Noname manuscript No.
(will be inserted by the editor)

An Empirical Evaluation of a Walk-Relax-Round Heuristic for Mixed Integer Convex Programs

Kuo-Ling Huang · Sanjay Mehrotra

Received: date / Accepted: date

Abstract Recently, a walk-and-round heuristic was proposed by Huang and Mehrotra [26] for generating high quality feasible solutions of mixed integer linear programs (MILPs). This approach uses geometric random walks on a polyhedral set to sample points in this set. It subsequently rounds these random points using a heuristic, such as the feasibility pump. In this paper, the walk-and-round heuristic is further developed for the mixed integer convex programs (MICPs). Specifically, an outer approximation relaxation step is incorporated. The resulting approach is called a walk-relax-round heuristic. Computational results on problems from the `CMU-IBM` library show that the points generated from the random walk steps bring additional value. Specifically, the walk-relax-round heuristic using a long step Dikin walk found an optimal solution for 51 out of the 58 MICP test problems. In comparison, the feasibility pump heuristic starting at a continuous relaxation optimum found an optimal solution for 45 test problems. Computational comparisons with a commercial software `Cplex` 12.1 on mixed integer convex quadratic programs (MIQPs) are also given. Our results show that the walk-relax-round heuristic is promising. This may be because the random walk points provide an improved outer approximation of the convex region.

Keywords Mixed integer convex programs · geometric random walk · feasibility pump · primal heuristic.

The research of both authors was partially supported by grant DOE-SP0011568 and ONR N00014-09-10518, N00014-210051.

S. Mehrotra
Dept. of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL, 60208
E-mail: mehrotra@iems.northwestern.edu

1 Introduction

We consider the problem of generating good quality solutions for mixed integer convex programs (MICPs) of the form:

$$\min \ c(x) \tag{1}$$

$$\text{s.t. } x \in \mathcal{C} := \left\{ x \;\middle|\; \begin{array}{l} g_i(x) \geq 0, i = 1, \ldots, \tilde{m} \\ a_i(x) = 0, i = 1, \ldots, \hat{m} \\ 0 \leq x \leq u \end{array} \right\} \subseteq \mathbb{R}^n, \tag{2}$$

$$x_j \in \mathbb{Z} \text{ for each } j \in \mathcal{I}, \tag{3}$$

where x are the decision variables, u is the vector of the upper bounds on x, $\mathcal{I}$ represents the set of integer elements in x, $c(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is a convex objective function, $g_i(\cdot) : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, \tilde{m}$ are concave, and $a_i(x) := A_i x - b_i$, $i = 1, \ldots, \hat{m}$, where $A_i \in \mathbb{R}^n$ is as a row vector, and $b_i \in \mathbb{R}$. We let $A := [(A_i)_{i=1,\ldots,\hat{m}}] \in \mathbb{R}^{\hat{m} \times n}$, and $b := [(b_i)_{i=1,\ldots,\hat{m}}] \in \mathbb{R}^{\hat{m}}$. We assume that the set $\mathcal{C}$ is bounded, and it has a nonempty relative interior.

Quickly finding a good quality solution of mixed integer programs is important for multiple reasons. A branch-and-cut algorithm uses available feasible solutions to efficiently prune segments of the search tree. Additionally, an improved quality solution allows for early termination of the algorithm if the desired tolerance is met. Hence, significant research effort has been spent towards developing different heuristics for finding a quality solution, particularly for the general mixed integer linear programs (MILPs). Among the known heuristics, the feasibility pump (FP) heuristic [15] is known to perform well for the unstructured problems. For a survey of the progress towards solving MICPs, see [17].

In a recent paper (Huang and Mehrotra [26]) we proposed a walk-and-round heuristic that combines the use of random walks [27,29,30,36] with a rounding heuristic for generating feasible solutions of MILPs. The FP [22] heuristic was used for the rounding step. The walk-and-round heuristic generates points in a polytope using a random walk. The walk is started from an interior solution, typically from near the log-barrier center of the polytope. It then attempts to round a random point to a feasible integer solution using the FP heuristic. In theory, the points generated after a sufficient number of random walk steps is known to follow a nearly uniform distribution on the feasible set [27,29,30,36]. Consequently, in theory the walk-and-round heuristic explores the continuous relaxation set by uniformly sampling on the set.

The FP heuristic was initially proposed by Fischetti *et al.* [22], and was later extended to mixed integer nonlinear programs (MINLPs). Fischetti *et al.* [22] showed that the FP heuristic is effective in generating feasible solutions for mixed binary linear problems (0-1 MILPs). This led to several subsequent papers for general MILPs [7,10,13,24]. In the context of MICPs, Bonami and Gonçalves [16] and Abhishek *et al.* [6] have shown that the basic principle of the FP heuristic can be used to find good solutions of MICPs in reasonable computational times. Bonami *et al.* [15] proposed a different variant of the

FP heuristic for MICPs that is based on outer approximation (OA) of the problem constraints. D'Ambrosio *et al.* [18,19] extended the FP for general MINLPs.

In (Huang and Mehrotra [26]) we adapt the hit-and-run walk [29,30,36] and Dikin-type walks [27]. Instead of taking a long walk to comply with the theory of generating a uniformly sampled point, only a few random walk steps are taken, and the resulting point is rounded. Additionally, FP is terminated if it fails to find a feasible integer solution within a specified termination criteria; and a new random point is generated to start another FP search. If a feasible integer solution is found, an objective constraint is used to cut off this feasible integer solution, and the method is restarted. The computational experiments in [26] showed that for a large number of test problems, an improved feasible solution is generated from the walk-and-round heuristic, when compared with the solution generated from the standard FP heuristic in the same amount of run time.

The goal of the current paper is to develop the walk-and-round heuristic for MICPs, and study its performance. Specifically, the use of an OA to the convex constraints is incorporated in the walk-and-round framework. The main modification to the standard FP approach for MICPs is that the OA information is generated at the points obtained from the random walk and subsequent FP points. The resulting heuristic is called the walk-relax-round heuristic. The walk-relax-round approach strategically reuses (or abandons) the generated OA constraints in the subsequent FP search, based on the quality of the current incumbent solution or the number of random walk steps. This heuristic is implemented in a software package named `iOptimize`. We test this walk-relax-round heuristic on 58 convex MICP instances taken form the `CMU-IBM` library Open source MINLP Project [2].

We find that the walk-relax-round heuristic is promising for MICPs. For most problems, using a similar computational effort, the walk-relax-round heuristic generates better solutions than FP running at either an optimal solution, or the analytic center of the continuous relaxation. In particular, the walk-relax-round heuristic using a long step Dikin walk finds an optimal solution for 51 out of the 58 test problems. In comparison, FP starting at the continuous relaxation solution an optimal solution for 45 test problems. A computational comparison with a commercial software `Cplex` 12.1 on nine small and medium size mixed integer quadratic programs (MIQPs) taken from the Hans D. Mittelmann's collections [4] shows that walk-relax-round method generates comparable computational results as `Cplex`.

The rest of this paper is organized as follows. In Section 2 we outline two different random walks, namely the hit-and-run walk, and the Dikin walk. In Section 3 we outline the walk-relax-round heuristic for solving MICPs arising in practice. In this section we also describe the FP heuristic used for rounding the sample points. In Section 4 we provide additional implementation details. In Section 5 we present and discuss computational results. Here the random walk performances are compared on a set of general MICPs taken from the `CMU-IBM` open source MINLP Project. Computational comparison

with `Cplex` 12.1 on MIQPs is also provided in this section. Finally, we conclude in the last section.

2 Random walk methods for a general convex set

Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a full dimensional convex set. A random walk on this set is a Markov chain that starts at some point $x^0 \in \mathcal{K}$. At step k, it moves from $x^k \in \mathcal{K}$ to a randomly chosen point $x^{k+1} \in \mathcal{K}$ which depends only on the current point x^k. Using a suitable walk step (see below) x^k is near uniformly sampled over $\mathcal{K}$ for a sufficiently large value of k. The number of walk steps required to obtain a near uniformly sampled point is called the mixing time of the walk.

Several walks have been studied in the literature. These include the hit-and-run walk first proposed by Smith [35] (see also [29,30,36]) and the Dikin-walk proposed by Kannan and Narayanan [27]. The random walks have been used to approximately count the number of lattice points in a polytope [28], sampling lattice over an integer hyper-rectangle [12], approximating the volume of a convex body [30], and solving convex optimization problems [14,27,38]. Lovász [29] showed that the hit-and-run walk has $O^*(n^2R^2/r^2)$ mixing time, where R and r are the radii of the inscribed and circumscribed balls of $\mathcal{K}$. Kannan and Narayanan's Dikin walk uses Dikin ellipsoids with a Metropolis modification. They showed that when started from the log-barrier center, the Dikin walk for sampling from a full dimensional polyhedral has a strongly polynomial mixing time. Narayanan [34] extended this walk for sampling from $\mathcal{K}$ equipped with a self-concordant barrier, and showed that the mixing time of the Dikin walk remains strongly polynomial for this set.

Let $w_j = u_j - x_j, j = 1, \ldots, n$. The log-barrier center of $\mathcal{C}$ is defined as the solution of

$$\min \left\{ B(x) := -\sum_{i=1}^{\tilde{m}} \ln g_i(x) - \sum_{j=1}^{n} \ln x_j - \sum_{j=1}^{n} \ln w_j \,\middle|\, a_i(x) = 0, i = 1, \ldots, \hat{m} \right\}. \tag{4}$$

Let $X = [diag(x_j)_{j=1,\ldots,n}]$ (resp. $W = [diag(w_j)_{j=1,\ldots,n}]$) be a diagonal matrix whose diagonal elements are x_j (resp. w_j), and let $e = (1, \ldots, 1)^T$. The gradient and Hessian of the log-barrier at a point x are given by

$$\begin{aligned} \nabla B(x) &= -e^T Z^{-1} J_g + e^T X^{-1} + e^T W^{-1} \text{ and} \\ \nabla^2 B(x) &= J_g^T Z^{-2} J_g + H + X^{-2} + W^{-2}, \end{aligned} \tag{5}$$

where $Z := [diag(g_i(x))_{i=1,\ldots,\tilde{m}}]$ is a diagonal matrix whose diagonal elements are $g_i(x)$, $J_g := [\nabla g_i(x)_{i=1,\ldots,\tilde{m}}]$ is the Jacobian matrix, and $H := -\sum_{i=1}^{\tilde{m}} \frac{\nabla^2 g_i(x)}{g_i(x)}$.

Although the random walks are naturally described over a full dimensional set, the set $\mathcal{C}$ is not full dimensional due to the presence of equality constraints.

In some cases one may be able to reformulate $\mathcal{C}$ as a full dimensional set using reduced space reformulations. However, the sparsity of the original problem data is typically destroyed in such reformulations. Hence, reformulation and the full dimensional assumption are not practical in general. Following Huang and Mehrotra [26], in the following we present modified versions of the hit-and-run [36] and the Dikin walk [27,34] over $\mathcal{C}$. These random walks are modified from the original walks by adding a step that projects a random direction on the equality in the sense that they are not described over a full dimensional convex body

We assume that all these walks are started from near the log-barrier analytic center. Implementation details for the log-barrier analytic center computations will be given in Section 4.1.1.

2.1 Projected hit-and-run walk for a general convex set

Given a full dimensional convex set $\mathcal{K}$, the basic scheme of the hit-and-run walk is as follows:

- Pick a uniformly distributed random line p through the current point.
- Move to a uniform random point along the chord $p \bigcap \mathcal{K}$.

Now, given a general convex set $\mathcal{C} \subseteq \mathbb{R}^n$, the projected hit-and-run walk first computes a random direction d in $\mathbb{R}^n$. Next, it orthogonally projects d onto the affine space given by the equality constraints $Ax = b$, yielding direction p. Finally, it computes a random point from the line segment $\ell := \{x + \lambda p\} \bigcap \mathcal{C}$ as follows. The line segment ℓ is computed (in Step 9 in Algorithm 2.1) by using a simple backtracking line-search heuristic. First, we compute the maximum and the minimum of the step-size, λ^+ and λ^-, in the following:

$$\lambda^+ := \max\{\lambda \mid 0 \leq x + \lambda p \leq u\} \text{ and } \lambda^- := \min\{\lambda \mid 0 \leq x + \lambda p \leq u\}.$$

Next, we check if $x + \lambda^+ p \in \mathcal{C}$ (resp. $x + \lambda^- p \in \mathcal{C}$). If not, we let $\lambda^+ := 0.9\lambda^+$ (resp. $\lambda^- := 0.9\lambda^-$). The procedure is continued until λ^+ and λ^- become a feasible step-size. The point x^{k+1} is picked by computing a uniform random number on (λ^-, λ^+) and taking the corresponding step. The projected hit-and-run walk is summarized in Algorithm 2.1.

