $\mathbf{x}^0$ to $\bar{\mathbf{x}}$ and $\mu > 0$ is a constant. If a satisfactory pivot is found, we accept it and terminate probing. On the other hand, if the entire pivot list was examined, we pick a pivot with the most negative $\widehat{\psi}(\mathbf{x}) - \widehat{\psi}(\bar{\mathbf{x}})$. If no improving pivot was encountered we declare the probing step to have failed.

The entire rounding procedure terminates either when it encounters an integer-feasible solution, or the probing step fails.

Remark 1.6 An important feature of the Rule 1′ is that it does not require a recomputation of the gradient after each pivot. Recall that the starting point of the procedure is denoted by $\mathbf{x}^0$. One possibility is to keep the gradient fixed until reaching $\mathbf{x}^1$, optimal with respect to the linear objective $\nabla\widehat{\psi}(\mathbf{x}^0)$. We then replace the gradient information with $\nabla\widehat{\psi}(\mathbf{x}^1)$, and repeat application of Rule 1′ until reaching $\mathbf{x}^2$, optimal with respect to the linear objective $\nabla\widehat{\psi}(\mathbf{x}^1)$, and so forth. We proceed in this fashion until we arrive at a point that is stationary with respect to the recomputed gradient. This modification of Rule 1′ results in a version of the classical *Frank-Wolfe method* for nonlinear programming; see for example Sect. 2.2.2 of Bertsekas (1999). Another possibility is to limit the number of pivots between gradient recomputations. An important practical advantage of the Frank-Wolfe variant is that it saves time in highly degenerate problems where the gradient actually remains constant for most pivots. Other advantages are that one can avoid the significant interface overhead that is often incurred when manipulating a simplex solver package to perform individual pivots, and one can rely on an existing simplex solver's built-in degeneracy handling techniques.

Remark 1.7 One may view the feasibility pump heuristic of Fischetti et al. (2005) as a form of Frank-Wolfe procedure in which $w = 0$ and $\phi_i(x_i) = \min\{x_i, 1 - x_i\}$ for all $i \in I$. This ϕ_i function is concave, but nonsmooth at $x_i = 1/2$. The feasibility pump uses an objective coefficient of 1 when $x_i^k < 1/2$ and -1 when $x_i^k > 1/2$, corresponding exactly to the gradient of $\min\{x_i, 1 - x_i\}$. Thus, it is possible to view our Frank-Wolfe rounding procedure as a variant of the feasibility pump using a smoothed merit function.

1.4 Rounding within a cutting plane method

Consider once again the ratio method: if probing fails, the rounding procedure has arrived at a fractional vertex $\bar{\mathbf{x}}$ of P whose entire pivot neighborhood is at least as fractional, as measured by $\psi(\cdot)$. In this case, the computations performed by the failed probing step lead naturally to a *convexity cut* satisfied by all integer-feasible solutions to (2–4) but violated by $\bar{\mathbf{x}}$. Here, we will give an abbreviated, somewhat informal explanation of these cuts; see Eckstein and Nediak (2005) for an extended, rigorous treatment—concepts found in that paper are marked with a "[∗]".

We begin by expressing

$$\bar{\mathbf{x}} = \begin{pmatrix} B^{-1}(\mathbf{b} - N\mathbf{g}) \\ \mathbf{g} \end{pmatrix},$$

where $g_i = l_i$ or $g_i = u_i$ for nonbasic variable at their lower or upper bounds, respectively. Next, consider the set

$$C = \{\bar{\mathbf{x}} \in \mathbb{R}^n \mid \psi(\bar{\mathbf{x}}) \geq 0\},$$

whose interior is $\operatorname{int} C = \{\bar{\mathbf{x}} \in \mathbb{R}^n \mid \psi(\bar{\mathbf{x}}) > 0\}$. By the construction of ψ, C is a closed convex set, and $\operatorname{int} C$ contains $\bar{\mathbf{x}}$ but cannot contain any integer feasible points. C is thus an *admissible set* [∗] for $\bar{\mathbf{x}}$ and the integer-feasible points Y.

Consider once more the cone $K \supseteq P$ emanating from $\bar{\mathbf{x}}$, as defined in Sect. 1.1. All elements of K (and thus of P) may be expressed as

$$\mathbf{x} = \bar{\mathbf{x}} + \sum_{i \in J_N} t_i \boldsymbol{\xi}^{(i)} \tag{14}$$

for some scalars $t_i \geq 0$, $i \in J_B$. Let J denote the indices $i \in J_N$ for which the ray consisting of all $\mathbf{x}^{(i)}(t) = \bar{\mathbf{x}} + t\boldsymbol{\xi}^{(i)}$, $t \geq 0$, intersects $\operatorname{bd} C$, and let $\hat{t}_i > 0$, $i \in J$, be such that $\psi(\mathbf{x}^{(i)}(\hat{t}_i)) = 0$ and thus $\mathbf{x}^{(i)}(\hat{t}_i) \in \operatorname{bd} C$. A point $\mathbf{x}$ of the form (14) for which $\sum_{i \in J} t_i/\hat{t}_i < 1$ can be expressed as a strict convex combination of $\bar{\mathbf{x}} \in \operatorname{int} C$ and $\mathbf{x}^{(i)}(\hat{t}_i) \in C$, $i \in J$. Thus, any such $\mathbf{x}$ must be in $\operatorname{int} C$ and cannot be integer-feasible. Therefore, any integer-feasible $\mathbf{x}$ must satisfy the condition $\sum_{i \in J} t_i/\hat{t}_i \geq 1$. In order to append this inequality to the LP, it should be reformulated as an inequality $\boldsymbol{\alpha}^\top \mathbf{x} \geq \beta$ written in terms of the original variables $x_1, \ldots, x_n$, instead of the edge displacements t_i, $i \in J_N$. One straightforward way of doing so is to define $\boldsymbol{\alpha} \in \mathbb{R}^n$ via

$$\alpha_i = \frac{s_i}{\hat{t}_i}, \quad i \in J, \qquad \alpha_i = 0, \quad i \in J_N \backslash J, \qquad \alpha_i = 0, \quad i \in J_B, \tag{15}$$

and

$$\beta = \boldsymbol{\alpha}^\top \bar{\mathbf{x}} + 1 = \boldsymbol{\alpha}_N \bar{\mathbf{x}}_N + 1 = \sum_{i \in J} \frac{s_i g_i}{\hat{t}_i} + 1, \tag{16}$$

resulting in the valid inequality

$$\sum_{i \in J} \frac{s_i}{\hat{t}_i} x_i \geq \sum_{i \in J} \frac{s_i g_i}{\hat{t}_i} + 1. \tag{17}$$

which must be satisfied by all $\mathbf{x} \in K \cap Y \subseteq P \cap Y$, but is clearly violated by $\bar{\mathbf{x}}$.

Employing the theory of convexity cut strength [∗], we generate a cut in this manner, but substitute for ψ the function $\overline{\psi}$ defined as a sum over $i \in I$ of

$$\overline{\phi}_i(x_i) = \min\{x_i \phi_i'(0), (x_i - 1)\phi_i'(1)\}.$$

When ψ is chosen as in Example 1.1, we then obtain

$$\overline{\psi}(\mathbf{x}) = \sum_{i \in I} \min\{x_i \phi_{\tilde{x}_i}'(0), (x_i - 1)\phi_{\tilde{x}_i}'(1)\} = 2 \sum_{i \in I} \min\left\{\frac{x_i}{\tilde{x}_i}, \frac{1 - x_i}{1 - \tilde{x}_i}\right\}. \tag{18}$$

In any case, $\overline{\psi}$ is concave and piecewise-linear, and it can be seen that $\overline{\psi}(\mathbf{x}) \geq \psi(\mathbf{x})$ for all $\mathbf{x} \in R^n$, and $\overline{\psi}(\mathbf{x}) = \psi(\mathbf{x}) = 0$ for all $\mathbf{x} \in Y$. The corresponding level set $\overline{C} =$

$\{\bar{\mathbf{x}} \in \mathbb{R}^n \mid \overline{\psi}(\bar{\mathbf{x}}) \geq 0\}$ has similar properties to C and strictly contains it. Therefore it will generate a cut *stronger* [∗] than the one generated from C, that is, guaranteed to eliminate at least as many points of K, and in most cases a strictly larger set of points. To generate this cut, we compute the positive solutions $\hat{t}_i$ to the equations

$$\overline{\psi}(\mathbf{x}^{(i)}(t)) = 0, \quad i \in J_N, \tag{19}$$

letting $J \subseteq J_N$ denote the set of indices i for which such solutions exist. We then employ the construction (15–16) to produce a cut $\boldsymbol{\alpha}^\top \mathbf{x} \geq \beta$, which we append to the LP formulation.

Note that the preceding failed probing step computes all the $\boldsymbol{\xi}^{(i)}$ in the process of considering possible pivots on all $i \in J_N$. Thus, if the probing step fails, the complete set of $\{\boldsymbol{\xi}^{(i)} : i \in J_N\}$ is available without any further computation. Once the set J and the intersection displacements $\hat{t}_i$ are in hand, each cut coefficient can be found in constant time. Thus, the main computational cost of generating the cut results from processing (19). The following proposition shows that this operation can be performed quite efficiently:

Proposition 1.8 *When the $\overline{\psi}(\cdot)$ is of the form (18), each equation (19) can be solved (or determined to have no positive solution) in $\mathrm{O}(|I| + |I \cap J_B| \log |I \cap J_B|)$ time.*

Proof To solve (19), we first identify an interval $[t_i', t_i'']$ that must contain $\hat{t}_i$ and on which $\overline{\psi}(\mathbf{x}^{(i)}(t))$ is purely linear. Consider the set

$$S = \{0\} \cup \left\{ \frac{\bar{x}_j - \tilde{x}_j}{s_i \xi_j^{(i)}} : j \in I \cap (J_B \cup i),\ \frac{\bar{x}_j - \tilde{x}_j}{s_i \xi_j^{(i)}} > 0 \right\} \cup \{+\infty\},$$

which includes all positive breakpoints of $\overline{\psi}(\mathbf{x}^{(i)}(\cdot))$ as well a 0 and $+\infty$. The largest value of S for which $\overline{\psi}(\mathbf{x}^{(i)}(\cdot))$ is nonnegative will give us t_i', and its next element is t_i''. Constructing and sorting S requires $\mathrm{O}(|I \cap J_B| \log |I \cap J_B|)$ time. Once S is known and sorted, t_i' may be identified in $\mathrm{O}(\log |I \cap J_B|)$ evaluations of $\psi(\mathbf{x}^{(i)}(\cdot))$ via a binary search procedure.

