European Journal of Operational Research 263 (2017) 62–71

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A hybrid primal heuristic for finding feasible solutions to mixed integer programs

Carlos E. Andrade[a,*], Shabbir Ahmed[a], George L. Nemhauser[a], Yufen Shao[b]

[a] H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology, 755 Ferst Drive NW, Atlanta, GA 30332, USA
[b] ExxonMobil Upstream Research Company, 3120 Buffalo Speedway, Houston, TX 77098, USA

ARTICLE INFO

Article history:
Received 17 June 2016
Accepted 3 May 2017
Available online 9 May 2017

Keywords:
Integer programming
Primal heuristics
Feasibility

ABSTRACT

We present a new framework for finding feasible solutions to mixed integer programs (MIP). We use the feasibility pump heuristic coupled to a biased random-key genetic algorithm (BRKGA). The feasibility pump heuristic attempts to find a feasible solution to a MIP by first rounding a solution to the linear programming (LP) relaxation to an integer (but not necessarily feasible) solution and then projecting it to a feasible solution to the LP relaxation. The BRKGA is able to build a pool of projected and rounded but not necessarily feasible solutions and to combine them using information from previous projections. This information is also used to fix variables during the process to speed up the solution of the LP relaxations, and to reduce the problem size in enumeration phases. Experimental results show that this approach is able to find feasible solutions for instances where the original feasibility pump or a commercial mixed integer programming solver often fail.

1. Introduction

An important feature of modern mixed integer programming (MIP) algorithms is the ability to effectively find feasible solutions quickly. A variety of heuristic methods have been developed for this purpose. See (Berthold & Hendel, 2015; Fischetti & Salvagnin, 2009; Glover & Laguna, 1997a, 1997b) for surveys of these heuristics. The *Feasibility Pump* heuristic from Fischetti, Glover, and Lodi (2005) has received a lot of attention because of its ability and efficiency to find feasible solutions relatively quickly. Several commercial and open source solvers use internal implementations of the feasibility pump. The feasibility pump and its improvements and variants (Achterberg & Berthold, 2007; Baena & Castro, 2011; Boland et al., 2014; Fischetti & Salvagnin, 2009; Santis, Lucidi, & Rinaldi, 2014) are built on a clever and straightforward idea for finding a feasible solution to a MIP by first rounding a solution to the LP relaxation to an integer (but not necessarily feasible) solution and then projecting it to a feasible solution to the linear programming (LP) relaxation. Essentially, the feasibility pump can be considered as a local search procedure operating over a search neighborhood consisting of linear projections and roundings. Due to this, the feasibility pump suffers from fast convergence to local optima and the inability to escape from them. This fact was noted by the authors of the original method in Fischetti et al. (2005) and also in subsequent works (Achterberg & Berthold, 2007; Bertacco, Fischetti, & Lodi, 2007; Fischetti & Salvagnin, 2009). To circumvent this situation, several types of perturbations were proposed. Most of them occur when a cycling is detected. Although such perturbations drive the algorithm to explore other regions of the search space, they often incur information loss since the feasibility pump has no memory. For instance, some variables may take the same value across several restarts, and they could be fixed to reduce the problem size. However, the feasibility pump does not use this information.

We present a primal framework that makes use of information collected during the iterations of the feasibility pump. Such information consists of fractional infeasible solutions and their respective final roundings. The framework is able to combine the roundings that are used in subsequent feasibility pump calls. In addition, the framework also provides valuable information that can be used for variable fixing which reduces the size of LP relaxations that need to be solved. We use a biased random-key genetic algorithm to evolve a set of roundings and projections but any population-based method may be used. We restrict the presentation to pure binary and mixed binary MIPs, although the techniques presented here can be easily adapted to general MIPs.

The structure of this paper is as follows. Section 2 describes the basic idea of the feasibility pump, improvements, and variations.

* Corresponding author. Present address: AT&T Labs Research, 200 South Laurel Avenue, Middletown, NJ 07748, USA.
E-mail addresses: carlos.andrade@gatech.edu, cea@research.att.com (C.E. Andrade), shabbir.ahmed@isye.gatech.edu (S. Ahmed), george.nemhauser@isye.gatech.edu (G.L. Nemhauser), yufen.shao@exxonmobil.com (Y. Shao).

http://dx.doi.org/10.1016/j.ejor.2017.05.003
0377-2217/© 2017 Elsevier B.V. All rights reserved.

In Section 3, we propose a framework used to improve the rounding and variable fixing processes. Section 4 describes the implementation details and experimental environment. Section 5 reports the computational experiments comparing the proposed framework with the feasibility pump 2.0, and a commercial solver. In Section 6, we give some conclusions.

2. The feasibility pump

2.1. Basic procedure

Here we present a high level overview of the feasibility pump. For a more detailed descriptions, see Fischetti and Salvagnin (2009) and Achterberg and Berthold (2007). Without loss of generality, consider the following mixed integer linear program

$$\text{MIP} = \min_x \{c^T x : Ax \le b, \ell \le x \le u, x_i \in \{0,1\} \text{ for all } i \in I\},$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c, \ell, u \in \mathbb{R}^n$, and $I \subseteq N = \{1, \dots, n\}$. Let $P = \{x : Ax \le b, \ell \le x \le u\}$ be the feasible region of the LP relaxation. Consider $x^* \in P$ a fractional feasible solution for MIP (we say that x^* is LP-feasible). Let $\tilde{x}$ satisfy $\tilde{x}_i \in \mathbb{Z}$ for all $i \in I$, be an integer and not necessarily feasible solution. We compute the distance between x^* and $\tilde{x}$ using

$$\Delta(x^*, \tilde{x}) = \sum_{i\in I} |x_i^* - \tilde{x}_i|. \quad (1)$$

Using this concept of distance, we fix $\tilde{x}$ and solve the following problem:

$$\text{LP}^* = \min_{x'} \{\Delta(x', \tilde{x}) : Ax' \le b, \ell \le x' \le u, x'_i \in \mathbb{R} \text{ for all } i \in N\},$$

i.e., the original objective function is replaced by the minimization of (1) and all integrality constraints are dropped. LP* is called *Linear Program (LP) projection*. Objective function (1) can be linearized by reformulating LP*, see Fischetti and Salvagnin (2009) for details. Let x^* be an optimal solution for LP*. If x^* is integer ($x_i^* \in \mathbb{Z}$, for all $i \in I$), or $\Delta(x^*, \tilde{x}) = 0$, we have obtained an integer feasible solution and we may stop the algorithm. Otherwise, we take $\tilde{x} = \lceil x^* \rfloor$ as a *rounding* of x^* and solve the LP projection again. If during this process, we find a rounded solution already found in previous iterations, we have detected a cycle. In this case, random perturbations are performed to escape from the local minima.

A number of variations of the feasibility pump has been proposed. Bertacco et al. (2007) proposed a 3-stage algorithm where the idea is to apply pumping cycles for different sets of variables in each stage. This variation is the most used in practice. In Achterberg and Berthold (2007), a variation called *Objective Feasibility Pump* is proposed where the distance function Δ is used together with the original objective function. Other authors considered modifications during the rounding phase. In Fischetti and Salvagnin (2009), the so called Feasibility Pump 2.0 makes use of constraint propagation techniques during the rounding phase. This version is considered the most effective variation of the feasibility pump. In Baena and Castro (2011), roundings are performed over a line segment between a computed feasible point and the analytic center of the relaxed linear problem. In Boland et al. (2014), a similar approach is developed where the authors used rays directed to the feasible region instead of the analytic center. Hanafi, Lazić, and Mladenović (2010) proposed a Variable Neighborhood Pump (VNP) heuristic that combines Variable Neighborhood Branching (VNB) local search (Hansen, Mladenović, & Urošević, 2006) with the feasibility pump.