Note that the direction p in Step 5 is computed by solving $\min_{Ap=0} \|p-d\|^2$. It can be verified that the explicit solution to this problem is $p = [I - A^T(AA^T)^{-1}A]d$. If, however, no equality constraint is presented in $\mathcal{C}$, the projection step (Step 5) is not needed. In this case Algorithm 2.1 reduces to the original hit-and-run walk.

2.2 Modified Dikin walk for a general convex set

The Dikin walk in Kannan and Narayanan [27] is a "Metropolis" type walk which picks a move and then decides whether to "accept" the move, or "reject"

Algorithm 2.1 *Projected hit-and-run walk for a convex set*

Input: A starting point $x^0 \in \mathcal{C}$ and the maximum number of walk steps $k_{\max}$.
Output: Random points $x^k \in \mathcal{C}, k = 1, \ldots, k_{\max}$.

Set $k := 0$.
while $k < k_{\max}$ **do**
  Pick a uniform random direction d in $\mathbb{R}^n$.
  if $\hat{m} \neq 0$ (affine equation exists) **then**
    Compute $p := [I - A^T(AA^T)^{-1}A]d$.
  else
    Set $p := d$.
  end if
  Find the line segment $\ell := \{x \mid x + \lambda p, \lambda \in \mathbb{R}\} \bigcap \mathcal{C}$.
  Uniformly pick a random point x^{k+1} from the line segment ℓ.
  Set $k := k + 1$.
end while

and stay at the current iterate with a repeated random direction computation. The move step is picked inside a Dikin ellipsoid centered at the current point in the convex set.

Huang and Mehrotra [26] proposed a modified Dikin walk by ignoring its Metropolis step. This modification is made because the computation of determinants as required in the original Dikin walk is not practical when equality constraints are present. We describe an adaptation of this modified Dikin walk in the following.

The Dikin ellipsoid of radius r centered at $x^c \in \mathcal{C}$ is defined as follows:

$$\mathcal{E}(x^c, \nabla^2 B(x^c); r) := \left\{x \mid p^T \nabla^2 B(x^c) p \leq r^2, Ax = b, 0 \leq x \leq u\right\},$$

where $p := (x - x^c)$, and $B(\cdot)$ and $\nabla^2 B(\cdot)$ are defined in (4) and (5).

A random direction $d \in \mathbb{R}^n$ is projected onto the null space of $A(\nabla^2 B(x^k))^{-1/2}$ to compute a random walk direction $(\nabla^2 B(x^k))^{-1/2} d$. Equivalently, the following ellipsoid constrained optimization problem is solved:

$$\begin{aligned} \min \quad & p^T d \\ \text{s.t.} \quad & Ap = 0, \\ & p^T \nabla^2 B(x^k) p = r^2. \end{aligned} \tag{6}$$

Following Ye [37, Chapter 3], it can be verified that the optimal solution to problem (6) is

$$p = \frac{r(\nabla^2 B(x^k))^{-1/2} p(x)}{\|p(x)\|}, \tag{7}$$

where

$$p(x) = \left[I - (\nabla^2 B(x^k))^{-1/2} A^T (A(\nabla^2 B(x^k))^{-1} A^T) A (\nabla^2 B(x^k))^{-1/2}\right] (\nabla^2 B(x^k))^{-1/2} d.$$

Algorithm 2.2 *Modified Dikin walk for a convex set*

Input: A starting point $x^0 \in \mathcal{C}$ and the maximum number of walk steps $k_{\max}$.
Output: Random points $x^k \in \mathcal{C}, k = 1, \ldots, k_{\max}$.

1: Set $k \coloneqq 0$.
2: **while** $k < k_{\max}$ **do**
3: Generate a random direction d uniformly distributed over a direction set in $\mathbb{R}^n$.
4: Compute p from (7).
5: Compute a new point x^{k+1} by using **Strategy I** or **Strategy II**.
6: **end while**

Once p is obtained, a new random point is computed by using one of the following strategies.

- **Strategy I** (Short step strategy): Take r to be a constant (we used $r = 0.95$) and λ uniformly distributed in $(0, 0.95]$. Let $x^{k+1} \coloneqq x^k + \lambda p$.
- **Strategy II** (Long step strategy): Uniformly pick a random point x^{k+1} from a line segment $\ell \coloneqq \{x \mid x + \lambda p, \lambda \in \mathbb{R}\} \bigcap \mathcal{C}$.

We note that the implementation for **Strategy II** is similar to that in the Projected hit-and-run walk (for details, see the discussion on the Step 9 in Algorithm 2.1).

Finally, we note that if we take $\nabla^2 B(x^k) = I$, then the modified Dikin walk with Strategy II studied in Algorithm 2.2 reduces to the projected hit-and-run walk. Hence, the hit-and-run walk does not use information from the geometric structure of $\mathcal{C}$, whereas the modified Dikin walk uses a local ellipsoidal approximation to capture this information. The pseudocode of the modified Dikin walk is described in Algorithm 2.2.

3 Walk-relax-round heuristic for practical MICPs

We now give a description of our walk-relax-round heuristic. Implementation details are given in Section 4. Starting from a near-analytic center of $\mathcal{C}$, defined in (2), we use the random walks described in the previous section to generate a random point in $\mathcal{C}$. Next, we perform a rounding heuristic that attempts to find a feasible integer solution satisfying the integrality requirements defined in (3). If the rounding heuristic fails to find a feasible integer solution within a specified termination criteria, we generate another random point from the current one by taking additional walk steps. This point is now rounded by using the rounding heuristic again. On the other hand, if a new (better) integer solution is found, we modify the feasible set $\mathcal{C}$ by introducing a (or updating the) objective cutoff constraint, which prevents the walk-relax-round heuristic from finding another integer solution with the same objective value. This will be discussed later. Afterward, we recompute a new (approximate) analytic center for the new feasible set, and restart the walk-relax-round heuristic.

A core step in the proposed walk-relax-round heuristic is to round a solution violating integrality requirements. This rounding is performed using an implementation of the feasibility pump (FP) heuristic. In the following we first provide necessary details about FP. Next, we present a soluiton polishing phase to improve the quality of the solutions found by FP.

3.1 Outer approximation based feasibility pump for MICPs

The basic scheme of FP is described below. Starting from a point $\bar{x} \in \mathcal{C}$, the method searches for a point x^* that is heuristically as close as possible to a rounded integer solution $\hat{x}$ of $\bar{x}$ by solving an l_1-norm minimization problem. For a given point $\bar{x}^k \in \mathcal{C}$, let $\hat{x}$ be the integer solution obtained by rounding $\bar{x}^k_j$ to the nearest integer for each $j \in \mathcal{I}$, and keep the other components equal to $\bar{x}^k_j$. The l_1-norm minimization problem is of the form

$$\begin{aligned} \min \ & \sum_{j \in \mathcal{I}} |x_j - \hat{x}^k_j| \quad \text{(FP-NLP)} \\ \text{s.t.} \ & x \in \mathcal{C}. \end{aligned} \tag{8}$$

Let x^{k*} be an optimal solution of (8). If $\sum_{j \in \mathcal{I}} |x^{k*}_j - \hat{x}^k_j| = 0$, we have a feasible solution for (1)–(3), else we take $\bar{x}^{k+1} \coloneqq x^{k*}$, set $k \coloneqq k+1$, and start a new iterate. The algorithm iterates until a maximum iteration or some other criteria (e.g., time limit) is reached.

Once a feasible integer solution x^{k*} is generated, the following artificial constraint is used to cutoff all inferior solutions:

$$c(x) \leq c(x^{k*}) - \epsilon, \tag{9}$$

where ϵ is a small positive constant. This objective cutoff constraint guarantees that the next feasible integer solution generated by the FP procedure is always better than the current one. After applying the objective cutoff constraint to problem (8), FP is restarted again (from the optimal solution of the continuous relaxation).

Bonami *et al.* [15] proposed a variant of FP for MICPs. In their method, $\bar{x}^k \in \mathcal{C}$ is generated in the same way as the basic FP heuristic, whereas $\hat{x}^k$ is generated by solving a sub-MILP. This sub-MILP is constructed by building an outer approximation (OA) of the nonlinear constraints in $\mathcal{C}$ (including the objective cutoff constraint if an integer solution is available) with linearizations taken at all points of the sequence $\langle \bar{x}^l \rangle_{l=0,\ldots,k}$. $\hat{x}^k$ is taken as the point in the current OA of the constraints that is closest to $\bar{x}^k$ in l_1-norm by solving the

following problem:

$$\begin{aligned}
\min\ & \sum_{j\in\mathcal{I}} |x_j - \bar{x}_j^k| \quad \text{(FP-OA)} \qquad (10)\\
\text{s.t.}\ & g_i(\bar{x}^l) + \nabla g_i(\bar{x}^l)(x - \bar{x}^l) \geq 0,\ i = 1,\ldots,\hat{m} \ \text{ and } l = 0,\ldots,k,\\
& \nabla c(x)(x - x^{k*}) \leq c(x^{k*}) - \epsilon, \ \text{if the (best) integer solution } x^{k*} \text{ is available},\\
& Ax = b,\\
& x_j \in \mathbb{Z} \text{ for each } j \in \mathcal{I},\\
& 0 \leq x \leq u.
\end{aligned}$$

Since (10) is a linear relaxation of the feasible set of $(\mathcal{C})$ combined with the integrality requirements, as a result, if (10) is empty, i.e, no integer solution exists, then FP terminates the search and claim that the available integer solution is optimum within a specific tolerance ϵ, or the MICP is infeasible.

Computational results [15] show that this variant of the FP (called OAFP) heuristic outperforms the basic FP heuristic in terms of finding better solutions. Therefore, we choose OAFP for rounding a point generated from the random walk methods. We note that Bonami *et al.* [15] proposed an enhanced version of OAFP that avoids cycling. However, in our experiments OAFP did not suffer from this issue; hence, our implementation mainly follows the basic version of OAFP with a modification given in the next section.

3.2 Improving the quality of the solutions found by feasibility pump

Although the original FP heuristic has been shown to be effective for finding feasible solutions of MILPs, the quality of heuristic solutions in terms of objective value can potentially be poor. Hence, once a feasible integer solution $\hat{x}^k$ is obtained, we try polishing the quality of $\hat{x}^k$ over the continuous relaxation domain, i.e., we solve (1)–(3) with constraints $x_j = \hat{x}_j^k, j \in \mathcal{I}$. In particular, we solve a convex program of the form:

$$\begin{aligned}
\min\ & c(x) \quad \text{(FP-POL)} \qquad (11)\\
\text{s.t.}\ & x \in \mathcal{C},\\
& x_j = \hat{x}_j^i, \ \text{for each } j \in \mathcal{I}.
\end{aligned}$$

Though additional computational overhead occurs in solving (11), we accept this cost tradeoff over the original OAFP procedure, with the hope to find a better solution and further strengthen the objective cutoff constraint.

Figure 1 illustrates the initial point, rounded point, the generated point, the feasible set, and the outer approximation in the OAFP procedure. The search starts from an initial point $\bar{x}^0$, which is rounded to an integer solution $\hat{x}^0$ by solving problem (10) constructed at $\bar{x}^0$. Since $\hat{x}^0 \notin \mathcal{C}$ and hence is not a feasible solution, we solve problem (8) that projects $\hat{x}^0$ back to $\mathcal{C}$, resulting another point $\bar{x}^1$. Again, we round $\bar{x}^1$ to an integer solution $\hat{x}^1$ by solving problem (10) constructed at $\bar{x}^0$ and $\bar{x}^1$. Since $\hat{x}^1$ is a feasible solution, we polish it by

solving problem (11), yielding another feasible solution $\hat{x}^{1\prime}$. Finally, we update the artificial objective cutoff constraint at $\hat{x}^{1\prime}$, and continue the OAFP search.