To evaluate the complexity of each calculation of $\psi(\mathbf{x}^{(i)}(\cdot))$, note that

$$\begin{aligned} \overline{\psi}(\mathbf{x}^{(i)}(t)) &= \sum_{j \in I} \overline{\phi}_j(\bar{x}_j + t\xi_j^{(i)}) \\ &= \sum_{j \in I \cap J_B} \overline{\phi}_j(\bar{x}_j + t\xi_j^{(i)}) + \sum_{j \in I \cap J_N j \neq i} \overline{\phi}_j(\bar{x}_j) \\ &\quad + \begin{cases} \overline{\phi}_i(\bar{x}_i + ts_i), & i \in I, \\ 0, & i \notin I, \end{cases} \end{aligned} \tag{20}$$

since $\xi_j^{(i)} = 0$ for $j \notin J_B \cup \{i\}$. Caching the constant value of second term in (20), which requires $\mathrm{O}(|I|)$ time to compute, each calculation of $\overline{\psi}(\mathbf{x}^{(i)}(t))$, requires evaluation of at most $|I \cap J_B| + 1$ of the $\overline{\phi}_j$'s in $\mathrm{O}(1)$ time each, for a total of $\mathrm{O}(|I \cap J_B|)$ time.

The values of $\psi(\mathbf{x}^{(i)}(\cdot))$ and its derivative on $t \in [t_i', t_i'']$ give the parameters of a linear equation whose root in $[t_i', t_i'')$ is the solution of (19) for a given i. As above, the evaluation of $\psi(\mathbf{x}^{(i)}(\cdot))$, and also its derivative, can be accomplished in $O(|I \cap J_B|)$ time. Thus, the total complexity, including caching the value of the second term in (20), is $O(|I| + |I \cap J_B| \log |I \cap J_B|)$. □

Remark 1.9 One can lower the complexity of finding the $\hat{t}_i$'s by using a sparse representation of the $\boldsymbol{\xi}^{(i)}$. Let J_B^i be the set of nonzero components of $\boldsymbol{\xi}^{(i)}$. Then the complexity of finding $\hat{t}_i$ is $O(|I| + |I \cap J_B^i| \log |I \cap J_B^i|)$, which ordinarily does not exceed the complexity of finding $\boldsymbol{\xi}^{(i)}$ itself. Thus, it is not prohibitive to automatically find $\hat{t}_i$ for each $\boldsymbol{\xi}^{(i)}$ examined while probing. In this case, we immediately obtain a convexity cut (17) at the end of every failed probing step.

While the convexity cut (17) is not depth-optimized [∗] and may not always be very deep, the failure of the probing step guarantees that it will cut off the current point $\bar{\mathbf{x}}$ and its immediate pivot neighborhood.

Thus, we may repeat the following steps until an integer feasible solution is found or the problem becomes infeasible:

1. Perform rounding as described in Sect. 1.1. If an integer feasible solution results, stop; otherwise go to step 2.
2. Add a convexity cut using information from the last failed probing step. Regain primal feasibility (by backtracking pivots or reoptimizing the problem). If the problem is infeasible, terminate: no feasible solution exists. Otherwise, go to step 1.

Remark 1.10 When using the sum method instead of the ratio method, one may still generate convexity cuts at local minima $\bar{\mathbf{x}}$ of $\widehat{\psi}$. Such points are not necessarily local minima of ψ, but if they are fractional, one may still use ψ or $\overline{\psi}$ to generate a convexity cut violated by $\bar{\mathbf{x}}$; however, it may not be violated by $\bar{\mathbf{x}}$'s entire pivot neighborhood, as in the ratio case.

Remark 1.11 Instead of adding a cutting plane in step 2 of the above procedure, one can change the merit function so that the current point is no longer a local minimum. Such modifications can be applied, for example, every fixed number of failed probing steps. The number of merit function changes should be limited, however, to avoid cycling among fractional vertices.

1.5 Combining cutting planes with diving

Convexity cuts were originally introduced by Tuy (1964), in the context of concave minimization over linear constraints. However, Zwart (1973) showed Tuy's algorithm to be nonconvergent, necessitating modifications such as in Bali and Jacobsen (1978), Tuy (1991) and Zwart (1974). Thus, we suspect it is not possible to prove that the algorithm presented in Sect. 1.4 must terminate finitely.

This section adds a conceptually simple feature that guarantees that the algorithm is finitely convergent. This same feature may in some cases facilitate finding feasible solutions more quickly.

Consider two disjoint subsets Q_0 and Q_1 of I. Solving the problem (1–4) can be reduced to solving two subproblems:

$$\begin{aligned} &\min \mathbf{c}^\top \mathbf{x} \\ &\text{s.t. } \mathbf{x} \in P, \\ &\quad x_i = 0, \qquad \forall i \in Q_0, \\ &\quad x_i = 1, \qquad \forall i \in Q_1, \\ &\quad x_i \in \{0,1\}, \quad \forall i \in I \setminus (Q_0 \cup Q_1) \end{aligned} \tag{21}$$

and

$$\begin{aligned} &\min \mathbf{c}^\top \mathbf{x} \\ &\text{s.t. } \mathbf{x} \in P, \\ &\quad \textstyle\sum_{i \in Q_0} x_i + \sum_{i \in Q_1} (1 - x_i) \geq 1, \\ &\quad x_i \in \{0,1\}, \quad \forall i \in I. \end{aligned} \tag{22}$$

Note that the first subproblem may be substantially easier than both the original problem and the second subproblem when the cardinality of $Q_0 \cup Q_1$ is close to that of I. We will generate and examine these subproblems systematically in a depth-first recursive search algorithm, with the child (21) always being examined first, and the child (22) being examined only when backtracking out of (21) without having found a feasible solution.

Classical branch-and-bound methods for integer programming (see for example Beale 1979) have long used an initial depth-first search phase in an attempt to identify integer feasible solutions, and the technique is often referred to as *diving*. However, the branching rule historically tends to involve simple rounding up or down of a single integer variable, as opposed to a large block of variables as attempted here.

Let $\bar{\mathbf{x}}$ be a current vertex of P and $\boldsymbol{\alpha}^\top \mathbf{x} \geq \beta$ be a cut. We define the *depth* [∗] of the cut with respect to $\bar{\mathbf{x}}$ to be the distance between $\bar{\mathbf{x}}$ and the hyperplane $L(\boldsymbol{\alpha}, \beta) = \{\mathbf{x} : \boldsymbol{\alpha}^\top \mathbf{x} = \beta\}$. Measuring distance with the 2-norm, we obtain

$$d(\bar{\mathbf{x}}, L(\boldsymbol{\alpha}, \beta)) = \frac{\beta - \boldsymbol{\alpha}^\top \bar{\mathbf{x}}}{\|\boldsymbol{\alpha}\|}. \tag{23}$$

We will use a cut's depth as a measure of its quality, and refer to the cuts of the form

$$\sum_{i \in Q_0} x_i + \sum_{i \in Q_1} (1 - x_i) \geq 1 \tag{24}$$

used in (22) as *vertex* cuts. An advantage of vertex cuts over convexity cuts is that one can control their depth

$$d(\bar{\mathbf{x}}, Q_0, Q_1) = \frac{1 - \sum_{i \in Q_0} \bar{x}_i - \sum_{i \in Q_1} (1 - \bar{x}_i)}{(|Q_0| + |Q_1|)^{1/2}}$$

via the choice of Q_0 and Q_1. Our goal is to include as many indices in Q_0 and Q_1 as possible, while still making sure that the cut (24) is "sufficiently" deep. Starting with $Q_0 = Q_1 = \emptyset$, we process the elements $\bar{x}_i$ of $\bar{\mathbf{x}}$ in increasing order of $\min\{\bar{x}_i, 1 - \bar{x}_i\}$, as follows:

- If $\bar{x}_i < 1 - \bar{x}_i$, let $Q_0' = Q_0 \cup \{i\}$ and $Q_1' = Q_1$, otherwise let $Q_0' = Q_0$ and $Q_1' = Q_1 \cup \{i\}$.
- If $d(\bar{\mathbf{x}}, Q_0', Q_1') \geq d^*$ (for some fixed $d^* > 0$) set $Q_0 \leftarrow Q_0'$, $Q_1 \leftarrow Q_1'$ and proceed to the next index, otherwise stop.

A natural value of d^* that guarantees at least one i in $Q_0 \cup Q_1$ with $\bar{x}_i \notin \{0, 1\}$ is $1/(2\sqrt{|I|})$. To see this, observe that $d(\bar{\mathbf{x}}, Q_0, Q_1)$ decreases monotonically as we add elements to Q_0 and Q_1. Moreover, observe that

$$d(\bar{\mathbf{x}}, Q_0, Q_1) = \frac{1 - \sum_{i \in Q_0 \cup Q_1} \min\{\bar{x}_i, 1 - \bar{x}_i\}}{(|Q_0| + |Q_1|)^{1/2}},$$

by our construction procedure. Note that $\min\{\bar{x}_i, 1 - \bar{x}_i\} \leq 0.5$ for all $i \in I$. Thus, at the stage of the procedure when it has just added the first index i' with $\min\{\bar{x}_{i'}, 1 - \bar{x}_{i'}\} > 0$, we have

$$d(\bar{\mathbf{x}}, Q_0, Q_1) = \frac{1 - \min\{\bar{x}_{i'}, 1 - \bar{x}_{i'}\}}{(|Q_0| + |Q_1|)^{1/2}} \geq \frac{1}{2\sqrt{|I|}}$$

since $|Q_0| + |Q_1| < |I|$. Thus, we have proved the following proposition:

Proposition 1.12 *If $d^* = 1/(2\sqrt{|I|})$, then, on termination of the above procedure, $Q_0 \cup Q_1$ will contain at least one index i such that $\bar{x}_i \notin \{0, 1\}$.*

We combine rounding, convexity cuts, and diving in the recursive procedure below. We start by invoking the procedure on arguments consisting of (1–4) and an optimal extreme point solution of its LP relaxation.

1. Perform rounding as in Sect. 1.1. If an integer feasible solution is found, stop and return it, otherwise continue with step 2.
2. Compute the depth d_{conv} of the convexity cut constructed in the last failed probing step. Also, construct sets Q_0 and Q_1 by the procedure outlined above, with a fixed $d^* \leq 1/(2\sqrt{|I|})$.
3. If $d_{\text{conv}} \geq d(\bar{\mathbf{x}}, Q_0, Q_1)$, add the convexity cut and go to step 6, otherwise go to step 4.
4. Construct a subproblem of the form (21) and determine whether its LP relaxation is feasible. If not, skip to step 5. Otherwise, recursively apply this entire procedure to the subproblem. If the procedure returns a feasible solution, return this solution. Otherwise, continue with step 5.
5. Add a vertex cut of the form (24). Continue with step 6.
6. Regain primal feasibility by backtracking pivots or reoptimizing the problem. If the problem becomes infeasible, return an error code indicating no feasible solution to the (sub)problem exists. Otherwise go to step 1.