2.2. Computational burden and loss of information

The key aspect of the feasibility pump is to alternate between LP-feasible solutions obtained from a LP projection, and infeasible integer solutions obtained from roundings of non-integer solutions. Both operations can be very computationally demanding. While simple rounding is very fast (but not so effective), constraint propagation rounding can take many CPU cycles depending on the implementation details. But either rounding schemes are one or more orders of magnitude faster than the LP projection that requires solving a linear program which can be very costly. Although solvers usually keep track of the information of previous optimizations, each LP projection has a different objective function (due to the new roundings) and may change the direction of optimization making the previous optimization information useless.

In the original feasibility pump and its variants, a large amount of computational effort may be required until the algorithm finds a local minima or detects cycling. However, most information generated during this optimization is lost. Some implicit information is retained since the iterations between roundings and LP projections tend to “fix” some variables, i.e., some variables will have the same value during several pumping cycles. In this sense, all information the feasibility pump uses is very localized and immediate with respect to the last iteration.

Other factors that contribute to the information loss are the restarts. Restarts help the algorithm to explore other regions in the solution landscape. However, the way that a restart is implemented in the feasibility pump redirects the search to other regions without carrying information other than the shifting based on the fractionality. Therefore, the algorithm may “forget” the frequent values of some variables which can be used to reduce the dimensionality of the problem.

We propose to use such information by means of a evolutionary framework. The collected information is kept and evolve a pool of projections and roundings which are used to fix variables, perform local MIP searches, and create other fractional (possibly not feasible) solutions. Next, we describe this approach.

3. A primal framework for the feasibility pump

Our approach is based on a *pool* or *population* of roundings and LP projections that are evolved towards “less infeasible” solutions. In this context, a *solution* is a pair of vectors representing a rounding and a LP projection from pumping cycles, respectively. There are three main components. The first component is called the *evolutionary phase* responsible for applying the feasibility pump from several starting points and combining the results to obtain “less infeasible” solutions. The second component is called the *local MIP search phase* that uses strategies to fix and unfix variables based on the pool of solutions obtained in the evolutionary phase, and then applies an enumeration procedure. The third component is called the *fixing phase* used to reduce the dimensionality of the problem used in the evolutionary phase based on information from the pool of solutions. Next, we present the general framework and describe each phase in detail.

3.1. The general framework

Algorithm 1 describes the main procedures of our framework. The algorithm starts by preprocessing the instance and trying to reduce the domain of the variables (line 1). At this time, it also collects information from the LP relaxation that can be used to determine which type of variable fixings are to be performed and the initial percentage for the fixing phase. Dual information is collected and use to filter constraints in the local MIP search phase.

The initial pool of solutions or population is built from LP relaxations and random vectors. First in line 2, a solution to a relaxation which drops all integrality constraints is added to the population $\mathcal{P}$ (w.l.o.g., we consider a minimization problem). To include this relaxation in the initial population ensures that the algorithm will

Algorithm 1: General feasibility pump evolutionary framework.

```
1  Preprocess the instance and collect the new variable bounds;
2  $\mathcal{P} \leftarrow \text{argmin}\{x : x \in P\}$;
3  Let $\pi$ be the list of the binary variables ordered from the
   most fractional to least fractional in the solution of LP
   relaxation;
4  foreach $i \in \pi$ in the given order and until num_relax is not
   reached do
5      $\mathcal{P} \leftarrow \mathcal{P} \cup \text{argmin}\{x : x \in P \text{ and } x_i = 0\}$;
6      $\mathcal{P} \leftarrow \mathcal{P} \cup \text{argmin}\{x : x \in P \text{ and } x_i = 1\}$;
7  If $\mathcal{P}$ is not complete, fill it with random vectors $x \in [0, 1]^n$;
8  while a stopping criterion is not reach do
9      Phase 1: Perform the evolutionary step;
10     if % of frac. variables $\leq$ mip_threshold then
11         Phase 2: Perform local MIP search;
12         if feasible solution is found then Stop;
13     if iteration $\equiv 0$ mod fixing_frequency then
14         Phase 3: Perform variable fixing;
15         if fixing was successful then
16             $fp \leftarrow fp(1 + fcr)$;
17         else
18             $fp \leftarrow fp(1 - fcr)$;
19         if feasible solution is found then Stop;
```

perform, at least, one run similar to stage 1 of the traditional feasibility pump in the evolutionary phase. In line 3, the variables are sorted from the most to least fractional. This order is used in lines 4–6 to generate several relaxations by fixing each variable to zero (line 5) or to one (line 6) until either all variables are considered or a given limit parameter *num_relax* is reached. In line 7, $\mathcal{P}$ is filled in with random vectors if the number of generated relaxations is less than the intended size for $\mathcal{P}$.

The three major phases are connected though a major loop that stops when either a feasible solution is found, or a time limit is reached, or the algorithm has stalled for a given number of iterations. The main loop begins with the application of the evolutionary step (line 9). Subsequently, if the percentage of fractional variables of the best solution so far is less than or equal to the parameter *mip_threshold*, a local MIP search is launched (line 10). If no feasible solution is found, the algorithm proceeds to variable fixing (line 13). The variable fixing only happens periodically according to the parameter *fixing_frequency*. After a successful fixing, the fixing percentage *fp* is incremented geometrically according to the fixing change rate parameter *fcr*, a percentage in the range [0, 1]. If the fixing fails, we reduce *fp* by the same amount.

3.2. Evolutionary phase

The evolutionary phase is responsible for collecting data from applications of the feasibility pump starting from different fractional vectors. To perform this task, we use the Biased Random-Key Genetic Algorithm (BRKGA). The BRKGA was proposed by Gonçalves and de Almeida (2002) and Ericsson, Resende, and Pardalos (2002), and has been shown to be useful to a wide variety of combinatorial problems (Andrade, Miyazawa, & Resende, 2013; Andrade, Resende, Karloff, & Miyazawa, 2014; Andrade et al., 2015a; Andrade, Toso, Resende, & Miyazawa, 2015b; Gonçalves & Resende, 2011b; Stefanello, Aggarwal, Buriol, Gonçalves, & Resende, 2015), and machine learning (Caserta & Reiners, 2016).