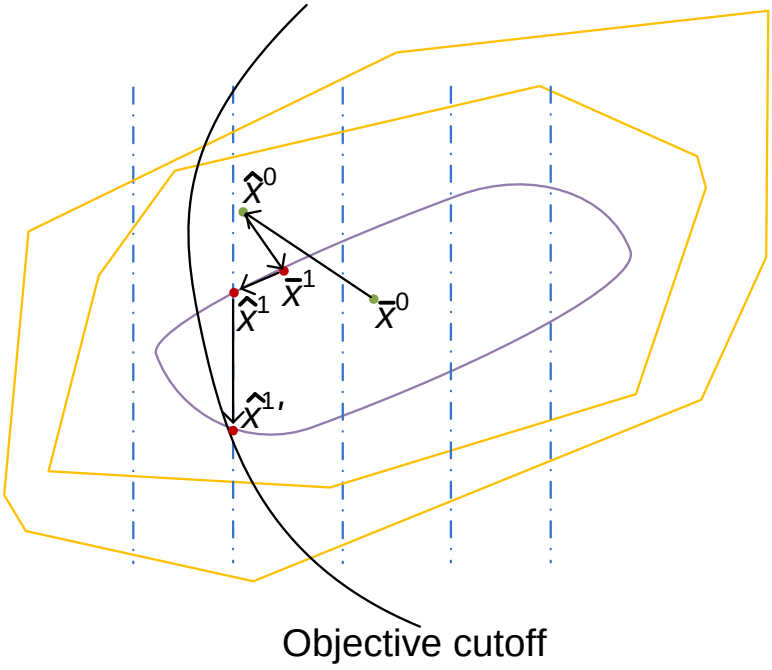


Fig. 1: Illustration of the OAFP search

A flow chart of the implementation of the OAFP heuristic is given in Figure 2. We note that the main difference between our implemented OAFP heuristic and the original OAFP heuristic proposed by Bonami *et al.* [15] is that our method includes solving FP-OA, FP-NLP, and FP-POL, whereas the original one includes only FP-OA and FP-NLP.

4 Implementation

We now present implementation details of the walk-relax-round heuristic. This heuristic is implemented in a software package referred to as `iOptimize`, which is being developed in C++. Details of continuous optimization algorithmic implementation are in Mehrotra and Huang [32], and Huang and Mehrotra [25]. Here we provide relevant information for completeness.

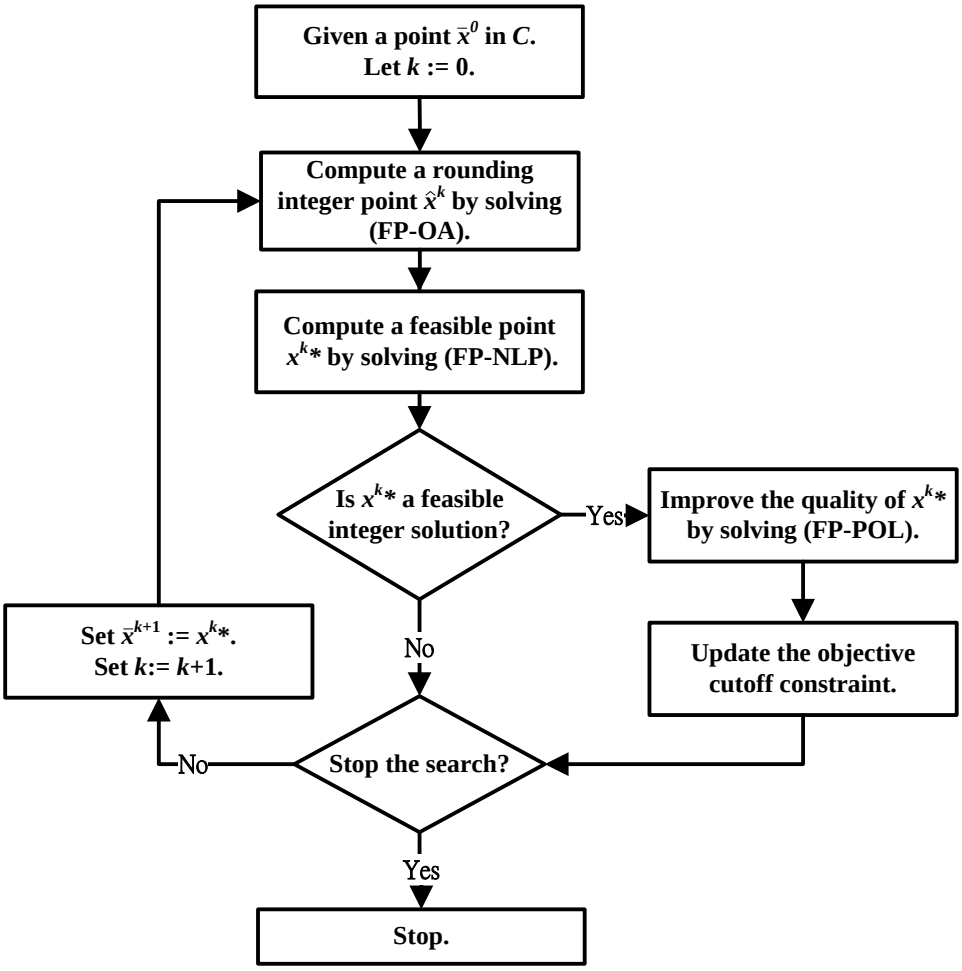


Fig. 2: Implementation of the OAFP heuristic

4.1 Solving convex optimization problems

We used `iOptimize` convex solver with Mehrotra predictor-corrector [31] variant of the homogeneous interior point method [8,9,25,32] to find an optimal solution of the continuous relaxation. The implementation is equipped with a refined potential function that ensures global convergence [25,32].

Let rv_p^k, rv_d^k, and rv_g^k be the residual vectors of the primal feasibility, dual feasibility, and complementarity of the optimality condition corresponding to the kth interior point iterate. We terminate the algorithm if one of the following criteria is satisfied:

– An (approximate) optimal solution is obtained if

$$\frac{\|rv_p^k\|}{\max\left\{1, \|rv_p^0\|\right\}} < 10^{-8}, \tag{12}$$

$$\frac{\|rv_d^k\|}{\max\left\{1, \|rv_d^0\|\right\}} < 10^{-8}, \tag{13}$$

$$\frac{|rv_g^k|}{\max\left\{1, |rv_g^0|\right\}} < 10^{-8}. \tag{14}$$

– The problem is (near) infeasible if

$$\frac{\mu^k}{\mu^0} < 10^{-8}, \quad \text{and} \quad \frac{\tau^k}{\min\left\{1, \kappa^k\right\}} < \frac{\tau^0}{\kappa^0} 10^{-12},$$

where τ and κ represent variables introduced in the homogeneous reformulation [8,9,25], and μ represents the centering parameter used in the context of standard interior point method [8,9,25].

4.1.1 Analytic center computations

Recall in Section 2 we assumed that all the walks are started from near the log-barrier analytic center. The results reported here were obtained by using a two-phase approach in `iOptimize` for computing the analytic center. Phase-I problem was solved as a continuous relaxation optimization problem except with a null-objective function and artificial upper bounds for the primal variables that ensure boundedness of the feasible region. The solution from Phase-I was processed to remove the redundant variables by forcing variables to their bounds or to set the linear inequality constraints to equality, if the slack to the bound is less than 10^{-8}. For the Phase-II analytic center computation, a feasible primal-dual potential-reduction method [37] was used. Newton iterations were used on the perturbed-KKT conditions of the log-barrier problem. Phase-II used $\|Xs - e\|_\infty \leq 10^{-1}$ and the infeasibility criterion in (12)–(13) for termination.

4.2 Implementation of the walk-relax-round heuristic

We now give the implementation logic of the walk-relax-round heuristic. Similar to the observation in Huang and Mehrotra [26], the walk-relax-round heuristic finds feasible solutions quickly in the beginning of the search, and needs more effort for improving the objective value. With this in mind, we find it useful to split the walk-relax-round heuristic in three stages. In the initial stage, we start the OAFP procedure at an optimal solution of the continuous relaxation as it is often a good candidate for generating an integer solution. In the second stage, we perform the random walk from an analytic center to generate sample points, and round them using OAFP. Note that all

the OA constraints built from the previous OAFP searches will not be reused, i.e., we restart the OAFP search from scratch. In the third stage, we perform the search as in the previous stage, but with a different setting in that we reuse all the OA constraints built from the previous OAFP searches at this stage. We update the objective cutoff constraint whenever a better integer solution is found. Consequently, a new (approximate) analytic center of the new continuous relaxation is recomputed, and the random walk is restarted from the new near-center point. In the following we give more details about each of these stages.

- Stage 1: We start the OAFP procedure at an optimal solution of the continuous relaxation. However, to avoid unnecessary computational effort, we perform only 15 OAFP iterations ($I_{\max} = 15$) for the search. In our experience this setting allows us to find at least one integer solution in most test instances, though the quality of these solutions tends to be inferior.
- Stage 2: We start to sample random points and round them using the OAFP procedure. Our goal here is to quickly reduce the integrality gap. However, as the number of OA constraints accumulate, the sub-MILP may become harder to solve. To save the computational effort, while the walk moves from a point to another, the OAFP search is restarted from scratch, i.e., all the OA constraints built from the previous OAFP search are not reused. Another reason to rebuild the OA constraints is that the OA constraints taken at all the random points generated at this stage may be far away from an integer optimal solution, and hence cannot provide a "good" approximation for $\mathcal{C}$. We terminate the OAFP search once the maximum OAFP iterations ($I_{\max} = 10$) is reached, or three integer solutions are found ($F_{\max} = 3$). We end this stage if the number of walk steps allowed is reached ($S_{\max} = 15$), or the relative integrality gap ($G_I < 40\%$) is observed. The relative integrality gap is calculated as

$$G_I = 100\% \times \frac{Obj - RObj}{\max\{1, |RObj|\}},$$

  where $RObj$ is the optimum value of the continuous relaxation of the problem instance, and Obj is the current best objective value.
- Stage 3: We perform the search as in Stage 2 but with different settings. We use $I_{\max} = 15$ and $F_{\max} = 5$ for terminating OAFP as before. Moreover, we reuse all the OA constraints built from OAFP searches at this stage, i.e, the number of OA constraints are accumulated from one OAFP search to the other, with the hope that these OA constraints are capable of providing a reasonably good approximation for $\mathcal{C}$. We terminate the search if the maximum execution time allowed ($T_{\max}$) is reached.

Recall from Section 3.1 that if problem (10) becomes empty, then the FP search should be terminated (since, no mixed integer solution exists). We have either found an optimal integer solution within a specific tolerance ϵ, or the problem is infeasible. Therefore, in addition to the termination criteria

mentioned above, we also terminate the OAFP search once a sub-MILP (FP-OA) becomes infeasible. On the other hand, if a feasible integer solution is generated, then the artificial objective cutoff constraint is applied to $\mathcal{C}$ defined in Section 2.

We note that the parameters for $I_{\max}$ and $F_{\max}$ in each stage are chosen arbitrarily, but they allow us to perform the search in each stage in a reasonable time. Our experience suggests that if we start to accumulate OA constraints right after Stage 1 (that is, we ignore Stage 2), the number of the OA constraints may grow significantly. In this case, the effort for solving FP-OA increases. Hence, care is required in building OA away from an integer optimum.

A flowchart of the implementation of the walk-relax-round heuristic is given in Figure 3.

4.3 Computational environment

All computations were performed on a 3.2 GHz Intel Core 2 Duo CPU with 4GB RAM; however, no parallelization is done in our current implementation. Our problem test set consists of 58 convex MICP instances taken from the `CMU-IBM` Open source MINLP Project (`CMU-IBM` for short) [2]. An `AMPL` [1] interface is implemented in `iOptimize` to read the `AMPL` MICP models and the corresponding first and second-derivatives are calculated using `AMPL`.

`Cplex` 12.1 [5] is used for solving the sub-MILPs (FP-OA). Here we do not insist on solving the sub-MILPs (FP-OA) to optimality. Following Bonami *et al.* [15], we terminate `Cplex` once a feasible integer solution is found, and it has not improved for 5000 nodes. Moreover, we set `CPX_PARAM_MIPEMPHASIS`= 1 that emphasizes feasibility over optimality, and we set parameter `CPX_PARAM_THREADS`= 1 that forces `Cplex` to solve the sub-MILPs sequentially in a single thread.

In our computational runs we set time limit of five minutes for each problem in each run. This time limit is chosen arbitrarily, but it allowed us to perform computations with a reasonable effort. We will, however, discuss results obtained from using long-runs on a few problems. Table 1 gives the instance names, the number of continuous variables (Conti), integer variables (Int), linear constraints (LRows), nonlinear constraints (NLRows), and the corresponding objective value (Obj). All the `CMU-IBM` test problems are mixed binary problems. Note that instance `trimloss12` remains unsolved.