It is now straightforward to prove convergence to some feasible solution, using techniques similar to standard cutting plane method convergence proofs for concave minimization; see for example Horst et al. (1995). It is necessary, however, to impose the assumption, standard in such proofs, that P is bounded.

Proposition 1.13 *Suppose P is bounded. Then the algorithm above terminates in a finite number of iterations either with an integer-feasible solution, or on detection of integer infeasibility.*

Proof The proof is by induction on the number of integral variables $|I|$. When $I = \emptyset$, any vertex of P is vacuously integer-feasible.

Now suppose that we have proved finiteness for the numbers of integral variables less than $|I|$, but the algorithm does not terminate for some problem with $|I|$ integer variables. Whenever we generate a subproblem of the form (21), its number of integer variables is less than $|I|$, so the algorithm must terminate on the subproblem by the induction hypothesis. Thus, the only possibility is that the sequence of cuts added to the subproblem (22) is infinite. Consider the sequence of polytopes $\{P^k\}_{k=0}^{\infty}$ generated by the algorithm, where $P^0 = P$. Let $\bar{\mathbf{x}}^k$ be a vertex of P^k being cut off by each respective cut. Since the depth of all the cuts is at least d^*, the distance from $\bar{\mathbf{x}}^k$ to any point of P^{k+1} is at least d^*. Thus, $\|\bar{\mathbf{x}}^k - \bar{\mathbf{x}}^l\| \geq d^*$ for all $l > k$. Note, however, that the sequence $\{\bar{\mathbf{x}}^k\}_{k=0}^{\infty}$ belongs to the bounded, finite dimensional and, thus, compact polytope P. It follows, that we can extract a convergent subsequence $\{\bar{\mathbf{x}}^{k_i}\}_{i=0}^{\infty}$, for which we would have $\|\bar{\mathbf{x}}^{k_i} - \bar{\mathbf{x}}^{k_{i+1}}\| \to 0$ as $i \to \infty$, a contradiction. □

Remark 1.14 Without sacrificing convergence, one can modify the decision rule between convexity and vertex cuts to $\gamma d_{\text{conv}} \geq d(\bar{\mathbf{x}}, Q_0, Q_1)$, where a positive constant γ specifies the level of preference given to convexity cuts.

Remark 1.15 In practice, vertex cuts may also be used to improve the stability of the method when one encounters numerical difficulties constructing convexity cuts.

Remark 1.16 One can also continue the algorithm after finding an initial integer-feasible solution, in the hope of obtaining better solutions in subsequent iterations. When the only termination criterion is infeasibility of the branch (22), and P is bounded, the algorithm then becomes an exact cutting-plane algorithm for mixed 0-1 integer programming.

2 Implementation issues

2.1 Starting point

To start, the algorithm needs only a feasible vertex of the relaxation, which does not have to be optimal. In fact, if we use a two-phase simplex method to solve the LP relaxation, one could start the heuristic run immediately after the first phase which finds a feasible vertex. However, for simplicity of the implementation of the experiments in the following section, we only run the heuristic from the LP relaxation solution.

2.2 Merit function parameter selection

One of the main practical issues in the implementation of Pivot, Cut, and Dive is a selection of parameters $\tilde{\mathbf{x}}$ of the merit function (7). In this paper's experiments, we explore the following possibilities (*types* of the merit function), all special cases of Remark 1.1:

1. Pure quadratic merit function: $\tilde{x}_i = 1/2$, for all $i \in I$.
2. Merit function with a shifted maximum: let $\mathbf{x}^0$ be a starting point of the algorithm and set
$$\tilde{x}_i = \begin{cases} x_i^0 - \sigma c_i, & \text{if } x_i^0 - \sigma c_i \in (0,1), \\ 1/2, & \text{otherwise,} \end{cases}$$
where σ is a sufficiently small positive number, for all $i \in I$. The idea is to prevent the starting point $\mathbf{x}^0$ from being a local minimum for the merit function.
3. Random merit function: $\tilde{x}_i$ drawn form a uniform distribution on $(0, 1)$ for all $i \in I$.

Although the above list is by no means exhaustive, we will see in our experiments that it does appear to present a sufficiently diverse collection of merit functions. Other possibilities are selecting all parameters $\tilde{x}_i$, $i \in I$ to be equal, but not necessarily at $1/2$, or drawing them from a non-uniform distribution. We can also extend a family of the merit functions by scaling its individual components, that is, $\psi(\mathbf{x}) = \sum_{i \in I} \gamma_i \phi_{\tilde{x}_i}(x_i)$ for some positive scalars γ_i, $i \in I$.

As already pointed out in Remark 1.11, it is possible to reset the merit function instead of adding a cut after failed probing step. Such a change can be easily accomplished when using merit function types 2 and 3. In the case of type 2, we reset the parameter vector $\tilde{\mathbf{x}}$ using a current vertex $\bar{\mathbf{x}}$ instead of the LP optimum $\mathbf{x}^0$. In the case of type 3, we can reset the parameters by random sampling. Again, there is a range of unexplored possibilities based on scaling components of the merit function.

2.3 Feasibility recovery techniques after cut addition

Since adding a cut makes the current vertex infeasible, the algorithm cannot proceed until feasibility is recovered. A typical approach in cutting plane algorithms for linear programming is to use a dual simplex method starting from the current basis to find a new optimum with respect to the original objective. Note, however, that using the original linear objective $\mathbf{c}^\top \mathbf{x}$ is inadvisable in our case, since the algorithm may have made a significant progress toward integer feasibility. Reoptimization with respect to $\mathbf{c}^\top \mathbf{x}$ does not take integrality into account, and may return us to the previous or even worse level of the merit function. On the other hand, there is no known equivalent of the dual simplex method for concave minimization. Thus, we do not have an efficient way to reoptimize from an infeasible point with respect to the merit function.

In this situation, we consider the following two approaches. First, we can keep a log of all pivots executed by the algorithm. After adding a cut, these pivots are backtracked until a feasible vertex is encountered or the log is exhausted. In the latter case, we can reoptimize with respect to the original linear objective. The second approach is to use a surrogate linear objective for reoptimization. This linear objective should take both integrality and the original objective into account. One possibility is to use the current gradient of the merit function. Unfortunately, this straightforward option seemed to perform poorly in preliminary testing. Another possibility is to minimize with respect to the an objective vector $\mathbf{d}$ such that
$$d_i = w' c_i + \begin{cases} 0, & i \notin I, \\ -1, & x_i < \epsilon, i \in I, \\ 1, & x_i > 1 - \epsilon, i \in I, \end{cases}$$

where $\mathbf{x}$ is a current vertex and $\epsilon, w' > 0$ are sufficiently small constants. The rationale here is to regain feasibility while discouraging changes in integer or near-integer variables. The term $w'\mathbf{c}^\top\mathbf{x}$ is intended to control possible deterioration in the original linear objective.

2.4 Cut management

Note that the convexity cuts generated by the algorithm can be quite dense. Moreover, in some instances, the number of cuts may be quite large before the first feasible solution is found. Adding large numbers of dense cuts will undoubtedly make each iteration of our algorithm much more time-consuming. On the other hand, computational experience shows that most added cuts stay active only briefly. Thus, one can hope that just a small fraction of all the cuts generated will be needed in the formulation at any one time. Thus, the sort of *cut management* techniques employed by exact branch-and-cut algorithms could be helpful. The general idea of our cut management scheme is standard: to maintain a pool of cuts which may be included as needed in the formulation that the LP solver operates on. As a run of the algorithm proceeds, cuts can be moved in and out of the current formulation.

We first discuss when cuts have to be added to the formulation. There are three possible scenarios:

1. A cut has just been generated and is violated by the current vertex.
2. A cut has to be added to the formulation during feasibility recovery by pivot backtracking.
3. A periodic test of the cut pool discovers a violated cut.

Scenario 1 is straightforward. The cut is added both to the pool and to the formulation. The algorithm proceeds by pivot backtracking.

Scenario 2 can occur as a consequence of scenarios 1 and 3. Two cases have to be considered:

2a. A pivot that is being backtracked makes the cut's slack a basic variable. In this case, we have no choice but to add this cut to the formulation if it is currently in the pool. Otherwise backtracking cannot proceed.
2b. After all the cuts which are currently in the formulation are satisfied, we still have to check the remaining cuts in the pool. If any cut in the pool is violated, it should be added to the formulation and the algorithm continues pivot backtracking.

We use periodic tests of the cuts in the pool to make sure the current vertex satisfies them. If any cut is violated, it has to be added to the formulation (scenario 3) with a subsequent feasibility recovery by pivot backtracking. Backtracking may well result in additional cuts being added to the formulation by scenario 2b. The periodicity of the test should be determined experimentally, since the trade-off between computationally more expensive iterations and cut management overhead will depend on the simplex solver being used and the problem at hand.

We now discuss how cuts can be removed from the formulation. The only reasonable place for removal of a cut is in the periodic test of scenario 3. Here, we remove a cut if it has stayed inactive for some number of iterations. This number should probably be at least the periodicity of the test. There is also a question of whether cuts

should be removed before or after the cut violation test. The first option will speed up backtracking, while the second will guarantee that a cut just deleted does not have to be put back immediately. Note, however, that this situation is not very likely to happen, unless we had to backtrack a number of pivots which is more than the periodicity of the test.

Implementation of the cut management requires the following data structures:

- A cut pool as a sparse matrix. Cuts may not have nonzero coefficients for the slacks of other cuts. Note that a convexity cut given by the formula (17) may not satisfy this condition. In this case, we substitute out explicit expressions for the slacks and use a modified cut instead.
- An array to establish correspondence between the cuts in the formulation and the row numbers in the pool matrix.
- An array of 0/1 flags for each cut, indicating whether the cut is in the formulation.
- An array of counters of "inactive" iterations for all cuts currently in the formulation. The counters will have to be updated after backtracking as well as in regular iterations.
- A pivot log (already required for the feasibility recovery) should be in a format that allows the algorithm to identify the slack variable of a cut with a row of the pool matrix. This indexing is necessary to properly handle pivots involving slacks of cuts that were deleted from the formulation, or if cut positions shift in the matrix. The current implementation simply reserves a fixed column for each slack variable.

Note also that if certain cuts are frequently moved in and out of the formulation, their maximum "inactive tenure" should probably be extended. The current implementation uses a "cut rank" which is initialized to 1 for each new cut. Whenever a cut is moved back into the formulation, its rank is incremented by 1. The maximum inactive tenure of a cut is given by a product of its rank with a fixed maximum tenure parameter.

2.5 Termination criteria

We now consider the imposition of an iteration-based termination criterion to keep our heuristic from running too long. It is especially important to have such criteria in local-search methods, like ours, that can be restarted and directed along different search paths.