The BRKGA works with a pool of solutions $\mathcal{P}$ called *population*. Each solution is called an *individual*. The population is partitioned into the *elite set* $\mathcal{P}_e$ and the *non-elite set* according to some performance metric (*fitness*). The solutions in the elite set are the best of the whole population. The non-elite set contains the remaining individuals ($\mathcal{P} \setminus \mathcal{P}_e$). The evolutionary step is done as follows: the elite set is copied to the population of the next generation. A small percentage of random solutions is also included in this population. These random individuals are called *mutants*. The remaining slots are filled in with offsprings from the combination (*mating*) of a randomly chosen individual from the elite set and a randomly chosen individual from the non-elite set, using biased uniform crossover, i.e., with probability ρ, a component is taken from the elite individual, otherwise the component is take from the non-elite individual. Therefore, when $\rho > 0.5$, the inheritance is biased towards the elite individual genes. In the BRKGA, the population is represented by a n-dimensional unitary hypercube where each individual is represented by a vector $v \in [0, 1]^n$, for a given n. To obtain a solution, it is necessary to apply a function $f : [0, 1]^n \to \mathcal{S}$ that maps a vector v to a solution in the space $\mathcal{S}$ which is the space of solutions of the problem to be solved. Typically, f also computes the fitness of the individual. Function f is called *decoder*. For a detailed description, see Gonçalves and Resende (2011a).

In this paper, we consider an individual as a tuple $(v^*, \tilde{v})$ where $v^* \in [0, 1]^n$ represents a fractional solution, $\tilde{v} \in \{0, 1\}^n$ represents an integer solution or a rounding with respect to the binary variables, where n is the number of binary variables.

Algorithm 2: Decoder using feasibility pump.

```
Input: vector $v^* \in [0, 1]^n$.
Output: tuple $(v^* \in [0, 1]^n, \tilde{v} \in \{0, 1\}^n, \Pi$ : number of frac.
        vars.).
1  $x^* \leftarrow v^*$;
2  best_dist $\leftarrow \infty$;
3  while a given number of iterations without improvement is not
   reached do
4      $\tilde{x} \leftarrow$ ConstraintPropagationRounding($x^*$);
5      if cycle is detected and perturbation is active then
6          Perturb $\tilde{x}$;
7      $x^* \leftarrow \text{argmin}\{\Delta(x', \tilde{x}) : x \in P, x_i \in \mathbb{R} \text{ for all } i \in N\}$;
8      if $\Delta(x^*, \tilde{x}) <$ best_dist then
9          best_dist $\leftarrow \Delta(x^*, \tilde{x})$;
10         $v^* \leftarrow x^*$;
11         $\tilde{v} \leftarrow \tilde{x}$;
12         $\Pi \leftarrow$ number of fractional variables in $x^*$;
13     if $x^*$ is integer then return $(x^*, x^*, 0)$;
14 return $(v^*, \tilde{v}, \Pi)$;
```

The decoder presented in Algorithm 2 implements the basic scheme of the feasibility pump with some small modifications. The first modification is in line 1 which takes the fractional input vector instead of the LP relaxation. This allows us to perform the feasibility pump from different fractional solutions or “starting points.” In the rounding phase, we always make use of the constraint propagation rounding, as shown in line 4, due to its good performance (Fischetti & Salvagnin, 2009). In line 5, we added an option to not use the perturbation phase, restricting the random signals to within the BRKGA. Deactivating the perturbation phase speeds up the decoding phase although it decreases it the convergence rate, as we explain later. Note that in the LP projection

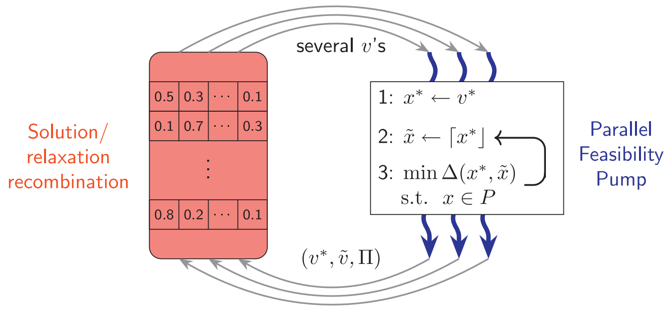
several v's
0.5 0.3 ··· 0.1
0.1 0.7 ··· 0.3
0.8 0.2 ··· 0.1
Solution/ relaxation recombination
1: $x^* \leftarrow v^*$
2: $\tilde{x} \leftarrow \lceil x^* \rfloor$
3: $\min \Delta(x^*, \tilde{x})$ s.t. $x \in P$
Parallel Feasibility Pump
$(v^*, \tilde{v}, \Pi)$

Fig. 1. General evolutionary scheme using BRKGA (box on the left) and parallel applications of the feasibility pump (box on the right). For each input vector v, a tuple $(v^*, \tilde{v}, \Pi)$ is returned.

phase (line 7), we may choose between the original distance function or the function used in the objective feasibility pump. We also keep the best solution found as shown in lines 8–12. We use the distance between the LP relaxation and the rounding as the performance criterion, as is done in the original feasibility pump. As stopping criterion, we use a given number of iterations without improvement in the best solution. Note that in case of non-active perturbation, we can stop immediately after detecting a cycle. In line 14, the decoder returns to the BRKGA framework a tuple containing the best relaxation v^*, the respective rouding $\tilde{v}$, and number of fractional variables Π, used as performance measure.

The interesting aspect of this approach is the independence of each application of the decoder. Since v^* can be any vector, we can use the decoder to explore different regions around the MIP polyhedron. Note that it is not required that v^* be LP-feasible, since LP-feasibility will be restored in the LP projection. This also allows parallel applications of the decoder. We give more details about these aspects in Section 4.2.

After decoding the individuals in the population, the BRKGA proceeds with the evolutionary step as explained earlier. Fig. 1 shows the general scheme. Two aspects are important here: the ranking of the individuals and the crossover process. Evolutionary algorithms typically make use of some performance measure (Π) to track the evolutionary process, and this is the case in BRKGA. Since we are dealing with infeasibility, we may choose to use as performance measure either the number of fractional variables, the fractionality (defined as $frac(x) = \sum_{i \in I} |x_i - \lfloor x_i + 0.5 \rfloor|$), or the distance $\Delta(x^*, \tilde{x})$. Although these functions are closely related, their performance varies a lot. Indeed, it is easy to see that the fractionality is always less than or equal to the number of fractional variables. The fractionality also may mislead the evolutionary process. For example, supposed we have two vectors $v_1 = (0.1, 0.9)$ and $v_2 = (0.5, 1.0)$. Since $frac(v_1) < frac(v_2)$, the algorithm will rank v_1 better than v_2. After some iterations, the population may have several individuals with low fractionality but high number of fractional variables. Although this fact can help the rounding process, it impairs effective variable fixing. In preliminary experiments, using fractionality as performance measure did not perform as well as using the number of fractional variables.

With respect to the crossover process, we only perform the combination among the fractional vectors, i.e., v^*'s. Although it is possible to perform this operation with the roundings as well, in preliminary trials, it did not yield any advantages. Indeed, one can observe that only the fractional vectors are considered as input in the decoder.

Note that the core function of the evolutionary phase is to combine different fractional solutions from different applications of the feasibility pump and it is independent of the utilization BRKGA. These fractional solutions compose a pool used to combine the solutions and perform variable fixing as we will explain in the next sections.

3.3. Local MIP search phase

Usually after a few iterations of the evolutionary phase, some variables will have the same values in most solutions. In addition, *high quality solutions* have only a small percentage of integer violations varying from 2% to 5% of the total number of integer variables. In such cases, it is natural to think of an enumeration procedure for finding high quality integer feasible solutions. Suppose we choose the best individual according to the performance metric (again, typically number of fractional variables), fix the integer variables and let the fractional variables be free. Since this individual comes from the feasibility pump cycles, the fractional variables can be seen as critical variables which the model is very sensitive to. Thus, performing an enumeration procedure on these variables will only yield infeasible solutions. One way to remedy this is to relax some other variables to increase the degree of freedom in the enumeration.