To provide a summary of the results, we use performance profiles [21]. Consider a set A of n_a algorithms, a set P of n_p problems, and a performance measure $m_{a,p}$, e.g., computation time or a objective value. We compare the performance on problem p by algorithm a with the best performance by any algorithm on this problem using the following performance ratio

$$r_{p,a} = \frac{m_{p,a}}{m_p^*},$$

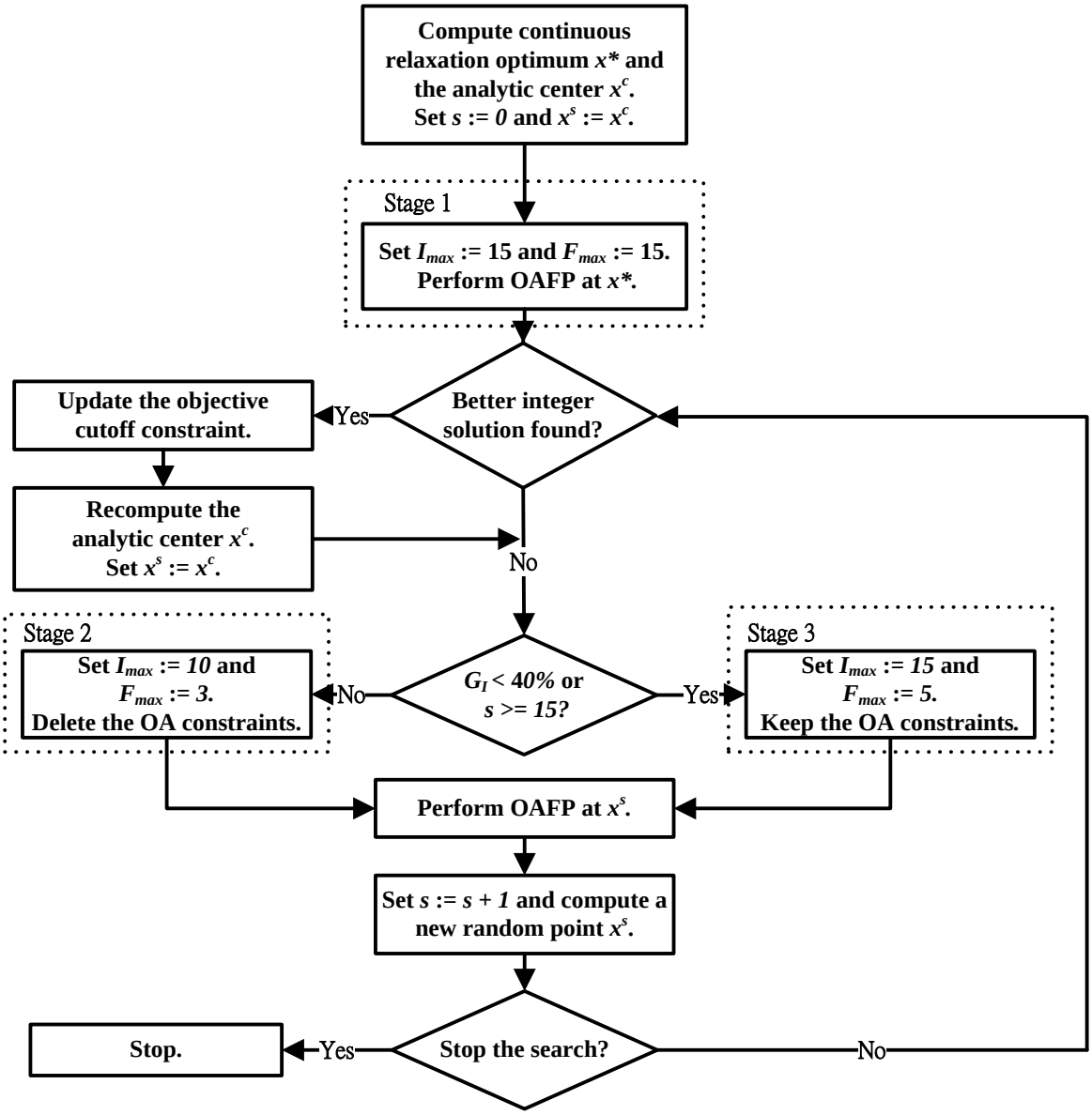


Fig. 3: Implementation of the walk-relax-round heuristic

where $m_p^* := \min\{m_{p,a} \mid a \in A\}$. We therefore obtain an overall assessment of the performance of the heuristic by defining the following value

$$\rho_a(\tau) = \frac{1}{n_p} cardinality\{p \in P \mid r_{p,a} \leq \tau\}.$$

This represents the probability for algorithm a that the performance ratio $r_{p,a}$ is within a factor τ of the best possible ratio. The function $\rho_a(\cdot)$ represents the distribution function for the performance ratio.

5 Computational results

In this section we present and discuss our computational results on the test problems. We first compare the performance of FP at an optimum (FP-OPT) and at the analytic center (FP-AC) of the continuous relaxation. In Section 5.2, we compare the performance of the walk-relax-round heuristic that uses the hit-and-run walk, and Dikin walk with Strategies I and II to generate a random point in two settings: (i) using the "default" seed as an initial seed to generate a walk; (ii) average performance over 30 runs using different initial seeds. In Section 5.3, we discuss additional computational comparisons. In particular, we compare the walk-relax-round heuristic using long step Dikin walk (Strategy II) with FP-OPT, present the performance of different stages of the walk-relax-round heuristic described in Section 4.2, discuss some details about the solution polishing phase, perform a long random walk on selected problems, and finally provide computational comparison with `Cplex` 12.1 on nine mixed integer (convex) quadratic programs (MIQPs).

In the following, we refer the walk-relax-round heuristic using hit-and-run walk as *HR*, and using Dikin walk with Strategies I (short step strategy) and II (long step strategy) as *DW1* and *DW2*.

5.1 Feasibility pump at an optimal solution and the analytic center of the continuous relaxation

An optimal solution of the continuous relaxation is often considered a good candidate for generating an integer solution, so this is an obvious starting choice for FP. However, it is possible that the analytic center may be closer to an integer solution. Table 2 gives the FP performance at an optimal solution (FP-OPT) and at the analytic center (FP-AC) of the continuous relaxation on the `CMU-IBM` test problems. The best available upper bound on the objective value is reported in Column "Obj". The total computational times are reported in Column "Time" in seconds. The optimality "Gap" is calculated as

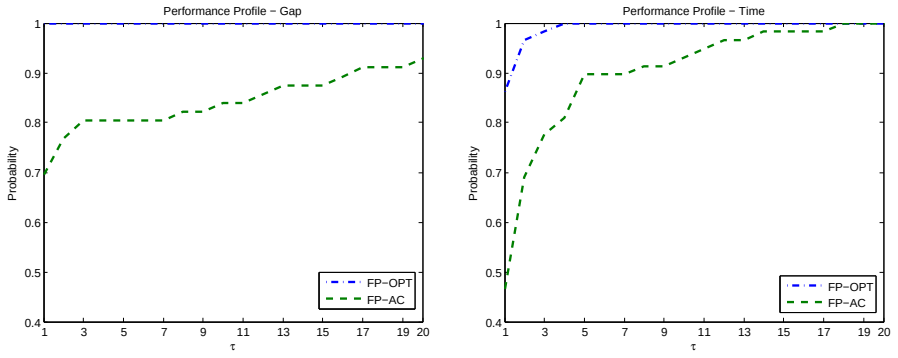
$$\text{Gap} := 100\% \times \frac{Obj - Obj^*}{|Obj^*|},$$

where Obj^* is the optimal value or the best bound of the problem instance. We note that the optimal value or the best bound of all the problem instances is not zero. "Avg" in the last row of Table 2 provides the arithmetic mean for the Gap and geometric mean for the computational times. We note that FP is implemented using OA described in Section 3.

Considering Avg in these experiments, the mean-gap of FP-OPT (Avg = 3.34%) is significantly better than that for FP-AC (Avg = 19.78%). In these experiments FP-OPT proves optimality of the solutions in 39 instances whereas FP-AC proves optimality in 30 instances. Although FP-AC performs worse than FP-OPT in terms of its average performance, it is able to find at least one feasible solution for instance `trimloss7` where FP-OPT fails. For instance `trimloss12`, both heuristics fail to find a feasible solution.

We note that the computational results of the integrality gap performance for instances `SLay08H`, `SLay09M`, `SLay10H`, and `SLay10M` are worse than those reported in Bonami *et al.* [15]. This may come from different continuous solver used for solving the continuous relaxation: we used `iOptimize` [25] HSD interior point method whereas Bonami *et al.* [15] used `Ipopt` [3] primal-dual interior point method.

Figure 4 shows the performance profiles for `CMU-IBM` problems comparing the integrality gap and CPU time performances of FP-OPT and FP-AC. FP-OPT dominates FP-AC in terms of either the integrality gap or the CPU time performance. Hence, FP-AC is not discussed further.



(a) Integrality gap performance (b) CPU time performance

Fig. 4: Performance profiles of FP with different starting solutions

5.2 Computational comparison of random walks

Since the performance of the random walk methods may depend on the initial seed of the random number generator, we now discuss the obtained results in two settings: (i) using the "default" seed as an initial seed to generate a walk;

(ii) average performance over 30 runs using different initial seeds. We note that because of the inherent randomness in the generated "short" walk, the conclusions must be validated with multiple walks. However, in practice only a short walk will be performed.

Recall from Section 4.2 that we split the search of the walk-relax-round into three stages, and start the OAFP procedure at an optimal solution of the continuous relaxation in the first stage. Hence, the first stage of the search in the walk-relax-round is same as FP-OPT.

5.2.1 Performance of single run computations

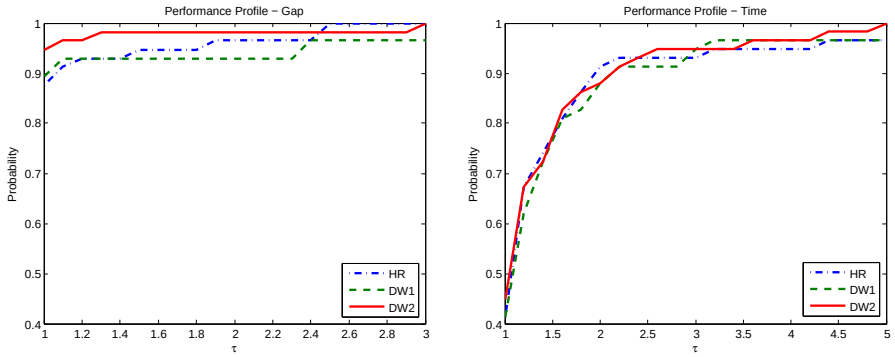
We now report the performance of the random walks on `CMU-IBM` test problems. The computational results are given in Table 3. This table reports the best objective value (Columns "Obj") obtained from HR, DW1, and DW2. Computational time (in seconds) is given under Column "Time", and the optimality "Gap" is calculated as indicated in Section 5.1. "Avg" in the last row of Table 3 provides the arithmetic mean for the Gap and geometric mean for the computational times.

The "Avg" mean-gap in these experiments are 1.26%, 1.35%, and 1.11% for HR, DW1, and DW2, respectively. In this case DW2 ranked first whereas DW1 ranked last. One may expect for DW1 to perform better than HR and DW2 because DW1 started from the center (with lazy step and a self-concordant barrier assumption) mixes in strongly polynomial time [34] while HR mixes in polynomial time [29]. However, the number of walk steps taken in our experiment is small. Moreover, DW2 appears better than HR because HR does not use geometric information from the convex body. Consequently when combined with OAFP, DW2 may be able to explore more region than HR and DW1.

We note that in our experiment the walk-relax-round heuristic may be able to find an integer solution in Stage 1, though the integrality gap of the solution tends to be inferior (more than 100%) for difficult problems. However, the heuristic usually improves the integrality gap of the incumbent solutions to less than 40% quickly in Stage 2. For easy problems, on the other hand, the heuristic may terminate in Stage 1.

Now we focus on the average computational times for various steps of the walk-relax-round heuristic. We observe that FP takes about 85%, the analytic center computation takes about 13%, the initial relaxation problem computation takes less than 1% time on average, and the walk computation takes less than 1% (the detailed results are not reported here). This suggests that when compared to the OAFP heuristic, additional computational effort in walking and computation of the log-barrier analytic center is not substantial.

Figure 5 shows the performance profiles comparing the best integrality gap and the CPU time performances of HR, DW1, and DW2.