First, observe that a simple criterion based only on the number of iterations of the main loop would not work well in our case. Different iterations of the loop may involve (almost) regular simplex pivots, probing, adding cutting planes or even diving, which require quite different amounts of CPU time. Thus, a good criterion should use more information than just the master iteration counter. We have noticed through profiling of the algorithm that the majority of the execution time is spent inside the subroutines of the LP solver. Thus, a reasonable termination criterion should involve the counts of calls to such subroutines. Experimentally, we found that a reasonable termination criterion can be constructed from a linear form in terms of the number of iterations of the main loop N, number of backtracked pivots N_{BT}, number of internal solver pivots N_{S} (number of internal pivot subroutine invocation by the LP solver) and the number of probing pivots N_{P}. The coefficients of this linear form

should indicate relative computational complexity associated with these operations, to be estimated statistically.

Of course, this criterion is not perfect, in the sense that each of the operations may contribute differently to the execution time depending on the number of cuts added to the formulation. Thus, this criterion can only be used to roughly compare complexity of running the heuristic to that of solving the LP. However, a *relative* contribution of different types of pivots should not be very dependent on the current number of cuts.

Additional termination criteria include thresholds on the number of cuts that can be generated, and the running time. Finally, we simply terminate a run if numerical difficulties are encountered by LP solver, since it is easier to restart the run than enter recovery procedures.

2.6 Running beyond the first feasible solution

Remark 1.16 has already noted that the algorithm may be run beyond the first feasible solution. However, this modification may result in trying to apply convexity cuts at integer-feasible points. Since such points are on the boundary of the admissible set $\overline{C}$ used to generate the cut, it is impossible to guarantee that the current point is cut off. This problem will typically exhibit itself as an absence of a sufficiently positive numerical solution of (19). One possibility is to use a vertex cut whenever the convexity cut computation encounters this situation. Another possible technique is adding an *objective cut* restricting the objective to be some increment better the value of the best known integer-feasible solution.

2.7 Specifics of the sum method

There are several additional features that can be implemented in conjunction with the sum method. First, according to Remark 1.6, we can employ a Frank-Wolfe type technique for rounding, which saves us computation of the merit function and its gradient at many steps of the algorithm. More importantly, in practice it also saves on interface overhead and allows us to use the simplex solver's own, typically advanced, degeneracy handling technique. These advantages are so important in some cases that Frank-Wolfe modification of the sum method is the only strategy capable of finding a feasible solution in the allotted time (see the numerical experiment discussion below). Of course, when we use this modification, feasibility recovery after cut addition cannot be accomplished through pivot backtracking, the main reason being that LP solvers typically do not provide a pivot sequence to the calling program. Thus, we used a reoptimization with the surrogate objective $\mathbf{d}$ of Subsect. 2.3 instead.

Second, one may wish to consider various manipulations of the weight parameter w. For example, if a problem proves to be difficult for the heuristic, it can dynamically lower w. Similarly, when the heuristic is left to run beyond the first feasible solution, it can increase the w, to put more emphasis on the objective function. In parallel environments, it is also possible to start sum method runs with different values of w on different processors. Strategies for updating w can then take into account which of these runs were successful in finding feasible solutions and what was these solutions' quality.

2.8 Software implementation methodology

For optimal performance, pivot-oriented heuristics such as ours should ideally be implemented at a low level, as an essentially internal part of a high-quality simplex LP solver. In practice, such implementations can be very difficult unless one is intimately familiar with the internal workings of the solver. Further, if the simplex solver is a proprietary commercial product, such an implementation would be possible only for the firm owning the solver. Such a low-level implementation is also "locked" into a particular solver, severely restricting portability and reusability. Thus, it makes sense, at least until the heuristic's utility is firmly established, to use a standard interface which provides identical interaction with a range of simplex solvers.

One such interface is an Open Solver Interface (OSI) of the Common Interface for Operations Research (COIN) initiative. OSI is a C++ class library that provides a standard way to interact with many industrial and academic linear and integer programming packages. Among other things, it includes classes for representation of sparse vectors and matrices (`OsiPackedVector`, and `OsiPackedMatrix`), and a generic solver interface class `OsiSolverInterface`. Through these classes, OSI provides many capabilities required from a solver by the heuristic:

- Convenient sparse vector and matrix representation;
- Calling a simplex solver for linear programming;
- Querying solution information: variable values and reduced costs;
- Querying and setting basis status;
- Problem modification.

On the other hand, OSI did not originally provide the following functionality:

- Computing reduced costs for an arbitrary cost vector (without explicitly resetting the cost vector in the solver object);
- Computing the results of a primal simplex pivot, given an entering variable (a leaving variable, a step size and an edge direction);
- Applying a precomputed pivot to a given basis;
- Access to simplex tableau information, which may be used in convexity cut generation procedures.

Thus, we decided to create an abstract class `OsiSimplexInterface` specifying appropriate simplex-method-based extensions to `OsiSolverInterface`. We then use the mechanism of multiple inheritance to create extended versions of already-implemented solver interfaces of OSI. In particular, we created such an extension for ILOG's CPLEX (however, its overhead for individual pivot operations was excessive due to safeguards in CPLEX's proprietary API). The same multiple inheritance mechanism can be used to create totally new extended interfaces. We implemented such an extended interface for the open-source textbook solver SIMPO; see Vanderbei (2001). The solver's code was first converted to object-oriented form to allow better access to its components and ensure complete reproducibility of results. We have also augmented it to explicitly handle general bounds on the variables (the original version used problem reformulation) and to use a standard EXPAND degeneracy handling technique of Gill et al. (1989). Note that since our original computational work was performed, `OsiSimplexInterface` functionality has been incorporated into the standard COIN library.

The rest of heuristic was implemented in C++ using object-oriented design techniques. Note that since the heuristic communicates with simplex solvers through abstract classes such as `OsiSolverInterface` and `OsiSimplexInterface`, it is theoretically portable and solver-independent.

3 Numerical experiments

We now present the results of several numerical experiments with a stand-alone version of the Pivot, Cut, and Dive heuristic. In particular, we compare performance of the ratio method with the standard and Frank-Wolfe variations of the sum method. Performance is evaluated in terms of the quality of solutions and the running time, using SIMPO as the LP solver. We use the SIMPO solver because of its relatively small overhead of individual pivot manipulations, allowing a closer comparison between the Frank-Wolf and individual pivot variants. Similar low overhead could probably be obtained from interfaces to other solvers, but only after extensive software engineering work by developers familiar with the solvers' internal architecture.

3.1 Test set

The test problems were selected from the standard MIPLIB 3.0 collection of Bixby et al. (1998). To improve numerical stability, all the problems were scaled. We excluded instances containing general integer variables, leaving 49 MIPLIB problems. Of these, 9 problems were ruled out due to the limitations of the SIMPO solver: `air04`, `air05`, `danoint`, `dano3mip`, `fast0507`, `gamsmod`, `nw04`, `rentacar` and `seymour`; on each of these, SIMPO would either encounter numerical difficulties or fail to terminate after 20,000 iterations. In future, we may be able to tackle such problems by using an industrial-strength LP solver. We present our results for the remaining 40 problems below.

3.2 Comparison of the default methods

We now consider a detailed comparison of the default implementation of three main methods: ratio, sum, and sum using Frank-Wolfe. Following our original motivation from parallel computing, we set up for each heuristic variant a run on 16 simulated, "embarrassingly" parallel processors, with each processor using a different merit function. In reality, we ran the algorithm in a serial fashion starting from the optimal solution to LP relaxation, without using information obtained in the previous runs. Merit functions were selected as follows:

1. One "processor" with a pure quadratic merit function.
2. One "processor" with a "shifted maximum" merit function.
3. Fourteen "processors" with randomly selected merit functions.

In cases 2 and 3, the algorithm changed the merit function, instead of adding a cut, at every sixth failed probing step; see Remark 1.11. A limit on the number of merit function changes therefore follows from a limit we imposed on the number of cuts.

To estimate the coefficients of the operation counters in our termination criterion, we conducted the following experiment. With cuts and diving switched off, we computed the ratio of the running time to the time per simplex iteration while solving the LP relaxation. The hypothesis is that in this way we can make timing results comparable between different problems by discarding the effect of problem size. Let the running time of the heuristic be denoted by T, and the time to solve the LP and the number of simplex iterations as T_{LP} and N_{LP}, respectively. The running time, expressed in "simplex iterations", and denoted by

$$Z = \frac{T}{T_{\mathrm{LP}}/N_{\mathrm{LP}}},$$

was used as a dependent variable in a regression model. Without cuts and diving, only the basic rounding operations could contribute to execution time. The following independent variables were used: main iteration counter (earlier denoted as N), probing step counter (N_{PS}), and probing pivot counter (N_{P}). The regular pivot counter is dependent on these three variables, as any iteration of the main loop includes a single complete pivot when cuts and diving are not used. The model

$$Z = \beta_N N + \beta_{\mathrm{P}} N_{\mathrm{P}} + \beta_{\mathrm{PS}} N_{\mathrm{PS}} + \epsilon,$$

where ϵ is an error term, was able to explain about 80% of the variability in the execution time. Of the three variables, the probing step counter N_{PS} proved to be statistically insignificant. The remaining two variables were significant at a very high level and had coefficients of $\beta_N = 3.0$ for the main iteration counter and $\beta_{\mathrm{P}} = 0.35$ for the probing pivot counter. Of course, in general, we also have other types of pivots: an internal solver pivot and a backtracking pivot; recall that we use the term "internal solver pivot" to refer to an internal invocation of the pivot subroutine by the LP solver. Both of them involve a refactorization of the basis and zero or very insignificant interface overhead. Thus, their coefficients can be assumed to be approximately 1.0. Of course, in this experiment, we did not estimate a complexity of adding a cut or performing diving step. We note, however, that the main computational effort in both cases will involve feasibility recovery, which is accounted for by the counts of backtracking and internal solver pivots N_{BT} and N_{S}, respectively.

Based on this analysis, runs of the heuristic were terminated whenever any of the following conditions was met:

- A limit of 20,000 on the linear form $3N + 0.35N_{\mathrm{P}} + N_{\mathrm{BT}} + N_{\mathrm{S}}$ was reached.
- A probing step failed after a maximum of 500 cuts had already been added.
- A time limit of 60 seconds was exceeded.
- A feasible solution was found.
- Numerical difficulties were encountered due to pivot selection or feasibility tolerance selection problems; see Gill et al. (1989).

The heuristic was run on a 500 MHz Pentium III system under RedHat GNU/Linux 7.2. The program was compiled using the Linux distribution's default g++ 2.96 compiler with optimization switched on.