The local MIP search phase follows the above observation but, instead of using the best individual available, it uses a combination of a subset of best individuals from the pool of solutions evolved from the evolutionary phase. An aggressive variable fixing is performed trying to fix as many variables as possible. After possibly unfixing some variables initially fixed, an enumeration procedure is applied for a given time limit. Algorithm 3 depicts this procedure. In line 1, the algorithm builds a normalized histogram summing element-wise the roundings from each individual of the set $\mathcal{P}' \subseteq \mathcal{P}$. We choose the least fractional individuals in the population since they are supposed to be close to a feasible solution. In lines 2–8, the variables are fixed according a discrepancy level parameter *dl*. The discrepancy level dictates the tolerance of a variable to take different values in each individual of $\mathcal{P}'$. If the discrepancy level is zero, only variables whose value is exactly the same across $\mathcal{P}'$ are fixed. Variables that do not fulfill this criterion, are set free and added to the set U for further use. If the discrepancy level is greater than zero, the algorithm fixes the variable to the most common value with some tolerance. For instance, if $dl = 0.05$, the algorithm will fix variables that differ, at most, by 5% from the individuals in $\mathcal{P}'$.

Frequently the number of fixed variables is too large and leads to violated constraints. In this case, lines 9–15 traverse each constraint verifying if they are violated. For a violated constraint, some fixed binary variables are unfixed according to the type of constraint and the coefficients of those variables in that constraint. Although this procedure removes constraint violations, it does not guarantee feasibility. Also note that if the fixing leads to a violated constraint, we can remove it by adding the following cut:

$$\sum_{i \in V : \tilde{x}_i = 1} x_i + \sum_{i \in V : \tilde{x}_i = 0} (1 - x_i) \le |V| - 1, \qquad (2)$$

where V is the set of fixed variables (prior to unfixing) of the current constraint. Inequality (2) is able to cut partial fixings based on constraint violation and help in the enumeration phases. Note that the number of inserted cuts can be as large as the number of the original constraints. However, in our experiments, the number of generated cuts is relatively small and contributes to finding feasible solutions.

The algorithm thus proceeds with the first enumeration phase for a given period of time (line 16). If a feasible solution is found or infeasibility is not proven, the algorithm returns it and aborts the execution. If infeasibility is proven, we add a cut considering all fixed variables (line 19). Note that if it did not find a feasible solution, it is possible that the fixing was too aggressive, even after some unfixing. To mitigate this effect, lines 22–35 unfix some variables. This procedure is based on the neighborhood of fixed variables given the subset of constraints $C' \subseteq C$, where C is the set of constraints of the MIP. The rationale is that the fixed variables

Algorithm 3: Local MIP search.

```
Input: Set of individuals $\mathcal{P}' \subseteq \mathcal{P}$; set of constraints $C' \subseteq C$;
       discrepancy level $dl$; unfixing level $ul$.
Output: A feasible solution if found.
1  Let $h = \sum_{(v^*,\tilde{v})\in\mathcal{P}'} \tilde{v}/|\mathcal{P}'|$; // Element-wise
2  $U \leftarrow \emptyset$;
3  foreach $i \in I$ do
4      if $h_i < dl$ or $h_i > 1 - dl$ then
5          Fix variable $i$ to round($h_i$);
6      else
7          Let variable $i$ be free;
8          $U \leftarrow U \cup \{i\}$;
9  foreach $c \in C$ do
10     if constraint $c$ is violated then
11         foreach variable $i \in I$ in constraint $c$ do
12             Let $coef$ be the coefficient of $i$ in $c$;
13             Let variable $i$ be free either if 1) $c$ is an equality; or
               2) $i$ is fixed to 0 and 2.1) $c$ is '$\leq$' inequality and
               $coef < 0$; or 2.2) $c$ is '$\geq$' inequality and $coef > 0$;
14             $U \leftarrow U \cup \{i\}$ with the same conditions above;
15         Insert a Cut (2) considering $V$ as the set of binary
           variables in $c$;
16 Perform enumeration using the resulting fixing for a given
   period of time;
17 if feasible solution is found then return the solution;
18 if the model is infeasible then
19     Insert a Cut (2) considering $V$ as the set of fixed variables;
20     For each $i \in I$, let
       $C_i = \{c \in C' : i$ has no zero coefficient in $c\}$;
21     For each $j \in C'$, let
       $V_j = \{i \in I : i$ has no zero coefficient in $j\}$;
22     $C^c \leftarrow \emptyset$;
23     $V^c \leftarrow \emptyset$;
24     $k \leftarrow 0$;
25     while $k < ul$ and $U \neq \emptyset$ do
26         $U' \leftarrow \emptyset$;
27         foreach $i \in U$ do
28             foreach $j \in C_i$ such that $j \notin C^c$ do
29                 $C^c \leftarrow C^c \cup \{j\}$;
30                 foreach $i' \in V_j$ such that $i' \notin V^c$ do
31                     Unfix variable $i'$;
32                     $V^c \leftarrow V^c \cup \{i\}$;
33                     $U' \leftarrow U' \cup \{i\}$;
34         $U \leftarrow U'$;
35         $k \leftarrow k + 1$;
36     Perform enumeration using the resulting fixing for a given
       period of time;
37 return feasible solution if found;
```

force the free variables to be set to certain values making the constraints their share, say A', feasible. However, once the free variables are set, constraints other than A' become infeasible, even A' have no fixed variables. Therefore, making free variables that share the same (but maybe not all) constraints with fixed variables, may bring feasibility. First, two cross-reference maps are created between the variables and the given constraints (lines 20–21). Then the set of checked constraints C^c and the set of checked variables V^c are initialized as empty. These sets are used to keep track of the variables and constraints already analyzed. The algorithm proceeds in a recursive fashion: it unfixes the variables that share some constraints in C^c with some other free variable in the current iteration. The depth is controlled by the unfix level parameter ul in the main loop of line 25. For each free variable i' (line 28), the algorithm traverses the constraints to which i' belongs (line 30), and unfixes the other variables of these constraints.

Note that the unfixing process can be very sensitive according to the model. While in sparse models few variables are unfixed, the opposite occurs for dense models breaking the functionality of the local MIP search phase. One way to control this is to limit the unfixing depth using the parameter ul. Another way is to use a limited number of constraints in C'. In the latter case, an important question is which constraints to choose. Following the suggestion of Li, Ergun, and Nemhauser (2015), we consider two complementary strategies to choose such constraints. Both strategies are tied to the solution obtained from the LP relaxation of the original problem. First, we consider in C', only constraints with positive dual values in the LP relaxation. The idea is to capture the "degree of importance" or marginal price of each constraint in the relaxation. Using this strategy, we may avoid unfixing variables in the unimportant constraints. In some cases, the dual values of all constraints are zero. Therefore, we implement a second strategy that is to rank the constraints by their slack values. Indeed, Li et al. (2015) suggest combining the two strategies, first considering the duals values and then the slack values.

After unfixing variables, we perform an enumeration procedure again, in line 36. If the algorithm finds a feasible solution, the last is returned and we stop. Otherwise, the algorithm restores the bounds of the fixed variables and returns to the main loop, described in Section 3.1.