(a) Integrality gap (b) CPU time performance

Fig. 5: Performance profiles of different walk-relax-round heuristics

5.2.2 Performance of multiple run computations

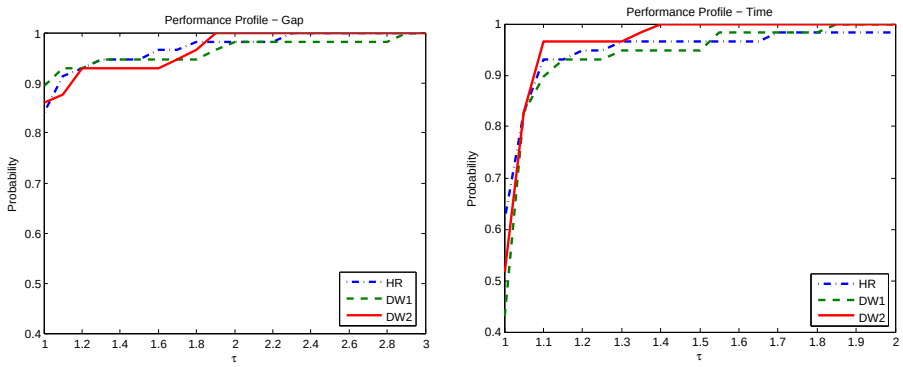
The walk-relax-round heuristic may behave differently with different starting seeds to generate random numbers. Hence, to draw more rigorous conclusions we tested all random walks with 30 different initial random seeds. Table 4 reports the average computational behavior for each problem in the test set. Column "MinGap" shows the minimum gap of a solution among the 30 runs whereas column "MaxGap" shows the maximum gap among a solution of the 30 runs, column "AvgGap" and "StDevGap" shows the arithmetic mean-gap and the corresponding standard deviation of the 30 runs. Column "AvgTime" reports the geometric mean times of the 30 runs.

By comparing the average of "AvgGap", we observe all three methods generate similar results (HR = 1.3%, DW1 = 1.26%, DW2 = 1.26%), though DW1 and DW2 are marginally better than HR. When comparing the average standard deviation, DW2 (0.24) is marginally better than HR (0.3) and DW1 (0.3). A pairwise t-test between their mean-gaps, however, shows that the differences are not statistically significant.

The performance profiles comparing the average integrality gap and the CPU time performances of HR, DW1, and DW2 over multiple runs are given in figure 6.

5.3 Detailed computational results and discussions

We now present additional computational comparisons. We first compare the walk-relax-round heuristic using long step Dikin walk (DW2) with FP-OPT. Next, we illustrate and analyze the performance of different stages of DW2 on two problems, showing the effectiveness of the search in different stage. Then, we discuss some details about the solution polishing phase. In Section 5.3.2, we perform a long random walk using one hour time limit on six problems.



(a) Integrality gap (b) CPU time performance

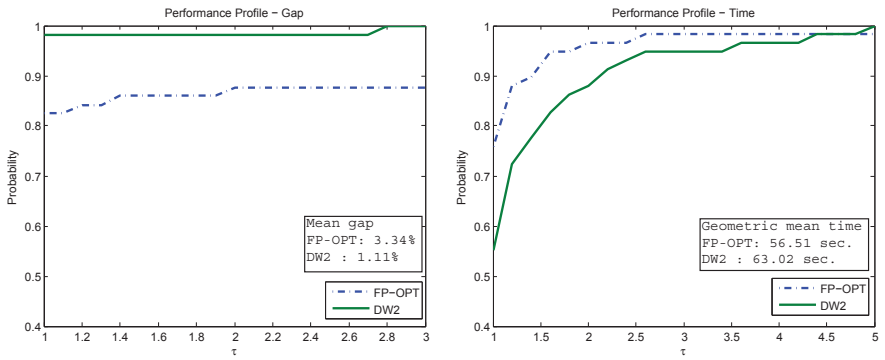
Fig. 6: Performance profiles of different walk-relax-round heuristics over multiple runs

Finally, we provide computational comparison with `Cplex` 12.1 on nine MIQPs selected from the Hans D. Mittelmann's collections [4].

5.3.1 Comparison of walk-relax-round heuristic with FP-OPT

In this section we compare DW2 with FP-OPT using a five minute time limit for both heuristics. The computational results are given in Tables 2 and 3. Observe a single run of DW2 heuristic finds an optimal solution for 51 out of the 58 test problems. In comparison the FP based relax-and-rounding heuristic finds an optimal solution for 45 test problems only. Optimality is proved for 40 problems in case of DW2 and 39 problems in case of FP-OPT. DW2 finds a better upper bound for 6 problems when compared to FP-OPT. Moreover, DW2 can generates a feasible solution for `trimloss7`, but FP-OPT fails. The "Avg" mean-gap for DW2 and FP-OPT are 1.11% and 3.34%, respectively. This suggests that DW2 generates better quality solutions. One possible reason is that DW2 is expected to build a better OA of the convex set than FP-OPT since it likely explores more continuous relaxation feasible region to generate a better OA. FP-OPT finds a better bound only in one instance (problem `RSyn0840M03M`). In this case DW2 and FP-OPT integrality gaps are 8.53% and 2.53%, respectively. We think this is likely because the number of walk steps performed in the given time limit during the search is relatively small (only 12 walk steps), and hence DW2 was not able to build a "good" OA for the convex feasible region $\mathcal{C}$. When the run time limit was increased to an hour both DW2 and FP-OPT solved this problem to optimality.

Although we gave a 5 minute time limit, several easier problems (e.g., the `CLay` and `Syn` families) were solved to optimality in much less time by both DW2 and FP-OPT heuristics. We note that for these problem classes FP-OPT requires less computational time than DW2. Clearly, for these easier problems



(a) Integrality gap performance (b) CPU time performance

Fig. 7: Performance profiles of FP-OPT and DW2

the computational burden from exploring the extra feasible region to build better OA is not justified.

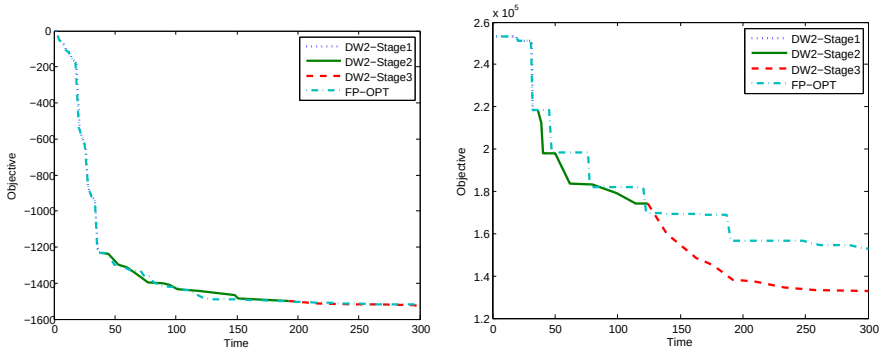
For `SLay` problem family we observe that the objective value gap between the integer feasible solution found by FP-OPT and that of the relaxation problem is significant, and this gap is similar to that of the feasible integer solution obtained from FP-AC. It appears that because of this gap, it is beneficial to build improved OA by performing a random walk in the feasible set for the `SLay` family.

Since the performance of DW2 depends on the randomly generated points, we now compare the performance of DW2 with FP-OPT over 30 runs. For simplicity we use mean integrality gap of FP-OPT with minimum, maximum, and average over 30 runs of the mean integrality gap in DW2. The minimum and the maximum integrality gaps for DW2 are 1.03% and 1.59%, respectively, while the average mean integrality gap over 30 runs is AvgGap = 1.26%. Even in the worst run the mean integrality gap of DW2 is better than that of FP-OPT.

Figure 7 shows the performance profiles comparing the best integrality gap and the CPU time performances of FP-OPT and a single run of DW2. We note that about 70% of the (easy) problems can be solved to optimality by both DW2 and FP-OPT. For the remaining 30% of the (difficult) problems, both methods used 5 minute time limit without solving the problem to optimality.

5.3.2 Performance of different stages of walk-relax-round heuristic on selected problems

Recall that in Section 4.2 we split the walk-relax-round heuristic in three stages. We now present the performance of different stages of DW2. Figure 8 shows the incumbent solution over time of FP-OPT and a single run for DW2 on problems `RSyn0830M03M` and `SLay10`. Performance at different stage is pre-



(a) RSyn0830M03M (b) SLay10

Fig. 8: Incumbent solution over time of FP-OPT and DW2 on selected problems

sented in Figure 8. Note that DW2 performs FP at an optimum of the continuous relaxation in the first stage. Hence, the result of this stage is identical to that of FP-OPT.

For problem `RSyn0830M03M`, both FP-OPT and DW2 improved the objective value of the incumbent solution from -2.03 to -1231.89 at around 40 seconds (the end of first stage of DW2). However, at around 130 seconds (the middle of stage 2 of DW2), the quality of the incumbent solution of DW2 is inferior than that of FP-OPT. Recall that DW2 performs the FP search from scratch in this stage, i.e., OA constraints built from the previous OAFP search are not reused; on the other hand, FP-OPT reuses all the OA constraints built from the beginning of the search. Hence, FP-OPT may benefit from these OA constraints, if they approximate the convex set accurately. Observe that at around 180 seconds, DW2 starts to slightly outperform FP-OPT. In fact, DW2 moves onto the last stage at this point. Overall, the difference between DW2 and FP-OPT in this problem is not significant.

For problem `SLay10`, DW2 dominates FP-OPT over almost the entire search. In fact, FP-OPT generates a slightly better solution at around 70 seconds and 120 seconds, both of which are in Stage 2 of the DW2 search. In the last stage, DW2 outperforms FP-OPT.

5.3.3 Relevance of solution polishing phase

We also note that the technique of improving the solution quality by solving FP-POL mentioned in Section 3.2 helps in finding a better solution, especially in the beginning of the search. For example, for problem `RSyn0830M03M`, the value of the first integer solution found by DW2 (the solution found in the rounding phase by solving FP-OA) is 4269.63. This value is subsequently improved to -2.03 in the solution polishing phase (by solving FP-POL). After 10

steps, DW2 found a solution with a value of -1474.82, and FP-POL improved it to -1479.88. After 15 walk steps, DW2 found a solution with a value of -1514.28, but no further improvement was made.

In general, our experience suggests that FP-POL is able to improve the quality of the solution significantly in the beginning of the search (e.g., the solution found in Stage 1). On the other hand, FP-POL gives limited improvement when the integrality gap of the incumbent solution is small (e.g., the integrality gap is less than 5%).

5.3.4 Performance of a long random walk

All the previous experiments were performed with relatively small computational efforts (at most five minutes). One may perform a long run for the walk-relax-round heuristic, with the hope to keep improving the objective value. We performed such long walks using at most one hour of running time for six difficult problems: `trimloss7`, `trimloss12`, `BatchS201210M`, `RSyn0830M03M`, `RSyn0840M03M`, and `SLay10H`. Table 5 summarizes the results for these problems except for `trimloss12`, for which none of the heuristics found a feasible solution. Here "Start Obj" and "Start Gap" respectively represents the objective value and the gap after 5 minutes (i.e., the results from Tables 2 and 3), whereas "End Obj" and "End Gap" respectively represents the objective value and the gap at the end of the run.

Observe that all methods solved problems `BatchS201210M`, `RSyn0830M03M`, and `RSyn0840M03M` to optimality. For the remaining two problems, DW1 (18.4 (`trimloss7`), 129971.1 (`SLay10H`)) generates equal or better results than DW2 (18.4 (`trimloss7`), 130996.5 (`SLay10H`)) and HR (18.9 (`trimloss7`), 130996.4 (`SLay10H`)). This result is different from the short run computation presented before, where DW2 ranked first. This suggests Dikin walk explores more region uniformly using short walk steps (DW1). This is consistent with the theory of Dikin walk.

For comparison, we also performed FP-OPT with a one hour time limit for these six problems. As the walk-relax-round heuristic, FP-OPT can also solve problems `BatchS201210M`, `RSyn0830M03M`, and `RSyn0840M03M` to optimality, but with less CPU times in most cases. However, for the remaining problems, FP-OPT (18.6 (`trimloss7`), 131351.7 (`SLay10H`)) performed slightly worse than DW1 (18.4 (`trimloss7`), 129971.1 (`SLay10H`)) and DW2 (18.4 (`trimloss7`), 130996.5 (`SLay10H`)).

5.3.5 Comparison of walk-relax-round on Mixed Integer Quadratic Programs

In this section we compare the walk-relax-round heuristic using long step Dikin walk (DW2) with `Cplex` 12.1 for solving MIQPs. We selected nine small and medium MIQPs from the test set at [4] by using the number of rows/columns is less than 5000, and the number of integer variables is less then 1000 as a criterion. Table 6 gives the instance names, the number of continuous variables (Conti), integer variables (Int), linear constraints (LRows), total number of

nonzero elements in the affine matrix (NZeros), and total number of nonzero elements in the quadratic objective matrix (QNZeros).

We used `Cplex` default parameter settings, except for parameter `CPX_PARAM_THREADS` = 1 that forces `Cplex` to solve the problems sequentially using a single thread. A five minute time limit is used as the termination criterion for both implementations. The computational results are given in Table 7. Here "Time (Found)" and "Time (Proven)", respectively, represents the computation times for finding an optimal solution, and that for proving its optimality within the time limit. Columns "Obj" reports the best objective value obtained from DW2 and `Cplex`. The optimality "Gap" is calculated as indicated in Section 5.1.