3.3 Performance evaluation: gap and run time

In this subsection, we compare the performance of the three methods using the *gap*, defined as a relative objective function difference between the feasible solution and the optimum (published in MIPLIB), and the run time. For each method, the experiment recorded the following:

- Gaps for both the first and the best feasible solution out of the 16 runs.
- Time to the first and best feasible solutions out of the 16 runs.
- Average execution time of the 16 runs.

Gaps of the first and the best feasible solution in the ratio, sum and Frank-Wolfe sum methods are presented in Figs. 1, 2, and 3, respectively. Only successfully solved problems are included, and their ordering is based on the best gap. Note that the gap of the first solution seems to be strongly correlated with the gap of the best solution, especially for the ratio method. This correlation implies that even the first solution may be useful when this heuristic is run in parallel environments.

To compare the three methods in terms of the first and the best solutions, we consider empirical distribution functions of the gap; see Figs. 4 and 5, respectively. Their plots show what fraction of the test problems (vertical axis) have a gap not exceeding a given value (horizontal axis). The higher the plot, the better the performance of the method. We note that the regular sum method is inferior both to the ratio and the Frank-Wolfe sum method. The ratio method is superior on easier problems, but the Frank-Wolfe sum method becomes competitive on harder ones. This is especially apparent for the gap of the best solution (Fig. 5).

We summarize the gap information in Table 1. Note that the Frank-Wolfe sum method was most successful in finding at least some feasible solution (34 problems

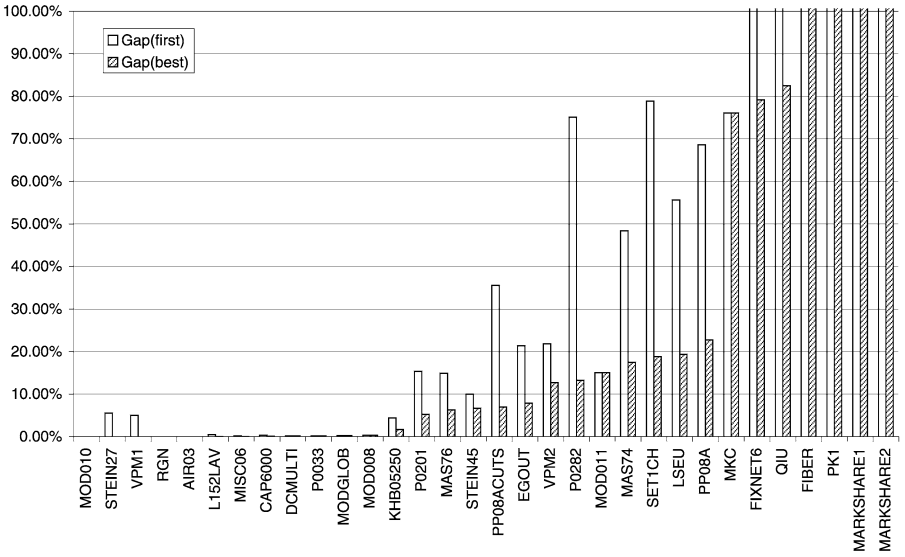

Fig. 1 Gap for the ratio method

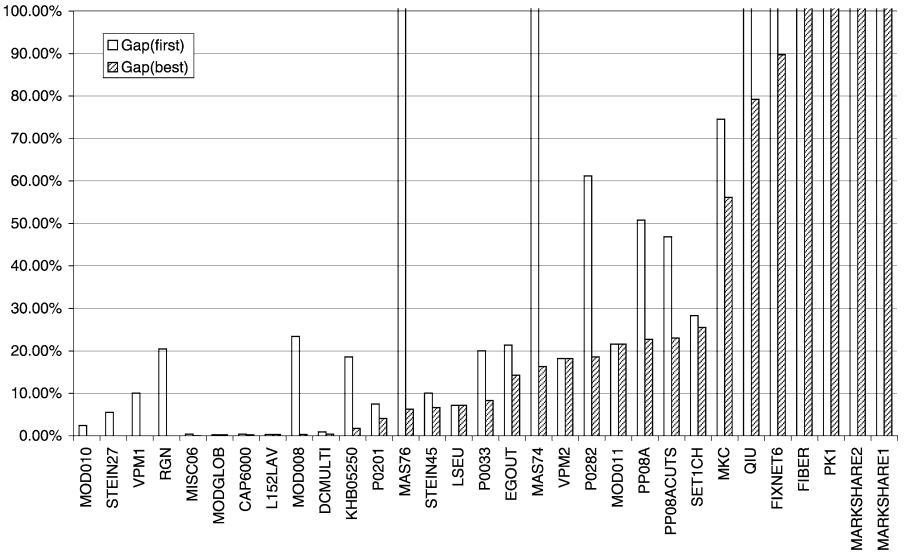

Fig. 2 Gap for the sum method

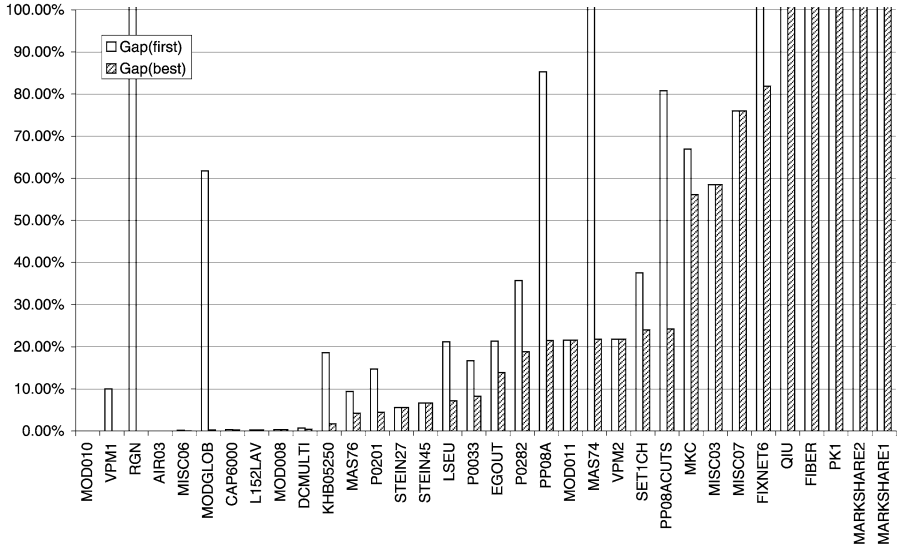

Fig. 3 Gap for the Frank-Wolfe sum method

out of 40, as opposed to 32 and 31 for the ratio and the regular sum methods, respectively). The Frank-Wolfe sum method was also most successful in finding a feasible solution with a gap of at most 100% (29 problems versus 28 and 27 respectively for ratio and regular sum). The ratio method, on the other hand, was most successful in finding a solution with a gap of at most 10% (18 problems versus 16 and 17 for regular and Frank-Wolfe sum). Thus, we conclude that, while ratio typically provides

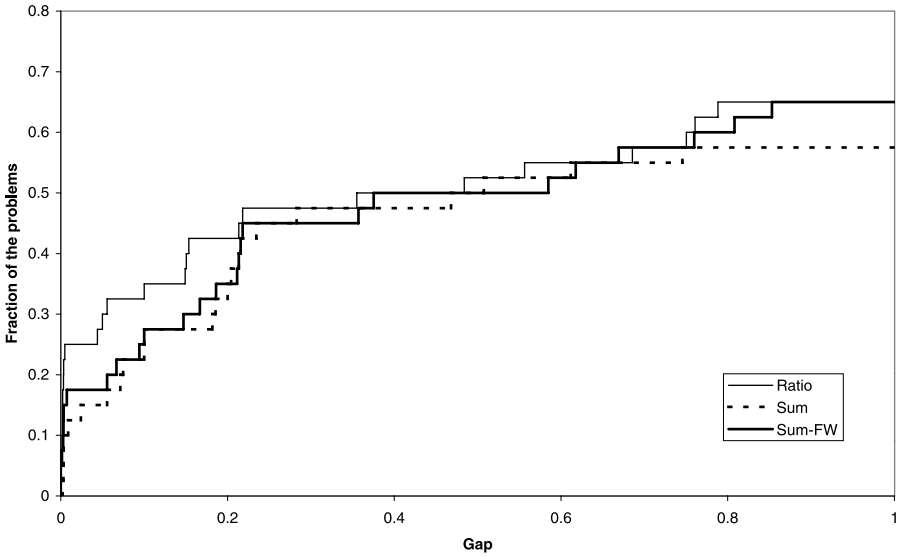

Fig. 4 Distribution of the gap of the first solution

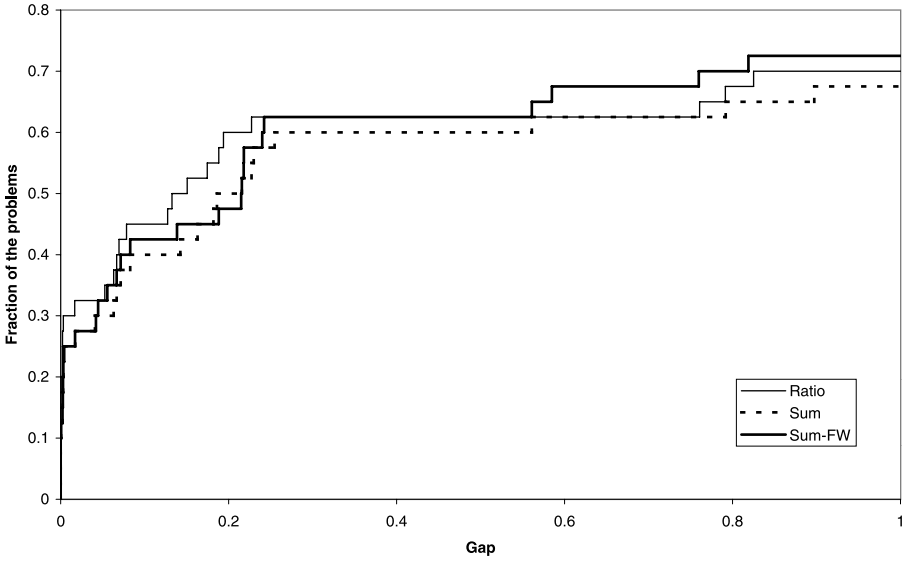

Fig. 5 Distribution of the gap of the best solution

better solutions, the Frank-Wolfe sum method is more robust. This conclusion is also supported by all 16 runs of the Frank-Wolfe sum method being successful in finding something feasible on 27 problems; this number is, respectively, 23 and 22 for ratio and regular sum.