3.4. Fixing phase

As observed in Section 3.3, after some iterations of the evolutionary phase, some variables tend to have the same values among the relaxations and roundings. Therefore, such variables can be fixed to reduce the problem size and, consequently, the time to solve the LP relaxations.

The fixing phase tries to reduce the size of the problem by fixing variables using information from the pool of solutions evolved by the evolutionary phase as in the local MIP search phase. While in the local MIP search phase we aggressively fix variables which frequently leads to constraint violation, in the fixing phase we want to keep LP feasibility. Algorithm 4 presents the scheme of this procedure. First, we just restore the bounds for each variable obtained after the instance preprocessing and constraint propagation (line 1). Next, we build a histogram h summing element-wise the roundings from each individual of the set $\mathcal{P}' \subseteq \mathcal{P}$. Using this histogram, we then sort the variables (line 3) according to the fixing type parameter ft which can be given by the user or determined automatically. ft is a categorical parameter that can be set to either *most zeros*, *most ones*, or *most disagreements*. In models that naturally have most binary variables set to zero in a feasible solution, we may have more success in fixing variables which appears with value zero in $\mathcal{P}'$. The opposite is true in models with naturally more variables set to one. We also can choose to fix the variables with most disagreements among individuals of $\mathcal{P}'$. In this case, the number of successful fixed variables will be smaller than the other options since, in general, the model is very sensitive to changes in these variables. Note that we can determine ft automatically from the original LP relaxation, counting the number of variables set to zero or to one in the solution.

Algorithm 4: Fixing variables based on pool of solutions.

```
Input: Set of individuals $\mathcal{P}' \subseteq \mathcal{P}$, fixing type ft, fixing
       percentage fp.
Output: A variable fixing or a feasible solution if found.
1  Restore the variables original bounds;
2  Let $h = \sum_{(v^*, \tilde{v}) \in \mathcal{P}'} \tilde{v}$; // Element-wise
3  Using h for comparison, sort the binary variables according to
   ft: either most zeros, most ones, or most disagreements. Let
   $\pi$ be variable indices according to the given order;
4  Try to fix fp of binary variables according to $\pi$;
5  if successful fixing then
6      Perform enumeration using the resulting fixing for a given
       period of time;
7      if a feasible solution is found then
8          return the feasible solution;
9      else
10         return the variable fixing;
11 else
12     return failure in the fixing;
```

In line 4, we try to fix a given percentage *fp* of variables according to the fixing type. Since we want to fix variables and keep LP-feasibility, we must check if that fixing is LP feasible. One approach is try to fix the block of *fp* variables at once and just report failure in case the LP is infeasible. However, this methodology is too conservative and usually results in invalid fixings, even for a small *fp*. We may try to fix variables one at a time, leaving free the ones that are not possible to fix. This option is more effective in terms of the number of fixed variables but is very time consuming since we must check the feasibility of each individual fixing. To mitigate the fixing time, we implement a strategy that divides the variables in blocks (according to π), and tries to fix each block one at time. If a given block cannot be fixed, we try to fix each variable of that block individually (first to the required value, then to the opposite in case of failure in the first value). In the worst case scenario (when the fixing fails for each variable), this strategy is $O(n)$ but is faster than the naive linear strategy in practice. It is important to note that this strategy does not necessarily fix all *fp* variables.

If the algorithm is able to fix variables, a short enumeration is performed using that fixing, for a given (typically short) time (line 6). Sometimes the fixing is enough to eliminate the major part of the search space and leads to a feasible solution quickly. We choose to use the original objective function in order to obtain a high quality feasible solution, and the algorithm stops if a feasible solution is found. If not, the algorithm returns the fixing to be used in the feasibility pump cycles in the evolutionary phase. In case of total failure, the algorithm just reports it and proceeds as usual. Although we can add cuts in this case , we did not find it to be beneficial to do so since these dense cuts increased the LP solution times.

4. Experimental setup

4.1. Instances

We tested two groups of instances. The first group is composed of pure binary and mixed binary problems from MIPLIB 2010. This subset has 273 instances. Since the proposed evolutionary framework is computationally intensive (because it uses several runs of the feasibility pump), we only consider the 259 instances for which the LP relaxation takes less than 30 seconds to be solved in our environment. In this group, we disregarded instances `neos-952987` and `ns1778858` since feasible solutions are not known for these problems, and instance `ns1663818` due to numerical problems during our experiments.

The second group of instances comes from the MIRPLib (Papageorgiou, Nemhauser, Sokol, Cheon, & Keha, 2014). This set is comprised of complex scheduling models from maritime routing problems from the petrochemical industry where vessels load cargo at production ports and unload at consumption ports, subject to constraints such as cargo limit, berth time, maximum load in a given port, travel times, fuel consumption, etc. We considered the 28 instances from group 1 described in Papageorgiou et al. (2014) for which it is very hard to find feasible solutions due to tight (un)loading constraints. The instances vary from 8045 variables and 7909 constraints to 109,519 variables and 108,566 constraints.

4.2. Computational environment

The experiments were conducted on identical machines with 12-core Intel Xeon 2.4 gigahertz CPUs with Hyper-Threading and 48 gigabytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. The algorithms are implemented in C++ using the GNU g++ compiler version 4.9. To solve LP relaxations and MIPs, we use the IBM ILOG CPLEX Optimizer version 12.6.1.0. We also use the original feasibility pump 2.0 (FP 2.0) code provided by Fischetti and Salvagnin (2009).

4.3. Parameter settings

For the proposed algorithm, we performed a fine parameter tuning using *iterated racing* (Birattari, Yuan, Balaprakash, & Stützle, 2010). The following values were suggested for BRKGA parameters: population size of $p = 100$; elite size $p_e = 30$; number of mutants introduced at each generation $p_m = 30$; probability of biased crossover $\rho = 0.6$; *fp_iter_lim* = 10. For the feasibility pump heuristic used in the decoder, the following values were suggested: iteration limit without improvements *fp_iter_lim* = 10; weak perturbation *fp_t* = 10; strong perturbation lower bound *fp_rho_lb* = −0.2; and upper bound *fp_rho_ub* = 0.5. (for details on feasibility pump parameters, refer to Bertacco et al. (2007); Fischetti and Salvagnin (2009). For Algorithm 1, the following parameter settings were chosen: the number of relaxations in first population *num_relax* = 1; *fixing_frequency* = 5; initial fixing percentage *fp* = 10% with incremental step *fcr* = 0.05 (used on Algorithm 4); percentage of fractional variables to activate the local MIP search phase *mip_threshold* = 4%. We set $\mathcal{P}' = \mathcal{P}_e$ for both Algorithms 3 and 4. For Algorithm 3, the enumeration time was set to 60 seconds for instances with time limit of 1 hour and to 300 seconds for instances with time limit of 5 hours. In Algorithm 4, 10 seconds were used to perform a short enumeration.

The above feasibility pump settings were used only in the proposed algorithm. For experiments using the original feasibility pump code, we use the values suggested in Fischetti and Salvagnin (2009). For standalone CPLEX runs, we set MIP emphasis to feasibility, heuristic frequency to one, and the feasibility heuristic to one. We also use a callback to stop immediately when a feasible solution is found.