Observe both DW2 and `Cplex` find an optimal solution for eight out of the nine test problems. `Cplex` also proves optimality for four problems; whereas DW2 does so for three problems. In particular, for problems `isqp0` and `isqp1`, both methods find an optimal solution within 10 seconds. In fact, DW2 finds an optimal solution in Stage 1 of the search, and `Cplex` finds an optimal solution at the root node. For problem `ibell3a`, DW2 takes about 20 seconds to find an optimal solution and about 35 seconds to prove its optimality; on the other hand, `Cplex` needs less 10 seconds to solve this problem to optimality.

For problem `iqiu`, DW2 finds an optimal solution faster than `Cplex`; however, it fails to prove its optimality within the time limit; whereas `Cplex` proves optimality in about 200 seconds. For problem `isqp`, DW2 requires significantly more effort (154.45 seconds) to find an optimal solution than `Cplex` (8.11 seconds). We note that for this problem `Cplex` finds the optimal solution after examining only about 50 nodes of the search tree; whereas DW2 performs more than 15 walk steps before it can find the optimal solution. For problem `itointqor`, DW2 not only finds an optimal solution slightly faster than `Cplex`, but also proves its optimality within the time limit; whereas `Cplex` fails to do so. However, for problem `ilaser0`, DW2 can only find a near optimal solution (with optimality gap less than 0.001). On the other hand, `Cplex` finds an optimal solution in 33 seconds (but fails to prove it to optimality within the time limit). Finally, for problem `iportfolio`, DW2 finds an optimal solution in about one minute and proves its optimality within the time limit; whereas `Cplex` fails to find a feasible solution.

Overall, though DW2 in general takes longer computation times than `Cplex` to find an optimal solution, we think that the results are still promising, as our implementation is not a fine-tuned software like `Cplex`, and it is able to find an optimal solution in one instance, where `Cplex` fails. We also note that our code uses `Cplex` as a black box, and does not take advantage of any of its internal features.

6 Concluding Remark

FP [15] is a useful heuristic for finding good quality solutions of mixed integer convex programs in a reasonable computational time. Conventionally,

an optimal solution of a continuous relaxation is used as a starting point for FP. First, our computational results show that, in general, the performance of FP started from an optimal solution of the continuous relaxation is a better than if it is started from the analytic center. This conclusion is consistent with that of Baena and Castro [10], and Huang and Mehrotra [26] in the context of mixed integer linear programs. However, for harder problems we found that the incorporation of a walk in FP allows us to generate better quality solutions with the same time limit. We conjectures that it is likely because solutions generated from the walk points allow us to build a better outer approximation of the convex region. Our computational results suggest that the long step Dikin walk is marginally better than the hit-and-run walk, and the walks using short steps. On the other hand, the short step Dikin walk generates a better quality solution when many walk steps are needed. This is consistent with the theory of Dikin walk.

We note that Huang and Mehrotra [26] experimented with a random ray strategy to sample feasible points from a polyhedral set. This method generates random points by *shooting* random rays from near the analytic center. More precisely, it performs the steps of Algorithm 2.2 with Strategy I repeatedly without updating the solution, i.e., we set $x^{k+1} := x^0 + \lambda p$ in Strategy I of Algorithm 2.2. Huang and Mehrotra [26] found that the results of this random ray search approach are generally inferior than those of the random walk approaches. In our experiments (detailed results not reported here) this conclusion also holds for MICPs.

We also note that Achterberg and Berthold [7] proposed a variant of the FP approach, called Objective FP. This variant considers the objective function while solving the projection problem. In particular, they proposed the use of a convex combination of the original objective function with the l_1-norm function. The hope is that Objective FP not only converges to a feasible solution but also concentrates the search in the region of high quality points. The principle of Objective FP can be extended to the context of MICP. However, finding a proper balance between the original objective function and the l_1-norm function remains unclear, and is a topic of future research.

Recently, Naoum-Sawaya [33] proposed a central rounding heuristic that recursively fixes a subset of integer variables while using the analytic center to re-center the remaining ones. Computational results show that this method provides good quality feasible solutions with a reasonable computational effort. Incorporating this idea to the walk-relax-round heuristic is also a possible future research topic. In addition to the FP heuristic, the dive and search heuristic [16] has also been studied in the context of MICP. Other heuristics that are originally designed for MILPs such as `OCTANE` heuristic [11], local branching heuristic [23], and Relaxation Induced Neighborhood Search (`RINS`) heuristic [20] may also be extended to the context of MICP. Combining random walks with these heuristics is also a topic of further investigation.

Problem	Conti	Int	LRows	NLRows	Objective
trimloss2	6	31	22	2	5.3
trimloss4	20	85	60	4	8.3
trimloss5	30	131	85	5	10.3
trimloss6	56	289	147	7	15.3
trimloss7	56	289	147	7	15.5
trimloss12	156	644	360	12	221.7†
BatchS101006M	149	129	1018	1	769440
BatchS121208M	242	203	1780	1	1543472
BatchS151208M	307	251	2326	1	2295349
BatchS201210M	20	36	58	48	40262.4
CLay0304H	152	24	210	48	40262.4
CLay0304M	20	36	58	48	40262.4
CLay0305H	235	40	335	60	8092.5
CLay0305M	30	55	95	60	8092.5
FLay05H	342	40	460	5	64.5
FLay05M	22	40	60	5	64.5
FLay06H	506	60	687	6	66.93
FLay06M	26	60	87	6	66.93
fo7	72	42	197	14	20.73
fo7_2	72	42	197	14	17.75
fo8	90	56	257	16	22.38
fo9	110	72	325	18	23.46
o7	72	42	197	14	131.64
o7_2	72	42	197	14	116.94
RSyn0830M	156	94	405	20	-510.07
RSyn0830M02H	962	210	1754	40	-730.51
RSyn0830M02M	372	248	1232	40	-730.51
RSyn0830M03H	1443	315	2874	60	-1543.06
RSyn0830M03M	558	372	2091	60	-1543.06
RSyn0840M	176	104	456	28	-325.55
RSyn0840M02H	1124	236	2050	56	-734.98
RSyn0840M02M	432	288	1424	56	-734.98
RSyn0840M03H	1686	354	3363	84	-2742.65
RSyn0840M03M	648	432	2424	84	-2742.65
SLay07H	392	84	609	0	64749
SLay07M	56	84	189	0	64749
SLay08H	520	112	812	0	84960
SLay08M	72	112	252	0	84960
SLay09H	666	144	1044	0	107806
SLay09M	90	144	324	0	107806
SLay10H	830	180	1305	0	129580
SLay10M	110	180	405	0	129580
Syn30H	217	11	325	20	-138.16
Syn30M	70	30	147	20	-138.16
Syn30M02H	494	82	860	40	-399.68
Syn30M02M	200	120	564	40	-399.68
Syn30M03H	741	123	1425	60	-654.15
Syn30M03M	300	180	981	60	-654.15
Syn30M04H	988	164	2080	80	-865.72
Syn30M04M	400	240	1488	80	-865.72
Syn40H	288	14	438	28	-67.71
Syn40M	90	40	198	28	-67.71
Syn40M02H	656	108	1156	56	-388.77
Syn40M02M	260	160	756	56	-388.77
Syn40M03H	984	162	1914	84	-395.15
Syn40M03M	390	240	1314	84	-395.15
Syn40M04H	1312	216	2792	112	-901.75
Syn40M04M	520	320	1992	112	-901.75

[1] † : previous best known objective value.

Table 1: Problem profiles of `CMU-IBM` library

Method	FP-OPT			FP-AC		
Problem	Obj	Gap	Time	Obj	Gap	Time
trimloss2	5.3*	0	0.33	5.3*	0	0.28
trimloss4	8.3*	0	173.69	9.6	15.66	128.51
trimloss5	11.3	9.71	‡	12.5	21.36	‡
trimloss6	15.6	1.96	‡	16.1	5.23	‡
trimloss7	–	NA	‡	38.7	149.68	‡
trimloss12	–	NA	‡	–	NA	‡
BatchS101006M	769440*	0	86.06	880612.7	14.45	‡
BatchS121208M	1241125*	0	227.18	3977749	220.5	‡
BatchS151208M	1543472	0	‡	4088185	164.87	‡
BatchS201210M	2324008	1.25	‡	7480235	225.89	‡
CLay0304H	40262.4*	0	10.81	40262.4*	0	9.27
CLay0304M	40262.4*	0	5.8	40262.4*	0	8.31
CLay0305H	8092.5*	0	26.75	8092.5*	0	40.37
CLay0305M	8092.5*	0	47.95	8092.5*	0	14.34
FLay05H	64.5*	0	87.2	64.5*	0	93.75
FLay05M	64.5*	0	21.01	64.5*	0	14.77
FLay06H	66.93	0	‡	66.93	0	‡
FLay06M	66.93	0	‡	66.93	0	‡
fo7	20.73*	0	110.98	20.73*	0	130.2
fo7_2	17.75*	0	63.1	17.75*	0	30.34
fo8	22.38*	0	230.86	22.38	0	‡
fo9	23.46	0	‡	23.46	0	‡
o7	131.64	0	‡	131.64	0	‡
o7_2	116.94	0	‡	116.94	0	‡
RSyn0830M	-510.07*	0	11.23	-510.07*	0	20.7
RSyn0830M02H	-730.51*	0	53.01	-730.51*	0	248.81
RSyn0830M02M	-730.51*	0	145.5	-730.51*	0	172.33
RSyn0830M03H	-1543.06*	0	200.91	-1256.44	18.57	‡
RSyn0830M03M	-1520.02	1.49	‡	-1507.88	2.28	‡
RSyn0840M	-325.55*	0	16.55	-325.55*	0	48.48
RSyn0840M02H	-734.98*	0	68.31	-659.49	10.27	‡
RSyn0840M02M	-718.31	2.27	‡	-571.5	22.24	‡
RSyn0840M03H	-2741.99*	0	68.03	-1877.13	31.54	‡
RSyn0840M03M	-2673.3	2.53	‡	-1584.38	42.23	‡
SLay07H	64749*	0	33.45	64749*	0	33.43
SLay07M	64749*	0	49.12	64749*	0	49.08
SLay08H	84960*	0	74.63	84960*	0	79.44
SLay08M	102755.7	20.95	‡	102755.7	20.95	‡
SLay09H	108781.1	0.9	‡	109651.9	1.71	‡
SLay09M	130353.5	20.91	‡	130353.5	20.91	‡
SLay10H	152751.8	17.88	‡	152751.8	17.88	‡
SLay10M	268114.8	106.91	‡	268385.8	107.12	‡
Syn30H	-138.16*	0	1.12	-138.16*	0	5.36
Syn30M	-138.16*	0	1.01	-138.16*	0	2.95
Syn30M02H	-399.68*	0	4.4	-399.68*	0	60.38
Syn30M02M	-399.68*	0	9	-399.68*	0	15.76
Syn30M03H	-654.15*	0	9.67	-654.15*	0	113.42
Syn30M03M	-654.15*	0	21.14	-654.15*	0	60.52
Syn30M04H	-865.72*	0	32.37	-861.72	0.46	‡
Syn30M04M	-865.72*	0	49.73	-865.72*	0	176.1
Syn40H	-67.71*	0	1.76	-67.71*	0	18.64
Syn40M	-67.71*	0	2.89	-67.71*	0	4.47
Syn40M02H	-388.77*	0	6.35	-388.77*	0	110.26
Syn40M02M	-388.77*	0	17.86	-388.77*	0	44.12
Syn40M03H	-395.15*	0	42.59	-395.15*	0	210.21
Syn40M03M	-395.15*	0	66.32	-395.15*	0	117.42
Syn40M04H	-901.75*	0	39.95	-825.19	8.49	‡
Syn40M04M	-901.75*	0	87.61	-901.75*	0	212.65
Avg		3.34	56.53		19.78	96.73

[1] NA: Statistics not available.
[2] *: Proven optimality.
[3] –: No solution found.
[4] ‡: 5-minute time limit reached.