Table 1 Number of problems by a gap category for the three methods

Category	Ratio		Sum		FW Sum	
	First	Best	First	Best	First	Best
optimal	1	3	0	3	1	2
$0\% < \text{gap} \leq 10\%$	13	15	11	13	10	15
$10\% < \text{gap} \leq 100\%$	12	10	12	11	15	12
$\text{gap} > 100\%$	6	4	8	4	8	5
Failure	8		9		6	

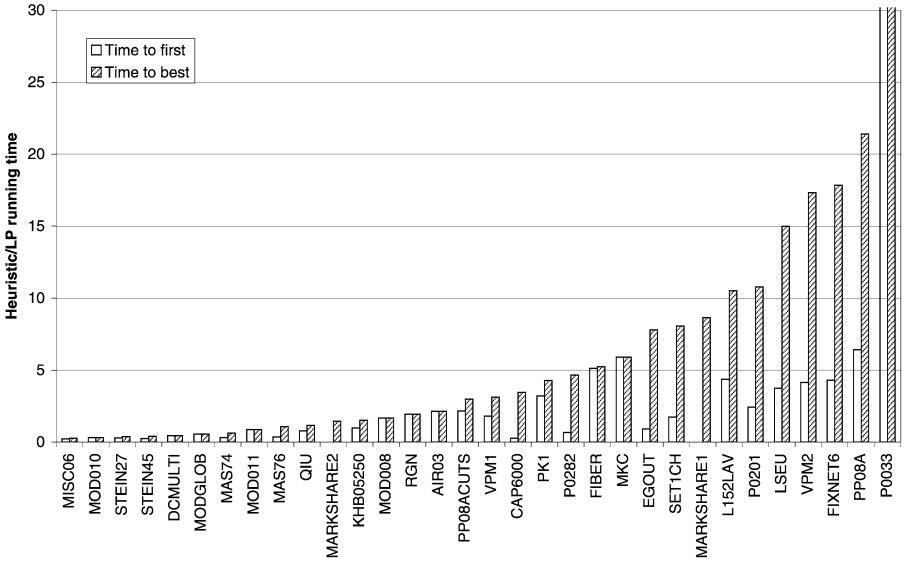

Fig. 6 Heuristic run time relative to LP for the ratio method

In terms of the run time, we start by looking at the comparison between the time to first and the time to best, relative to the time to solve the respective linear program. Figures 6, 7, and 8 present such relative times for the ratio, sum and Frank-Wolfe sum methods, respectively. We again only present successfully solved problems and order them by the time to best. Note that the time to first and even the time to best are mostly comparable to the solution time of the LP relaxation. This relation is especially pronounced for the Frank-Wolfe sum method. Also, the time to first is often just a small fraction of the time to best.

The empirical distributions functions of the time to first, the time to best, and the average run time (all relative to LP run time) in Figs. 9, 10 and 11 show the regular sum method to be a clear loser, and the Frank-Wolfe sum method to be a clear winner, another indication of robustness of the Frank-Wolfe sum method.

Time performance information both in absolute and relative terms is summarized in Table 2, where T_{first}, T_{best}, and T_{LP} denote time to first, time to best, and LP run time, respectively. The Frank-Wolfe sum method dominates with respect to all char-

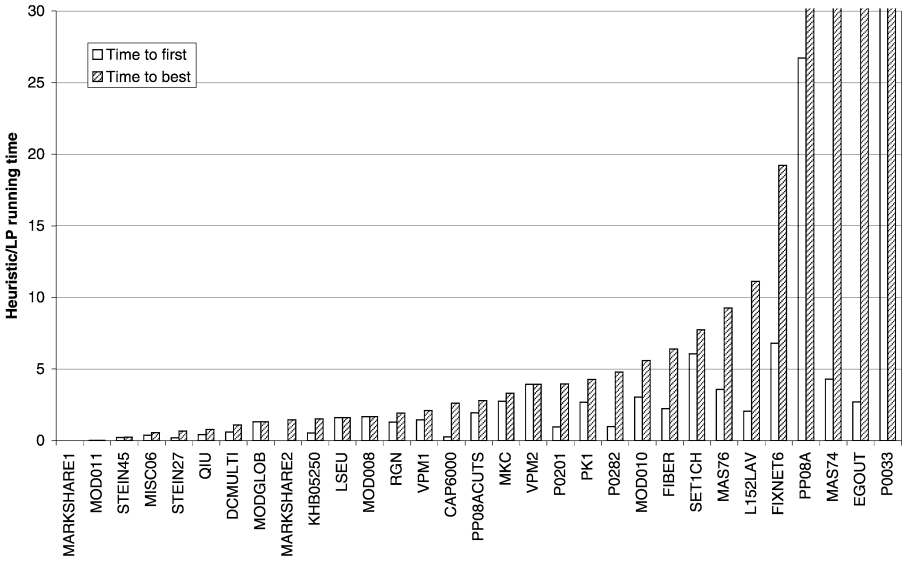

Fig. 7 Heuristic run time relative to LP for the sum method

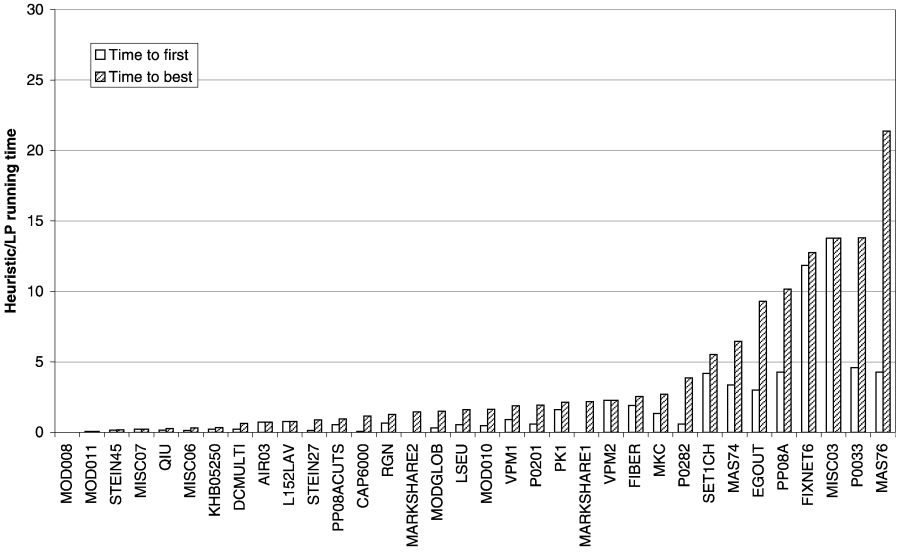

Fig. 8 Heuristic run time relative to LP for the Frank-Wolfe sum method

acteristics, and the time to first, and even the time to best, is quite frequently less than time to solve the LP. Moreover, the time to first was shorter than $20T_{\mathrm{LP}}$ on all successfully solved problems for the ratio and Frank-Wolfe sum methods. Even the time to best was less the $20T_{\mathrm{LP}}$ on all but one successfully solved problem for these two methods. Moreover, even the stand-alone version of the heuristic is likely to benefit from the full parallel implementation involving communication between the runs.

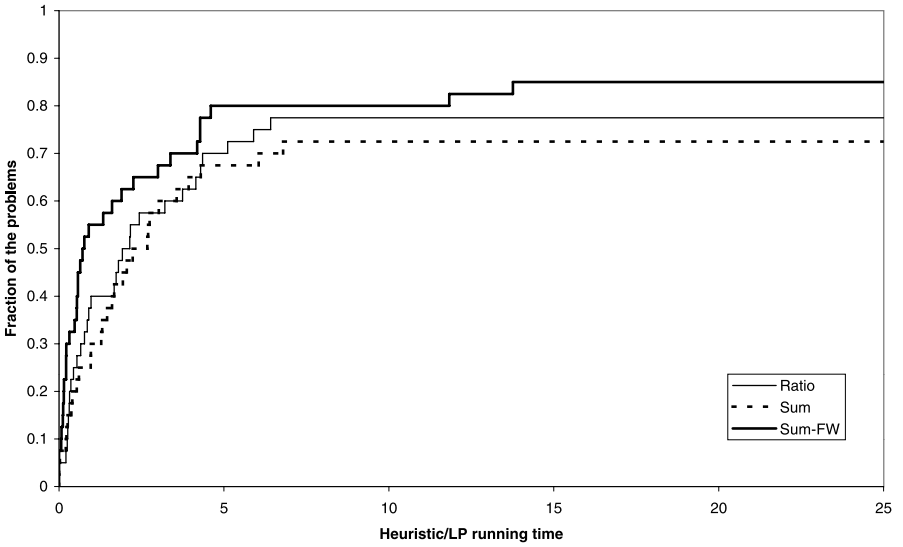

Fig. 9 Distribution of the relative time to first

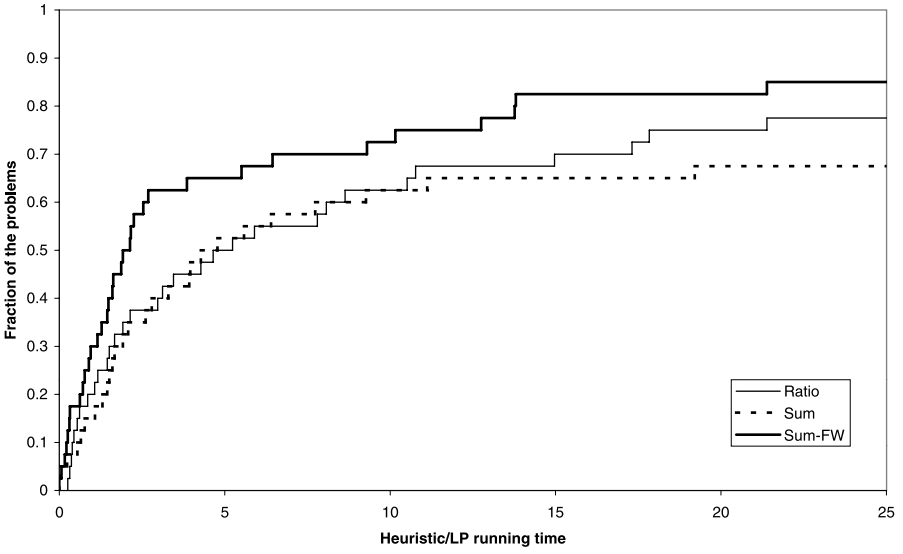

Fig. 10 Distribution of the relative time to best

Indeed, the results show that the time to best was shorter than the average run time on 24, 26, and 19 problems for the ratio, sum, and Frank-Wolfe sum method, respectively. In each case, these counts are well above a half of all successfully solved problems (31, 29 and 34, respectively).