For each instance, ten independents runs (using different seeds) were performed for each algorithm. For instances with prefix `LR1`, we limited the runs to 1 hour (wall-clock). For instances with prefix `LR2`, we found no feasible solution in one hour experiments. Therefore, we redid the experiments using a 5 hour time limit.

In addition to the time limit, we use 1000 iterations without improvement as a stopping parameter for BRKGA. For all algorithms, we allow 12 parallel threads. For further implementation details, refer to supplement materials.

5. Experimental results and discussion

5.1. Local MIP search effects

The local MIP search component has three major control parameters that can affect the search: the discrepancy level *dl*; the unfixing level *ul*; and set of constraints C' used to unfix variables. We allow 0%, 5%, and 15% of discrepancy level when fixing variables in the local MIP search. For unfixing, level 0 means that the unfixing is not performed, and for level 1, only variables sharing constraints C' with free variables are unfixed. Preliminary tests showed that more unfixing is undesirable. The set C' was defined in three different ways: using all constraints, using only constraints with nonzero dual values in the LP relaxation, and using constraints with both nonzero dual value and zero slack value in the LP relaxation.

We performed a full factorial experiment using all instances collecting results of ten independent runs. For each design point, we have tested the decoder of the evolutionary phase (phase 1) with and without perturbation enabled. The results are shown in Table 1.

First, we observe that, in all cases, the percentage of runs resulting in a feasible solution is substantially smaller when perturbation is not enabled. This means that the noise introduced by BRKGA during the evolutionary process is not enough to reach a feasible solution. Indeed, BRKGA is able to learn starting points for the feasibility pump, by “soft-fixing” some variables. Even starting from a promising point in the feasible region, experiments show that few perturbations during the feasibility pump are necessary to achieve a feasible solution. These experiments raise the question of whether the evolutionary phase using BRKGA is really necessary or a simple application of the feasibility pump over several random vectors is enough. Section 5.3 gives evidence that the BRKGA evolutionary process is necessary to achieve good results.

Since disabling perturbations in the feasibility pump did not result in many feasible solutions, we decided to concentrate on the results with perturbation enabled. Fig. 2 depicts the boxplots

Table 1
Comparison among several strategies during local MIP search. The last two columns represents the percentage of runs where feasible solutions were found.

Discrepancy	Constraint filtering	Unfixing level	% of runs	
			Pert.	No Pert.
0%	All	0	49.44	20.81
		1	53.15	23.42
	Duals	0	49.44	20.07
		1	49.07	18.95
	Duals & slacks	0	48.69	21.18
		1	49.81	17.47
5%	All	0	41.26	18.21
		1	50.55	20.81
	Duals	0	39.40	19.33
		1	44.60	16.72
	Duals & slacks	0	40.89	17.84
		1	44.98	16.35
15%	All	0	34.94	16.35
		1	36.43	16.72
	Duals	0	33.45	15.61
		1	32.34	16.72
	Duals & slacks	0	33.08	15.98
		1	35.31	15.24

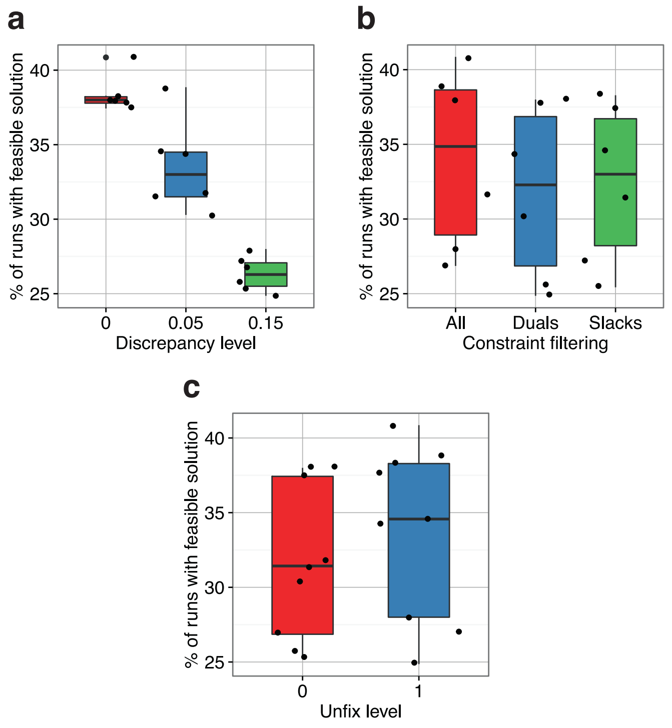

Fig. 2. Boxplots comparing the percentage of runs where a feasible solution was found.

for the strategies used during the local MIP search. In general, not allowing discrepancy leads to a feasible solution in an average of $38 \pm 1\%$ of the runs while discrepancy at 5% and 15% leads to feasible solutions in an average of $33 \pm 3\%$ and $24 \pm 1\%$ of the runs, respectively. To confirm the differences observed in Fig. 2a, we applied the Mann–Whitney–Wilcoxon U test to the results, performing a false discovery rate correction on the *p*-values. With a confidence level of 95%, these strategies differ among themselves significantly (all *p*-values are less than 0.05).

To consider all constraints in the unfixing step results in a feasible solution in an average of $34 \pm 5\%$ of the runs. Filtering constraints by duals values, and duals and slack values leads to feasible solutions in an average of $31 \pm 5\%$ and $32 \pm 5\%$ of the runs, respectively. Although using all constraints is better on average, these strategies do not differ significantly at a 95% confidence level. To unfix variables leads to feasible solutions in an average of $32 \pm 5\%$ of the cases while to not unfix variables succeeds on $31 \pm 5\%$. Again, the differences here are not statistically significant.

The best configuration is to not allow discrepancy between the roundings, use all constraints and perform the unfixing which results in feasible solutions in 39% of the runs. This means that, although the fixing procedure tries to fix as many variables as possible, the enumeration procedure needs a certain degree of freedom provided by variable unfixing.

5.2. Algorithm phases and feasible solutions

Since any phase can produce feasible solutions, a natural question is which phase produces the most feasible solutions and, as expected, the local MIP search phase produces the most feasible solutions. Local MIP search performs two enumerations that can produce feasible solutions. One of them is just after the variable fixing and insertion of cuts (line 16 of Algorithm 3), and the other after the variable unfixing (line 36). In most cases, a feasible solution was found between the first and the third enumerations. However, the average number of enumerations was 14 ± 34, with maximum of 382 iterations for instance `bnatt400`. For instances `LR1_1_DR1_4_VC3_V9a_t45` and `LR2_11_DR2_33_VC4_V11a_t45`, the unfixing procedure from Algorithm 3 was necessary to reach a feasible solution. For these instances, a feasible solution was found before unfixing in, respectively, 70% and 80% of the times. For instances `LR1_1_DR1_4_VC3_V8a_t45`, `LR2_11_DR2_22_VC3_V6a_t45`, `bnatt350`, and `bnatt400` feasible solutions were found only after unfixing variables. These results show that the aggressive fixing does not always work.

The percentage of free variables during local MIP search varies substantially according to the instance. On runs with feasible solutions, the percentage of free variables was $38 \pm 25\%$. One can observe that about 60% of the variables have been fixed. This substantial number was able to drive the search to find a feasible solution with only a small amount of enumeration.