Table 2: FP performance on `CMU-IBM` library

Method	HR			DW1		
Problem	Obj	Gap	Time	Obj	Gap	Time
trimloss2	5.3*	0	0.33	5.3*	0	0.36
trimloss4	8.3*	0	298.77	8.3	0	‡
trimloss5	10.3	0	‡	10.7	3.88	‡
trimloss6	15.3	0	‡	15.5	1.31	‡
trimloss7	23.2	49.68	‡	23	48.39	‡
trimloss12	–	NA	‡	–	NA	‡
BatchS101006M	769440*	0	57.94	769440*	0	64.11
BatchS121208M	1241125*	0	143.05	1241125*	0	176.22
BatchS151208M	1565979	1.46	‡	1543472	0	‡
BatchS201210M	2302924	0.33	‡	2345035	2.16	‡
CLay0304H	40262.4*	0	94.19	40262.4*	0	85.14
CLay0304M	40262.4*	0	33.94	40262.4*	0	30.15
CLay0305H	8092.5*	0	38.61	8092.5*	0	37.52
CLay0305M	8092.5*	0	96.03	8092.5*	0	93.79
FLay05H	64.5*	0	86.92	64.5*	0	87.22
FLay05M	64.5*	0	92.34	64.5*	0	60.26
FLay06H	66.93	0	‡	66.93	0	‡
FLay06M	66.93	0	‡	66.93	0	‡
fo7	20.73*	0	111.44	20.73*	0	110.84
fo7_2	17.75*	0	47.59	17.75*	0	43.74
fo8	22.38*	0	143.85	22.38*	0	200.3
fo9	23.46	0	‡	23.46	0	‡
o7	131.64	0	‡	131.64	0	‡
o7_2	116.94	0	‡	116.94	0	‡
RSyn0830M	-510.07*	0	21.83	-510.07*	0	23.07
RSyn0830M02H	-730.51*	0	65.94	-730.51*	0	64.5
RSyn0830M02M	-730.51*	0	167.3	-730.51*	0	176.01
RSyn0830M03H	-1543.06*	0	233.42	-1543.06*	0	290.2
RSyn0830M03M	-1498.04	2.92	‡	-1523.37	1.28	‡
RSyn0840M	-325.55*	0	33.23	-325.55*	0	33.51
RSyn0840M02H	-734.98*	0	73.38	-734.98*	0	72.8
RSyn0840M02M	-734.98	0	‡	-734.98	0	‡
RSyn0840M03H	-2741.99*	0	67.93	-2741.99*	0	68.09
RSyn0840M03M	-2486.93	9.32	‡	-2513.14	8.37	‡
SLay07H	64749*	0	37.83	64749*	0	35.71
SLay07M	64749*	0	19.27	64749*	0	20.23
SLay08H	84960*	0	80.04	84960*	0	89.84
SLay08M	84960*	0	43.84	84960*	0	33.84
SLay09H	107806	0	‡	107806*	0	202.03
SLay09M	107805.7	0	‡	107806	0	‡
SLay10H	139824.1	7.91	‡	144585	11.58	‡
SLay10M	130174	0.46	‡	129651.8	0.06	‡
Syn30H	-138.16*	0	1.11	-138.16*	0	1.11
Syn30M	-138.16*	0	1.03	-138.16*	0	1.03
Syn30M02H	-399.68*	0	4.37	-399.68*	0	4.35
Syn30M02M	-399.68*	0	10.04	-399.68*	0	10.09
Syn30M03H	-654.15*	0	9.66	-654.15*	0	9.66
Syn30M03M	-654.15*	0	35.54	-654.15*	0	38.63
Syn30M04H	-865.72*	0	32.34	-865.72*	0	32.34
Syn30M04M	-865.72*	0	81.42	-865.72*	0	98.73
Syn40H	-67.71*	0	1.76	-67.71*	0	1.76
Syn40M	-67.71*	0	3.77	-67.71*	0	3.88
Syn40M02H	-388.77*	0	6.33	-388.77*	0	6.3
Syn40M02M	-388.77*	0	35.69	-388.77*	0	51.43
Syn40M03H	-395.15*	0	52.13	-395.15*	0	51.81
Syn40M03M	-395.15*	0	99.01	-395.15*	0	100.31
Syn40M04H	-901.75*	0	39.81	-901.75*	0	39.89
Syn40M04M	-901.75*	0	172.5	-901.75*	0	167.25
Avg		1.26	64.93		1.35	65.32

[1] NA: Statistics not available.
[2] *: Proven optimality.
[3] –: No solution found.
[4] ‡: 5-minute time limit reached.

Method	DW2			FP-OPT		
Problem	Obj	Gap	Time	Obj	Gap	Time
trimloss2	5.3*	0	0.34	5.3*	0	0.33
trimloss4	8.3	0	‡	8.3*	0	173.69
trimloss5	10.3	0	‡	11.3	9.71	‡
trimloss6	15.6	1.96	‡	15.6	1.96	‡
trimloss7	23	48.39	‡	–	NA	‡
trimloss12	–	NA	‡	–	NA	‡
BatchS101006M	769440*	0	54.48	769440*	0	86.06
BatchS121208M	1241125*	0	123.68	1241125*	0	227.18
BatchS151208M	1543472	0	‡	1543472	0	‡
BatchS201210M	2310423	0.66	‡	2324008	1.25	‡
CLay0304H	40262.4*	0	46.28	40262.4*	0	10.81
CLay0304M	40262.4*	0	28.22	40262.4*	0	5.8
CLay0305H	8092.5*	0	37.95	8092.5*	0	26.75
CLay0305M	8092.5*	0	30.06	8092.5*	0	47.95
FLay05H	64.5*	0	87.53	64.5*	0	87.2
FLay05M	64.5*	0	73.71	64.5*	0	21.01
FLay06H	66.93	0	‡	66.93	0	‡
FLay06M	66.93	0	‡	66.93	0	‡
fo7	20.73*	0	111.02	20.73*	0	110.98
fo7_2	17.75*	0	45.71	17.75*	0	63.1
fo8	22.38*	0	208.91	22.38*	0	230.86
fo9	23.46	0	‡	23.46	0	‡
o7	131.64	0	‡	131.64	0	‡
o7_2	116.94	0	‡	116.94	0	‡
RSyn0830M	-510.07*	0	22.59	-510.07*	0	11.23
RSyn0830M02H	-730.51*	0	74.27	-730.51*	0	53.01
RSyn0830M02M	-730.51*	0	141.46	-730.51*	0	145.5
RSyn0830M03H	-1543.06*	0	279.16	-1543.06*	0	200.91
RSyn0830M03M	-1526.02	1.1	‡	-1520.02	1.49	‡
RSyn0840M	-325.55*	0	39.84	-325.55*	0	16.55
RSyn0840M02H	-734.98*	0	72.55	-734.98*	0	68.31
RSyn0840M02M	-734.98	0	‡	-718.31	2.27	‡
RSyn0840M03H	-2741.99*	0	68.11	-2741.99*	0	68.03
RSyn0840M03M	-2507.26	8.58	‡	-2673.3	2.53	‡
SLay07H	64749*	0	38.66	64749*	0	33.45
SLay07M	64749*	0	19.34	64749*	0	49.12
SLay08H	84960*	0	83.6	84960*	0	74.63
SLay08M	84960*	0	47.8	102755.7	20.95	‡
SLay09H	107806	0	‡	108781.1	0.9	‡
SLay09M	107806*	0	190.77	130353.5	20.91	‡
SLay10H	132926.2	2.58	‡	152751.8	17.88	‡
SLay10M	129580	0	‡	268114.8	106.91	‡
Syn30H	-138.16*	0	1.11	-138.16*	0	1.12
Syn30M	-138.16*	0	1.01	-138.16*	0	1.01
Syn30M02H	-399.68*	0	4.34	-399.68*	0	4.4
Syn30M02M	-399.68*	0	10.39	-399.68*	0	9
Syn30M03H	-654.15*	0	9.64	-654.15*	0	9.67
Syn30M03M	-654.15*	0	39.52	-654.15*	0	21.14
Syn30M04H	-865.72*	0	32.42	-865.72*	0	32.37
Syn30M04M	-865.72*	0	87.83	-865.72*	0	49.73
Syn40H	-67.71*	0	1.78	-67.71*	0	1.76
Syn40M	-67.71*	0	3.79	-67.71*	0	2.89
Syn40M02H	-388.77*	0	6.33	-388.77*	0	6.35
Syn40M02M	-388.77*	0	39.76	-388.77*	0	17.86
Syn40M03H	-395.15*	0	52.07	-395.15*	0	42.59
Syn40M03M	-395.15*	0	96.58	-395.15*	0	66.32
Syn40M04H	-901.75*	0	39.79	-901.75*	0	39.95
Syn40M04M	-901.75*	0	186.73	-901.75*	0	87.61
Avg		1.11	63.02		3.34	56.53

[1] NA: Statistics not available.
[2] *: Proven optimality.
[3] –: No solution found.
[4] ‡: 5-minute time limit reached.

Table 3: Random walk performance on `CMU-IBM` library

Method	HR				
Problem	MinGap	AvgGap	MaxGap	StDevGap	AvgTime
trimloss2	0	0	0	0	0.34
trimloss4	0	0	0	0	299.75
trimloss5	0.00	4.85	10.68	5.45	‡
trimloss6	0.00	0	0	0	‡
trimloss7	48.39	49.29	50.32	0.87	‡
trimloss12	NA	NA	NA	NA	‡
BatchS101006M	0	0	0	0	68.58
BatchS121208M	0	0	0	0	233.15
BatchS151208M	0	0.47	1.46	0.67	‡
BatchS201210M	0.33	1.88	2.61	0.89	‡
CLay0304H	0	0	0	0	95.88
CLay0304M	0	0	0	0	32.3
CLay0305H	0	0	0	0	38.82
CLay0305M	0	0	0	0	105.41
FLay05H	0	0	0	0	87.58
FLay05M	0	0	0	0	82.84
FLay06H	0	0	0	0	‡
FLay06M	0	0	0	0	‡
fo7	0	0	0	0	112.09
fo7_2	0	0	0	0	39.72
fo8	0	0	0	0	147.88
fo9	0	0	0	0	‡
o7	0	0	0	0	‡
o7_2	0	0	0	0	‡
RSyn0830M	0	0	0	0	24.8
RSyn0830M02H	0	0	0	0	64.79
RSyn0830M02M	0	0	0	0	156.43
RSyn0830M03H	0	0	0	0	258.07
RSyn0830M03M	1.4	2.3	3.51	0.88	‡
RSyn0840M	0	0	0	0	31.7
RSyn0840M02H	0	0	0	0	73.06
RSyn0840M02M	0	0.06	0.32	0.14	‡
RSyn0840M03H	0	0	0	0	71.99
RSyn0840M03M	7.08	8.91	11.2	1.8	‡
SLay07H	0	0	0	0	39.17
SLay07M	0	0	0	0	19.35
SLay08H	0	0	0	0	73.6
SLay08M	0	0	0	0	37.49
SLay09H	0	0	0	0	292.66
SLay09M	0	0.06	0.3	0.13	244.73
SLay10H	1.23	6.27	15.67	5.86	‡
SLay10M	0	0.18	0.46	0.25	‡
Syn30H	0	0	0	0	1.11
Syn30M	0	0	0	0	1.03
Syn30M02H	0	0	0	0	4.36
Syn30M02M	0	0	0	0	10.43
Syn30M03H	0	0	0	0	9.79
Syn30M03M	0	0	0	0	36.92
Syn30M04H	0	0	0	0	32.42
Syn30M04M	0	0	0	0	92.61
Syn40H	0	0	0	0	1.76
Syn40M	0	0	0	0	3.67
Syn40M02H	0	0	0	0	6.33
Syn40M02M	0	0	0	0	33.57
Syn40M03H	0	0	0	0	52.18
Syn40M03M	0	0	0	0	105.35
Syn40M04H	0	0	0	0	39.92
Syn40M04M	0	0	0	0	171.28
Avg	1.03	1.3	1.69	0.3	65.38

[1] NA: Statistics not available.
[2] ‡: 5-minute time limit reached.