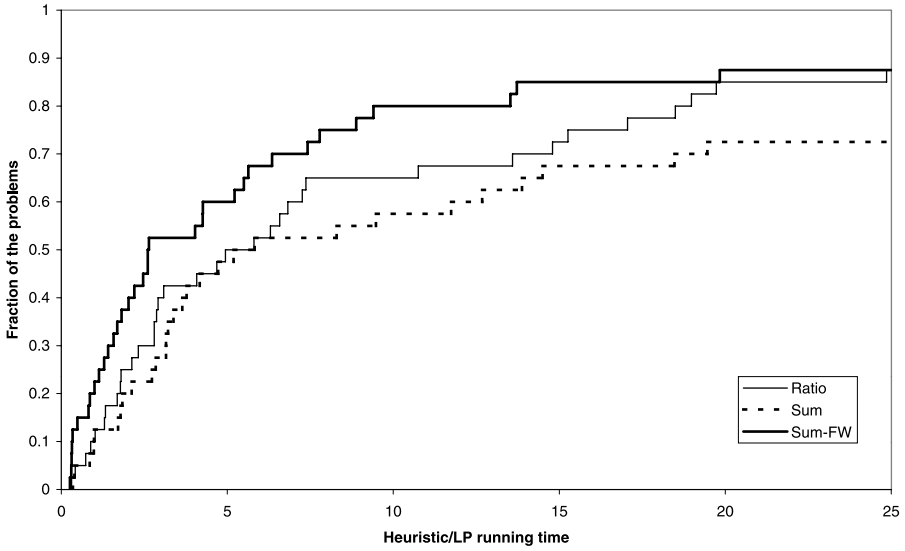

Fig. 11 Distribution of the average relative run time

Table 2 Number of problems by a run time category for the three methods

Category	Ratio	Sum	FW sum
$T_{\text{first}} < 1$ sec	23	21	25
$T_{\text{best}} < 1$ sec	19	16	23
$T_{\text{first}} < T_{\text{LP}}$	16	12	22
$T_{\text{best}} < T_{\text{LP}}$	8	6	12
$T_{\text{first}} < 20T_{\text{LP}}$	31	29	34
$T_{\text{best}} < 20T_{\text{LP}}$	30	27	33

3.4 Algorithm profile

We now consider how frequently the various features of the heuristic are used; for brevity, we restrict out attention to the ratio variant.

First, Fig. 12 shows the average number of probing steps for each problem. The problems are ordered by increasing gap of the best solution over 16 runs, with the vertical scale truncated at 150 probing steps. The number of probing steps seems to increase with the gap, which can be roughly associated with the difficulty of a problem for this heuristic. There are a few exceptions to this rule, such as problems `p0033`, `p0201` and `10teams`. We note that `p0033` and `p0201` are solved successfully, but their structure requires many probing steps. On the other hand, `10teams` is highly degenerate, and degeneracy seems to be one of the main reasons for failure to find any integer-feasible solutions within a specified iteration limit.

Figure 13 displays the average number of cuts for each problem. Note that a significant number of cuts are sometimes required for a solution, even for the problems with relatively small gaps, such as `l152lav` and `cap6000`. Again, problems

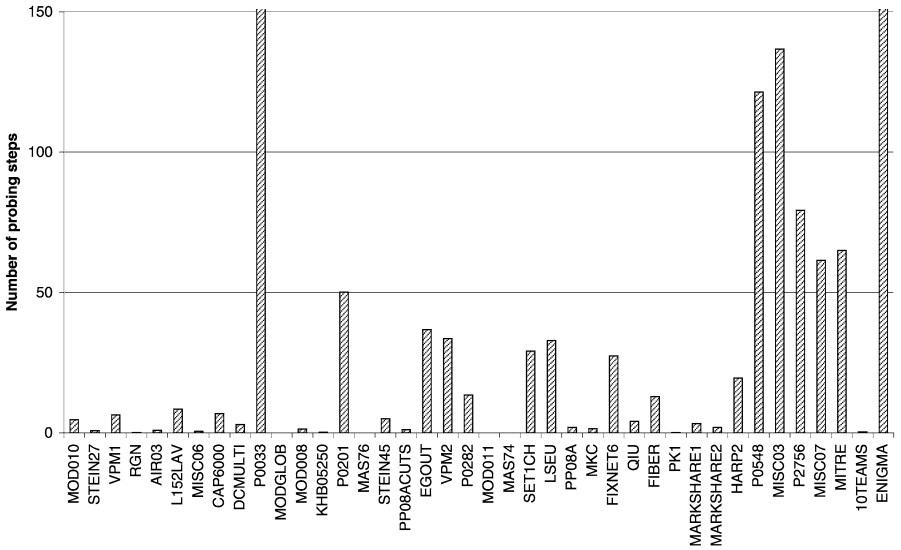

Fig. 12 Average number of probing steps in the ratio method

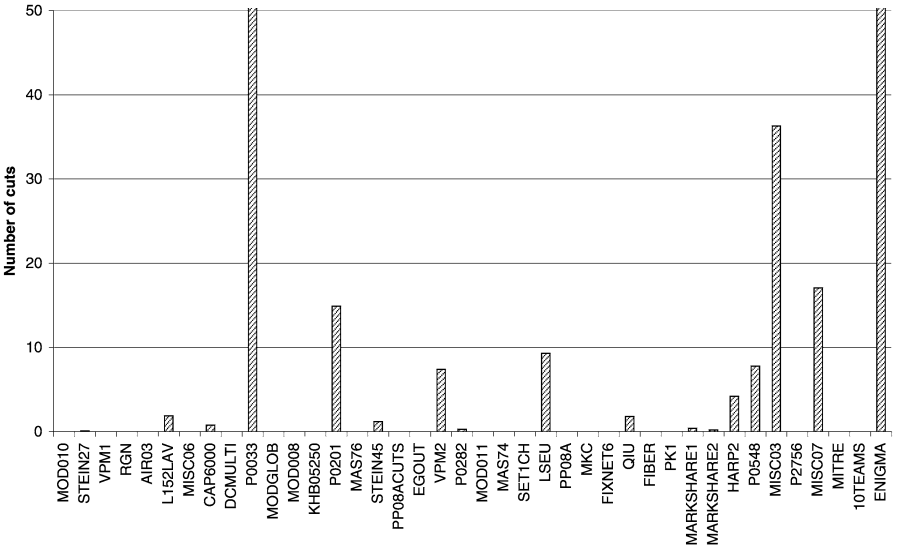

Fig. 13 Average number of cuts in the ratio method

`p0033` and `p0201` require quite large numbers of cuts compared to other problems of similar difficulty. For readability, the graph was truncated at 50 cuts, but `p0033` in fact requires around 100 cuts on average.

Examining the evolution of the objective and merit functions, we observed strikingly varied behavior, even for the problems adjacent in Fig. 13. Consider, for example, the ratios of the objective and the merit function to their respective initial values

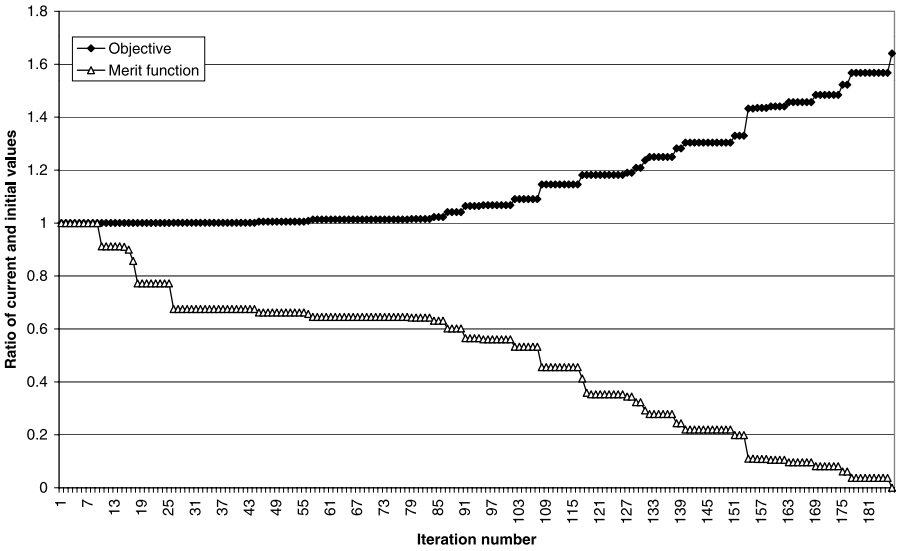

Fig. 14 Evolution of a typical run for `modglob`

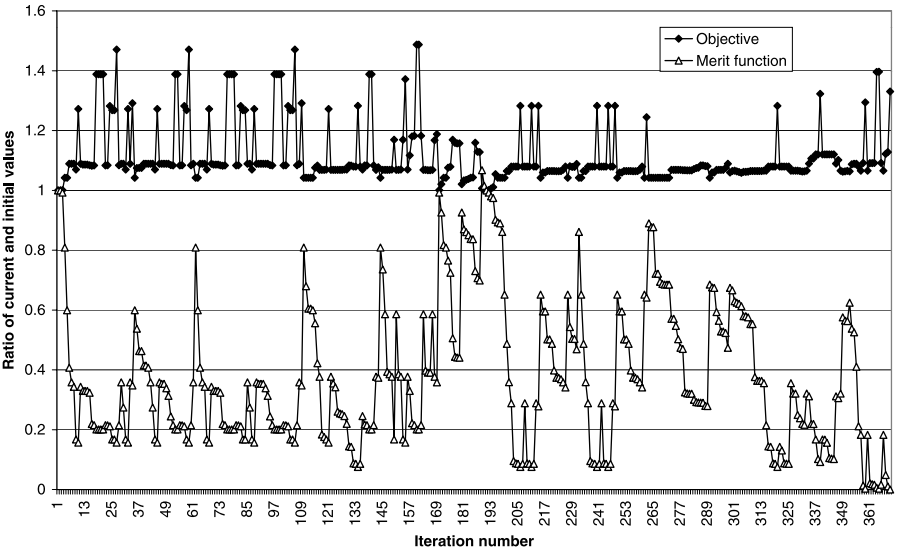

Fig. 15 Evolution of a typical run for `p0033`

for problems `modglob` and `p0033`; see Figs. 14 and 15. The run for `modglob` seems to make a steady progress toward integer feasibility. The flat areas of the graph correspond to sequences of degenerate pivots. On the other hand, for `p0033`, the behavior of both the merit and objective functions is highly erratic. Spikes in the merit function value correspond to cut additions with subsequent pivot backtracking.