The number of cuts inserted during the local MIP search was very small. With the exception of five instances, the average number of cuts was 3 ± 2. We believe that the algorithm is able to learn relatively fast a good set of variables to fix avoiding constraint violation. Instance `shipsched` presented curious results: on average 3490 were inserted but a feasible solution was found in all runs during the local MIP search.

5.3. Is the evolutionary phase really necessary?

A natural question in this framework is whether the evolutionary component is really necessary, or should we just feed the local MIP phase with random vectors. A secondary question is whether the feasibility pump cycles are necessary using either vectors from the BRKGA or random vectors. We addressed these questions by performing a set of experiments turning the evolutionary process off. In the first set, random vectors $v^* \in [0, 1]^n$ are generated and constraint propagation rounding is applied to generate the roundings $\tilde{v} \in \{0, 1\}^n$ which form the set $\mathcal{P}'$ used in local MIP search (Algorithm 3). We call this plain approach $\texttt{Rand}_{\texttt{pl}}$. In the second experiment, the vectors v^* are also randomly generated but we apply the feasibility pump to them through Algorithm 2. We call this approach $\texttt{Rand}_{\texttt{fp}}$. Note that, in each iteration of the general Algorithm 1, a new $\mathcal{P}'$ is generated which means that new v^*'s are generated too. In these two cases, the stopping criterion is the maximum time allowed.

In the third experiment, $\mathcal{P}'$ is built from LP relaxations. In each relaxation, one binary variable is fixed to zero or to one and the others are free, similar to lines 4–6 of Algorithm 1. Therefore, $\mathcal{P}'$ contains LP-feasible solutions with, at least, one integral variable. As before, the plain approach just provides $\mathcal{P}'$ directly to the local MIP search and is called $\texttt{Relax}_{\texttt{pl}}$. The $\texttt{Relax}_{\texttt{fp}}$ approach applies feasibility pump cycles to $\mathcal{P}'$ before providing it to the local MIP search. The stopping criteria are either the maximum time, or the maximum number of relaxations which is $2n$. We must emphasize that the evolutionary mechanism of the BRKGA was turned off in $\texttt{Rand}_{\texttt{pl}}$, $\texttt{Rand}_{\texttt{fp}}$, $\texttt{Relax}_{\texttt{pl}}$, and $\texttt{Relax}_{\texttt{fp}}$ although the decoder is used in $\texttt{Rand}_{\texttt{fp}}$ and $\texttt{Relax}_{\texttt{fp}}$. The parameter values of the previous experiments are used in this section.

Deactivating the evolutionary step produced results much worse than we expected: none of the four strategies were able to produce a feasible solution in ten independent runs. During the local MIP search, the number of free binary variables of $\texttt{Rand}_{\texttt{pl}}$ is relatively high with average of $45 \pm 14\%$ which leads to too many variables during the enumeration. Since $\mathcal{P}$ has no important information other than the roundings from constraint propagation, it does not result in good fixings. $\texttt{Relax}_{\texttt{pl}}$ contains rich information from the relaxations and leads to fewer free binary variables: $2 \pm 1\%$ on average. The situation is the opposite for $\texttt{Rand}_{\texttt{pl}}$: the small number of free variables does not provide enough degrees of freedom to the enumeration (even with unfixing level set to one). When the feasibility pump is used, the number of free variables is very small too. For both $\texttt{Rand}_{\texttt{fp}}$ and $\texttt{Relax}_{\texttt{fp}}$, the average number of free variables is less than 1%.

Table 2
Instances for which no feasible solutions were found in 5 hours.

LR1_1_DR1_4_VC3_V12a_t60	LR2_22_DR2_22_VC3_V10a_t45
LR1_1_DR1_4_VC3_V8a_t60	LR2_22_DR2_22_VC3_V10a_t60
LR1_2_DR1_3_VC2_V6a_t60	LR2_22_DR3_333_VC4_V14a_t45
LR1_2_DR1_3_VC3_V8a_t60	LR2_22_DR3_333_VC4_V14a_t60
LR2_11_DR2_22_VC3_V6a_t60	LR2_22_DR3_333_VC4_V17a_t45
LR2_11_DR2_33_VC4_V11a_t60	LR2_22_DR3_333_VC4_V17a_t60
LR2_11_DR2_33_VC5_V12a_t45	hanoi5
LR2_11_DR2_33_VC5_V12a_t60	

5.4. Comparison with other strategies

To assess the quality of the proposed algorithm (BRKGA-FP), we compared it with the results obtained from CPLEX and the feasibility pump 2.0 (FP 2.0). For 207 instances (80.86%), both BRKGA-FP, FP 2.0, and CPLEX found feasible solutions in all runs. For the 15 instances (5.86%) listed in Table 2, none of the algorithms found feasible solutions in 5 hours. Note that the unique instance from MIBLIB in this table is `hanoi5`.

Table 3 shows the results for the remaining instances for which a feasible solution was found in, at least, one run for any of the algorithms. For presentation purposes, we omitted instances for which either the algorithms found a feasible solution in all runs or they have not found a feasible solution in any run. The first column shows the name of the instance. The next three columns depicts the percentage of runs in which a feasible solution was found. Note that we have highlighted in bold the best results. The last three columns shows the geometric mean and maximum time (in seconds) to reach a feasible solution. Times for unsuccessful runs were disregarded.

On average, BRKGA-FP is able to find feasible solutions more frequently than CPLEX and FP 2.0. BRKGA-FP found feasible solutions for 93.35% of the instances while CPLEX and FP 2.0 succeeded for 91.02% and 89.45% of the instances, respectively. These proportions are similar to the percentage of runs resulting in feasible solutions, as shown in the last line of Table 3. For only five instances (`LR1_1_DR1_3_VC1_V7a_t60`, `LR1_1_DR1_4_VC3_V12b_t60`, `bnatt350`, `bnatt400`, and `circ10-3`), CPLEX was able to obtain more feasible solutions than BRKGA-FP. FP 2.0 and CPLEX obtained similar results: in six instances FP 2.0 obtained more feasible solutions than CPLEX, and for other eight instances CPLEX obtained more feasible solutions than FP 2.0.

BRKGA-FP obtained better results on MIRPLIB instances when compared with the other strategies. For eight instances, BRKGA-FP resulted in more feasible solutions than CPLEX and FP 2.0. CPLEX was able to beat the other strategies only in one instance, while FP 2.0 was not superior in any case to the other algorithms. For MIPLIB instances, both BRKGA-FP and CPLEX obtained more feasible solutions in two instances each, while FP 2.0 was superior in one instance. Note that for six instances (marked with *), only BRKGA-FP was able to find feasible solutions. However, BRKGA-FP did find any feasible solution for the `circ10-3` while CPLEX and FP 2.0 found feasible solutions in all runs. Neither BRKGA-FP or FP 2.0 have produced feasible solutions for `LR1_1_DR1_4_VC3_V12b_t60`, and CPLEX found a feasible solution in just one run for the same instance.

Considering all instances for which a feasible solution was found, CPLEX and FP 2.0 had similar running times and the Mann–Whitney–Wilcoxon U test shows that we cannot reject the hypoth-

Table 3
Comparison among algorithms for instances for which a feasible solution was found. The geometric average and maximum times are given in seconds.