Method	DW1				
Problem	MinGap	AvgGap	MaxGap	StDevGap	AvgTime
trimloss2	0	0	0	0	0.37
trimloss4	0	0	0	0	‡
trimloss5	0.00	1.55	3.88	1.76	‡
trimloss6	0.00	1.83	5.23	1.98	‡
trimloss7	48.39	49.55	53.55	2.25	‡
trimloss12	NA	NA	NA	NA	‡
BatchS101006M	0	0	0	0	74.34
BatchS121208M	0	0	0	0	234.03
BatchS151208M	0	0.19	0.92	0.41	‡
BatchS201210M	2.11	2.42	2.92	0.33	‡
CLay0304H	0	0	0	0	85.47
CLay0304M	0	0	0	0	30.44
CLay0305H	0	0	0	0	37.5
CLay0305M	0	0	0	0	62.78
FLay05H	0	0	0	0	87.98
FLay05M	0	0	0	0	69
FLay06H	0	0	0	0	‡
FLay06M	0	0	0	0	‡
fo7	0	0	0	0	112.45
fo7_2	0	0	0	0	37.32
fo8	0	0	0	0	151.55
fo9	0	0	0	0	‡
o7	0	0	0	0	‡
o7_2	0	0	0	0	‡
RSyn0830M	0	0	0	0	25.29
RSyn0830M02H	0	0	0	0	64.55
RSyn0830M02M	0	0	0	0	144.37
RSyn0830M03H	0	0	0	0	258.35
RSyn0830M03M	0.59	1.13	1.56	0.48	‡
RSyn0840M	0	0	0	0	33.88
RSyn0840M02H	0	0	0	0	73.3
RSyn0840M02M	0	0	0	0	‡
RSyn0840M03H	0	0	0	0	68.29
RSyn0840M03M	0	7.33	10.09	4.16	‡
SLay07H	0	0	0	0	37.71
SLay07M	0	0	0	0	17.84
SLay08H	0	0	0	0	84.61
SLay08M	0	0	0	0	42.01
SLay09H	0	0	0	0	280.41
SLay09M	0	0.06	0.3	0.13	261.3
SLay10H	2.06	6.71	11.58	3.61	‡
SLay10M	0.06	1.21	4.52	1.86	‡
Syn30H	0	0	0	0	1.11
Syn30M	0	0	0	0	1.03
Syn30M02H	0	0	0	0	4.36
Syn30M02M	0	0	0	0	10.5
Syn30M03H	0	0	0	0	9.67
Syn30M03M	0	0	0	0	35.29
Syn30M04H	0	0	0	0	32.46
Syn30M04M	0	0	0	0	94.25
Syn40H	0	0	0	0	1.77
Syn40M	0	0	0	0	3.76
Syn40M02H	0	0	0	0	6.33
Syn40M02M	0	0	0	0	51.22
Syn40M03H	0	0	0	0	51.97
Syn40M03M	0	0	0	0	103.76
Syn40M04H	0	0	0	0	40.01
Syn40M04M	0	0	0	0	171.89
Avg	0.93	1.26	1.66	0.3	65.11

[1] NA: Statistics not available.
[2] ‡: 5-minute time limit reached.

Method	DW2				
Problem	MinGap	AvgGap	MaxGap	StDevGap	AvgTime
trimloss2	0	0	0	0	0.35
trimloss4	0	0	0	0	‡
trimloss5	0.00	3.69	9.71	4.03	‡
trimloss6	0.00	0.78	1.96	1.07	‡
trimloss7	48.39	49.16	49.68	0.54	‡
trimloss12	NA	NA	NA	NA	‡
BatchS101006M	0	0	0	0	57.79
BatchS121208M	0	0	0	0	246.93
BatchS151208M	0	0.37	1.11	0.49	293.76
BatchS201210M	0.66	1.74	2.5	0.69	‡
CLay0304H	0	0	0	0	56.59
CLay0304M	0	0	0	0	29.46
CLay0305H	0	0	0	0	38.65
CLay0305M	0	0	0	0	34.23
FLay05H	0	0	0	0	87.99
FLay05M	0	0	0	0	65.48
FLay06H	0	0	0	0	‡
FLay06M	0	0	0	0	‡
fo7	0	0	0	0	112.19
fo7_2	0	0	0	0	38.47
fo8	0	0	0	0	156.75
fo9	0	0	0	0	‡
o7	0	0	0	0	‡
o7_2	0	0	0	0	‡
RSyn0830M	0	0	0	0	25.57
RSyn0830M02H	0	0	0	0	67.81
RSyn0830M02M	0	0	0	0	142.48
RSyn0830M03H	0	0	0	0	270.28
RSyn0830M03M	1.1	2.47	3.19	0.98	‡
RSyn0840M	0	0	0	0	34.26
RSyn0840M02H	0	0	0	0	73.41
RSyn0840M02M	0	0.1	0.5	0.22	‡
RSyn0840M03H	0	0	0	0	72.81
RSyn0840M03M	6.3	8.65	10.88	1.92	‡
SLay07H	0	0	0	0	39.01
SLay07M	0	0	0	0	18.1
SLay08H	0	0	0	0	76.11
SLay08M	0	0	0	0	48.93
SLay09H	0	0	0	0	294.15
SLay09M	0	0	0	0	240.26
SLay10H	2.06	3.6	5.96	1.61	‡
SLay10M	0	1.22	5.08	2.18	‡
Syn30H	0	0	0	0	1.11
Syn30M	0	0	0	0	1.03
Syn30M02H	0	0	0	0	4.36
Syn30M02M	0	0	0	0	10.22
Syn30M03H	0	0	0	0	9.66
Syn30M03M	0	0	0	0	37.45
Syn30M04H	0	0	0	0	32.46
Syn30M04M	0	0	0	0	90.28
Syn40H	0	0	0	0	1.78
Syn40M	0	0	0	0	3.69
Syn40M02H	0	0	0	0	6.35
Syn40M02M	0	0	0	0	45.4
Syn40M03H	0	0	0	0	52.27
Syn40M03M	0	0	0	0	103.59
Syn40M04H	0	0	0	0	39.98
Syn40M04M	0	0	0	0	181.84
Avg	1.03	1.26	1.59	0.24	63.85

[1] NA: Statistics not available.
[2] ‡: 5-minute time limit reached.

Table 4: 30 Runs of random walk performance on `CMU-IBM` library

Method	HR				
Problem	Start Obj	Start Gap	End Obj	End Gap	Time
Trimolss7	23.2	49.68	18.9	21.94	‡
BatchS201210M	2302924	0.33	2295349*	0	1618.83
RSyn0830M03M	-1498.04	2.92	-1543.06*	0	1503.2
RSyn0840M03M	-2486.93	9.32	-2742.65*	0	1427.04
SLay10H	139824.1	7.91	130969.4	1.07	‡

Method	DW1				
Problem	Start Obj	Start Gap	End Obj	End Gap	Time
Trimolss7	23	48.39	18.4	18.71	‡
BatchS201210M	2345035	2.16	2295349*	0	1377.54
RSyn0830M03M	-1523.37	1.28	-1543.06*	0	985.91
RSyn0840M03M	-2513.14	8.37	-2742.65*	0	1440.42
SLay10H	144585	11.58	129971.1	0.3	‡

Method	DW2				
Problem	Start Obj	Start Gap	End Obj	End Gap	Time
Trimolss7	23	48.39	18.4	18.71	‡
BatchS201210M	2310423	0.66	2295349*	0	1820.36
RSyn0830M03M	-1526.02	1.1	-1543.06*	0	904.76
RSyn0840M03M	-2507.26	8.58	-2742.65*	0	1386.98
SLay10H	132926.2	2.58	130996.5	1.09	‡

Method	FP-OPT				
Problem	Start Obj	Start Gap	End Obj	End Gap	Time
Trimolss7	–	NA	18.6	20	‡
BatchS201210M	2324008	1.25	2295349*	0	1001.36
RSyn0830M03M	-1520.02	1.49	-1543.06*	0	1084.5
RSyn0840M03M	-2673.3	2.53	-2742.65*	0	1133.8
SLay10H	152751.8	17.88	131351.7	1.37	‡

[1] *: Proven optimality.
[2] ‡: One hour time limit reached.

Table 5: Numerical experiments of the long run test

Problem	Conti	Int	LRows	NZeros	QNZeros
ibell3a	62	60	104	390	59
ibienst1	477	28	576	2185	27
ilaser0	850	151	1000	3000	3231
iportfolio	233	967	202	202200	200
iqiu	792	48	1192	3744	47
iqsp	950	50	250	3552	17801
isqp0	950	50	250	3552	17801
isqp1	900	100	250	3552	17801
itointqor	0	50	1	48	115

Table 6: Problem profiles of selected MIQPs from the Hans D. Mittelmann's collections

Method	DW2				`Cplex`			
Problem	Obj	Gap	Time (Found)	Time (Proven)	Obj	Gap	Time (Found)	Time (Proven)
ibell3a	87875.03*	0	21.14	36.47	87875.03*	0	5.56	6.47
ibienst1	34.21	0	23.74	‡	34.21	0	226.87	‡
ilaser0	2412537.96	0.001	287.57	‡	2412505.12	0	33.24	‡
iportfolio	-0.49*	0	58.71	260.83	–	NA		‡
iqiu	-126.66	0	101.19	‡	-127.08*	0	140.95	206.87
isqp	-21000.45	0	154.45	‡	-21000.45	0	8.11	‡
isqp0	-20319.51	0	7.24	‡	-20319.51*	0	2.03	19.71
isqp1	-18992.68	0	8.48	‡	-18992.68*	0	3.74	36.47
itointqor	-1146.7*	0	118.14	287.61	-1146.7	0	141.5	‡

[1] NA: Statistics not available.
[2] * : Proven optimality.
[3] –: No solution found.
[4] ‡: 5-minute time limit reached.

Table 7: Computational comparison of DW2 with `Cplex` 12.1

References

1. AMPL: A modeling language for mathematical programming (www.ampl.com).
2. CMU-IBM open source MINLP project (http://egon.cheme.cmu.edu/ibm/page.htm).
3. COIN-OR Ipopt (http://www.coin-or.org/ipopt/).
4. Hans d. mittelmann's MIQP test problems. http://plato.asu.edu/ftp/miqp.html.
5. IBM Cplex optimizer (http://www.ibm.com/).
6. K. Abhishek, S. Leyffer, and J. Linderoth. Feasibility pump heuristics for mixed integer nonlinear programs. Unpublished working paper, 2008.
7. T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.
8. E. Andersen and Y. Ye. A computational study of the homogeneous algorithm for large-scale convex optimization. *Computational Optimization and Applications*, 10(3):243–269, 1998.
9. E. Andersen and Y. Ye. On a homogeneous algorithm for the monotone complementarity problem. *Mathematical Programming*, 84:375–399, 1999.
10. D. Baena and J. Castro. Using the analytic center in the feasibility pump. *Operations Research Letters*, 39(5):310–317, 2011.
11. E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. Octane: A new heuristic for pure 0-1 programs. *Operations Research*, 49(2):207–225, 2001.
12. S. Baumert, A. Ghate, S. Kiatsupaibul, Y. Shen, R. L. Smith, and Z. B. Zabinsky. Discrete hit-and-run for sampling points from arbitrary distributions over subsets of integer hyper-rectangles. *Operations Research*, 57:727–739, 2009.
13. L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.
14. D. Bertsimas and S. Vempala. Solving convex programs by random walks. *The Journal of the ACM*, 51(4):540–556, 2004.
15. P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009.
16. P. Bonami and J. Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51(2):729–747, 2012.
17. P. Bonami, M. Kilinç, and J. Linderoth. Algorithms and software for convex MINLP. *Discrete Optimization*, 5:186–204, 2008.
18. C. D'Ambrosio, A. Frangioni, L. Liberti, and A. Lodi. Experiments with a feasibility pump approach for nonconvex minlps. *Lecture Notes in Computer Science*, 6049:350–360, 2010.
19. C. D'Ambrosio, A. Frangioni, L. Liberti, and A. Lodi. A storm of feasibility pumps for nonconvex minlp. *Mathematical Programming*, 136(2):375–402, 2012.
20. E. Danna, E. Rothberg, and C. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.
21. E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
22. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.
23. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003.
24. M. Fischetti and D. Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1:201–222, 2009.
25. K.-L. Huang and S. Mehrotra. Solution of monotone complementarity and general convex programming problems using a modified potential reduction interior point method. http://www.optimization-online.org/DB_HTML/2012/04/3431.html, 2012.
26. K.-L. Huang and S. Mehrotra. An empirical evaluation of walk-and-round heuristics for mixed integer linear programs. *Computational Optimization and Applications*, 2012 (to appear).
27. R. Kannan and H. Narayanan. Random walks on polytopes and an affine interior point method for linear programming. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 561–570, 2009.

28. R. Kannan and S. Vempala. Sampling lattice points. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 696–700, 1997.
29. L. Lovász. Hit-and-run mixes fast. *Mathematical Programming*, 86(3):443–461, 1999.
30. L. Lovász and S. Vempala. Hit-and-run from a corner. *SIAM Journal on Computing*, 35(4):985–1005, 2006.
31. S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.
32. S. Mehrotra and K.-L. Huang. Computational experience with a modified potential reduction algorithm for linear programming. *Optimization Methods And Software*, 27(4-5):865–891, 2012.
33. J. Naoum-Sawaya. Recursive central rounding heuristic for mixed integer programs. *Computers & Operations Research*, 43:191–200, 2014.
34. H. Narayanan. Randomized interior point methods for sampling and optimization. `http://arxiv.org/abs/arXiv:0911.3950`, 2009.
35. R. Smith. Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.
36. S. Vempala. Geometric random walks: a survey. *Combinatorial and Computational Geometry*, 52:573–612, 2005.
37. Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley, 1997.
38. Z. Zabinsky, R. Smith, J. McDonald, H. Romeijn, and D. Kaufman. Improving hit-and-run for global optimization. *Journal of Global Optimization*, 3(2):171–192, 1993.