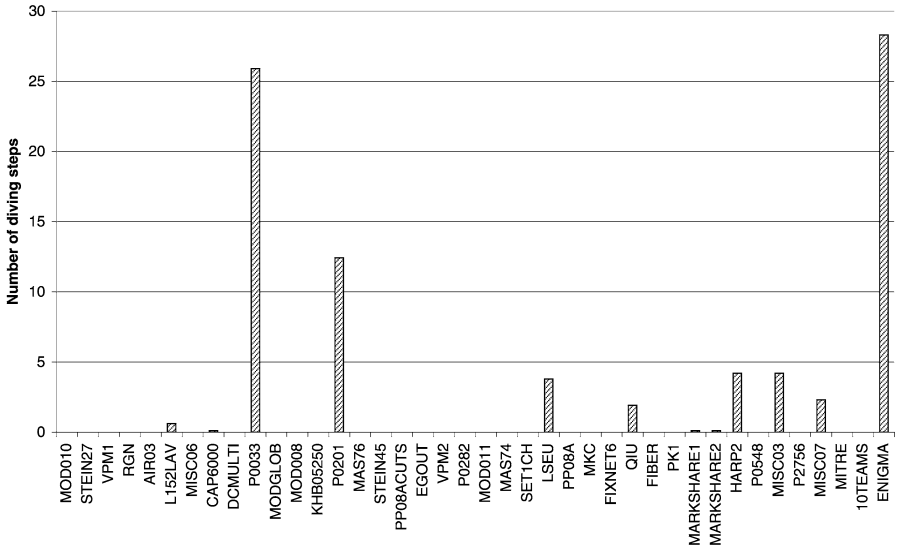

Fig. 16 Average number of diving steps in the ratio method

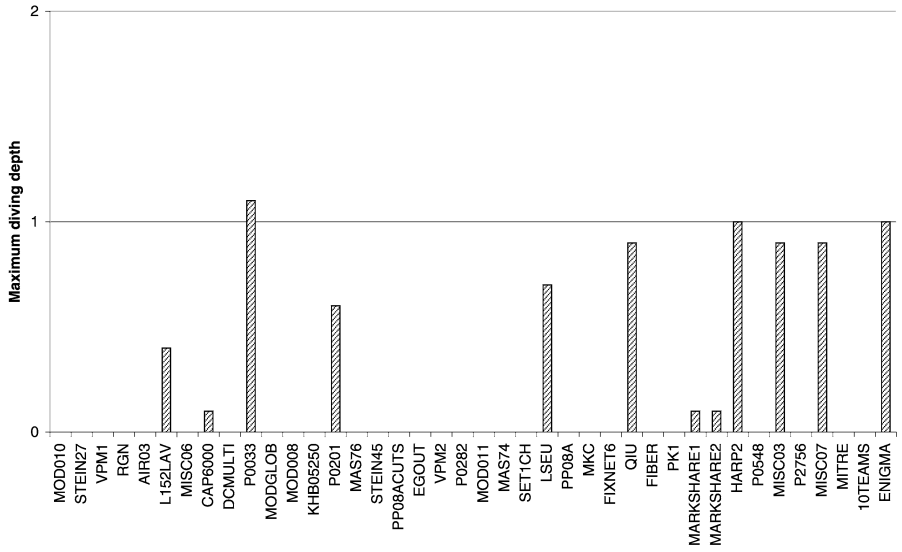

Fig. 17 Average maximum diving depth in the ratio method

We now consider the average number of diving steps and the average maximum diving depth, as displayed in Figs. 16 and 17. The "depth" refers to the number of steps in this bound-fixing process until a subproblem is infeasible or an integer-feasible solution is found. We see that application of diving steps is required across the entire range of problem difficulty. As for the maximum diving depth, it is quite often nonzero (meaning the first diving subproblem was feasible) but rarely exceeds 1.

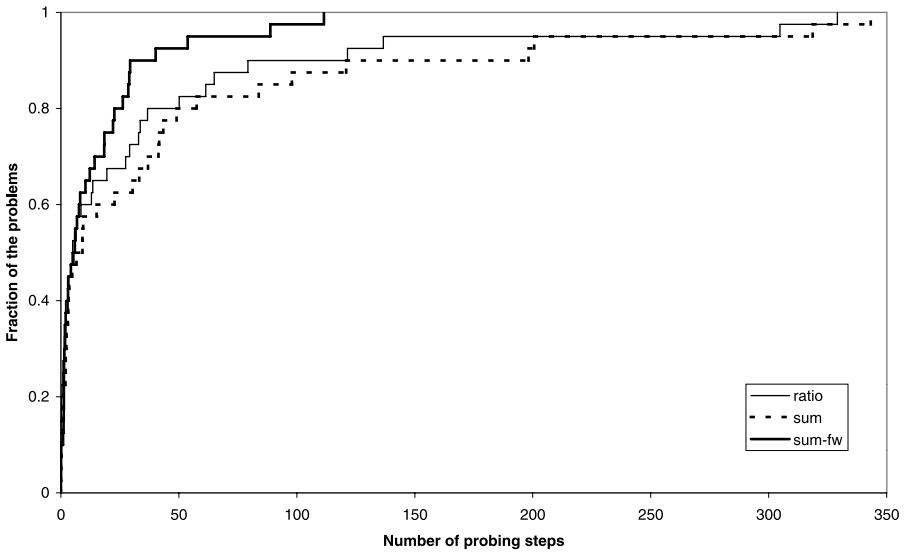

Fig. 18 Distribution of the average number of probing steps

The above profiling results show that all features of Pivot, Cut, and Dive may be useful, depending on the problem instance.

Finally, we present empirical data verifying our hypothesis that the Frank-Wolfe variation of the sum method requires lower number of probing steps. Figure 18 presents empirical distribution functions of the number of probing steps for the three methods. Clearly, the Frank-Wolfe sum method has the fewest such steps as its distribution function is highest in the graph. It appears that the Frank-Wolfe approach is "pivoting through" vertices identified as local minima by the other methods.

The variants presented here are not mutually exclusive; for example, one could switch between the Franke-Wolfe sum method and the ratio methods in mid-run depending on the amount of degeneracy detected and the apparent progress towards integrality.

Acknowledgements This work was supported in part by U.S. NSF grant CCR-9902092 and by DIMACS. We thank the referees of this and earlier versions of this paper for their comments and constructive suggestions.

References

Anbil, R., Barahona, F., Rushmeier, R., Snowdon, J.: IBM makes advances in airline optimization. Research Report RC21465(96887), IBM T.J. Watson Research Center, Yorktown Heights, NY (1999)

Balas, E.: Intersection cuts—a new type of cutting planes for integer programming. Oper. Res. **19**, 19–39 (1971)

Balas, E.: Integer programming and convex analysis: intersection cuts from outer polars. Math. Program. **2**, 330–382 (1972)

Balas, E., Martin, C.H.: Pivot-and-complement—a heuristic for 0-1 programming. Manag. Sci. **26**(1), 86–96 (1980)

Balas, E., Bowman, V.J., Glover, F., Sommer, D.: An intersection cut from the dual of the unit hypercube. Oper. Res. **19**, 40–44 (1971)
Balas, E., Ceria, S., Dawande, M., Margot, F., Pataki, G.: OCTANE: a new heuristic for pure 0-1 programs. Oper. Res. **49**(2), 207–225 (2001)
Bali, S., Jacobsen, S.E.: On the convergence of the modified Tui algorithm for minimizing a concave function on a bounded convex polyhedron. In: Optimization Techniques. Proc. 8th IFIP Conf., Würzburg, 1977, Part 2. Lecture Notes in Control and Information Science, vol. 7, pp. 59–66. Springer, Berlin (1978)
Beale, E.M.L.: Branch and bound methods for mathematical programming systems. Ann. Discret. Math. **5**, 201–219 (1979). Discrete optimization (Proc. Adv. Res. Inst. Discrete Optimization and Systems Appl., Banff, Alta., 1977), II
Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific, Boston (1999)
Bixby, R.E., Ceria, S., McZeal, C.M., Savelsberg, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Technical Report 98-3, Department of Computational and Applied Mathematics, Rice University (1998)
Burdet, C.-A.: Enumerative inequalities in integer programming. Math. Program. **20**(1), 32–64 (1972)
Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. Math. Program. **102**(1), 71–90 (2005)
Eckstein, J.: Parallel branch-and-bound methods for mixed-integer programming on the CM-5. SIAM J. Optim. **4**(4), 794–814 (1994)
Eckstein, J., Nediak, M.: Depth-optimized convexity cuts. Ann. Oper. Res. **139**, 95–129 (2005)
Eckstein, J., Phillips, C.A., Hart, W.E.: PICO: an object-oriented framework for parallel branch and bound. In: Inherently Parallel Algorithms in Feasibility and Optimization and Their Applications, Haifa, 2000, pp. 219–265. North-Holland, Amsterdam (2001)
Faaland, B.H., Hillier, F.S.: Interior path methods for heuristic integer programming procedures. Oper. Res. **27**(6), 1069–1087 (1979)
Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. **104**(1), 91–104 (2005)
Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**(1–3), 23–47 (2003)
Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: A practical anti-cycling procedure for linearly constrained optimization. Math. Program. **45**(3), 437–474 (1989)
Glover, F.: Cut search methods in integer programming. Math. Program. **3**, 86–100 (1972)
Glover, F.: Convexity cuts and cut search. Oper. Res. **21**, 123–134 (1973)
Glover, F., Laguna, M.: General purpose heuristics for integer programming—part I. J. Heuristics **2**, 343–358 (1997a)
Glover, F., Laguna, M.: General purpose heuristics for integer programming—part II. J. Heuristics **3**, 161–179 (1997b)
Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Boston (1997c)
Hillier, F.S.: Efficient heuristic procedures for integer linear programming with an interior. Oper. Res. **17**(4), 600–637 (1969)
Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to Global Optimization. Kluwer Academic, Dordrecht (1995)
Ibaraki, T., Ohashi, T., Mine, H.: A heuristic algorithm for mixed-integer programming problems. Math. Program. Study **2**, 115–136 (1974)
Linderoth, J.T.: Topics in parallel integer optimization. Ph.D. thesis, Department of Industrial and Systems Engineering, Georgia Institute of Technology (1998)
Løkketangen, A., Glover, F.: Solving zero/one mixed integer programming problems using tabu search. Eur. J. Oper. Res. **106**, 624–658 (1998)
Raghavachari, M.: On connections between zero-one integer programming and concave programming under linear constraints. Oper. Res. **17**, 680–684 (1969)
Ralphs, T.K., Ladányi, L.: SYMPHONY User's Manual: preliminary draft. http://branchandcut.org/SYMPHONY/man/man.html (2000)
Tuy, H.: Concave programming under linear constraints. Sov. Math. **5**(6), 1437–1440 (1964)
Tuy, H.: Normal conical algorithm for concave minimization over polytopes. Math. Program. **51**(2), 229–245 (1991)
Vanderbei, R.J.: Linear Programming: Foundations and Extensions, 2nd edn. Kluwer Academic, Boston (2001)
Zwart, P.B.: Nonlinear programming: counterexamples to two global optimization algorithms. Oper. Res. **21**(6), 1260–1266 (1973)
Zwart, P.B.: Global maximization of a convex function with linear inequality constraints. Oper. Res. **22**(3), 602–609 (1974)