Instance	% of feasible runs			Geometric Avg./Maximum time (seconds)		
	CPLEX	FP 2.0	BRKGA-FP	CPLEX	FP 2.0	BRKGA-FP
LR1_1_DR1_3_VC1_V7a_t60	**100**	0	80	279/766	–	516/669
LR1_1_DR1_4_VC3_V11a_t45	**100**	80	**100**	399/843	2897/3425	439/727
LR1_1_DR1_4_VC3_V11a_t60	60	0	**90**	1982/3405	–	1871/2398
LR1_1_DR1_4_VC3_V12a_t45*	0	0	**100**	–	–	1304/2942
LR1_1_DR1_4_VC3_V12b_t45	**100**	0	**100**	642/1812	–	811/1653
LR1_1_DR1_4_VC3_V12b_t60	**10**	0	0	3489/3489	–	–
LR1_1_DR1_4_VC3_V8a_t45	0	40	**80**	–	2513/3522	927/1958
LR1_1_DR1_4_VC3_V9a_t45	60	**100**	**100**	918/2725	2093/2365	237/483
LR1_1_DR1_4_VC3_V9a_t60*	0	0	**90**	–	–	1497/2582
LR1_2_DR1_3_VC2_V6a_t45	90	0	**100**	296/1681	–	238/556
LR1_2_DR1_3_VC3_V8a_t45*	0	0	**90**	–	–	1161/2227
LR2_11_DR2_22_VC3_V6a_t45*	0	0	**30**	–	–	4402/11002
LR2_11_DR2_33_VC4_V11a_t45*	0	0	**50**	–	–	10,084/17,018
atm20-100*	0	0	**100**	–	–	403/1141
bnatt350	**70**	30	10	1437/3524	1035/1316	2046/2046
bnatt400	20	**30**	10	3600/3600	1882/2516	2693/2693
circ10-3	**100**	**100**	0	3244/3600	3178/4386	–
neos-1429212*	0	0	**10**	–	–	12,166/12,166
neos-849702	20	**90**	**90**	1073/2932	443/1384	674/1465
netdiversion	90	**100**	**100**	694/3600	63/79	5308/7636
ns894786	10	**100**	**100**	3600/3600	139/227	2286/4158
shipsched	**100**	10	**100**	520/1167	3647/3647	591/1500
van	90	**100**	**100**	2739/3601	125/142	26/57
Average for all runs (all inst.)	88.79	88.20	**91.52**			

esis that both run-time distributions are similar at 95% confidence level (p-value > 0.05). BRKGA-FP was significantly slower than the other strategies (at 95% confidence level). It is expected that BRKGA would have a larger running time due to the several relaxations solved during the evolutionary phase. However, it is interesting to note that, if we consider only the results in Table 3, BRKGA-FP is significantly faster than FP 2.0 and has similar running times to CPLEX according to the U test (p-value < 0.05). Since those instances can be considered challenge instances, we believe that the time used during the evolutionary phase is compensated by the high quality fixing that it provides to the other steps, thus allowing small enumerations.

6. Conclusions

We proposed a framework that helps to find feasible solutions for mixed integer problems for which finding such solutions is very hard. The proposed algorithm uses information from several applications of the feasibility pump starting from different fractional (and possible infeasible) solutions, to identify variables to fix and, consequently reduce the size of the problem and speed up the search. The algorithms consists of three phases: the evolutionary phase using a modified version of the feasible pump heuristic; the local MIP search phase that uses information from the evolutionary phase to fix variables, add cuts, and perform enumeration; and the fixing phase responsible for fixing variables and reducing the size of the problem for the evolutionary phase. The experimental results on a hard class of MIP instances indicate that the proposed approach is able to find more feasible solutions than the original feasibility pump and the commercial solver IBM ILOG CPLEX.

One of the key features of the proposed framework is its ability to learn the values of a subset of variables in a feasible integer solution that, when fixed to their respective values, not only reduce the size of the problem, but also allow better enumeration steps. This is possible due to the population of roundings that are evolved during the time, from where the fixing is taken through a combination of several roundings. Because our approach is independent of specific problem characteristics or structure, it is applicable to any binary mixed integer program. Furthermore the extension to general integer variables is straightforward.

To the best of our knowledge, this is the first work to combine the local search offered by the feasibility pump with evolutionary algorithms, and to use this information to create high quality variable fixing. A nice addition to this framework would be to gather structural information of the problem and use it to improve the evolutionary process and the variable fixing.

Disclaimer

This work was done when the first author was a postdoctoral fellow at the H. Milton Stewart School of Industrial & Systems Engineering (ISyE) at Georgia Institute of Technology. The information presented is not based on nor imply actual business cases for AT&T, ExxonMobil, and associated companies.

Acknowledgments

The authors wish to thank Matteo Fischetti and Domenico Salvagnin for kindly providing the source code of feasibility pump 2.0. They also thank Lluis Miquel Munguia and Natashia Boland for enriching discussions, and the anonymous reviewers for providing comments that improved this paper. This research was support by ExxonMobil Upstream Research Company and by AFOSR [Grant no. FA9550-12-1-0151, “New Approaches for Very Large-Scale Integer Programming”].

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2017.05.003

References

Achterberg, T., & Berthold, T. (2007). Improving the feasibility pump. *Discrete Optimization, 4*(1), 77–86. doi:10.1016/j.disopt.2006.10.004.

Andrade, C. E., Miyazawa, F. K., & Resende, M. G. C. (2013). Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem.

In *Proceedings of the 15th annual conference on genetic and evolutionary computation GECCO'13* (pp. 463–470). New York, NY, USA: ACM. doi:10.1145/2463372.2463434.

Andrade, C. E., Resende, M. G. C., Karloff, H. J., & Miyazawa, F. K. (2014). Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the 16th conference on genetic and evolutionary computation GECCO'14* (pp. 405–412). New York, NY, USA: ACM. doi:10.1145/2576768.2598284.

Andrade, C. E., Resende, M. G. C., Zhang, W., Sinha, R. K., Reichmann, K. C., Doverspike, R. D., & Miyazawa, F. K. (2015a). A biased random-key genetic algorithm for wireless backhaul network design. *Applied Soft Computing, 33*, 150–169. doi:10.1016/j.asoc.2015.04.016.

Andrade, C. E., Toso, R. F., Resende, M. G. C., & Miyazawa, F. K. (2015b). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary Computation, 23*, 279–307. doi:10.1162/EVCO_a_00138.

Baena, D., & Castro, J. (2011). Using the analytic center in the feasibility pump. *Operations Research Letters, 39*(5), 310–317. doi:10.1016/j.orl.2011.07.005.

Bertacco, L., Fischetti, M., & Lodi, A. (2007). A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization, 4*(1), 63–76. doi:10.1016/j.disopt.2006.10.001.

Berthold, T., & Hendel, G. (2015). Shift-and-Propagate. *Journal of Heuristics, 21*(1), 73–106. doi:10.1007/s10732-014-9271-0.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Springer Berlin Heidelberg.

Boland, N. L., Eberhard, A. C., Engineer, F. C., Fischetti, M., Savelsbergh, M. W. P., & Tsoukalas, A. (2014). Boosting the feasibility pump. *Mathematical Programming Computation, 6*(3), 255–279. doi:10.1007/s12532-014-0068-9.

Caserta, M., & Reiners, T. (2016). A pool-based pattern generation algorithm for logical analysis of data with automatic fine-tuning. *European Journal of Operational Research, 248*(2), 593–606. doi:10.1016/j.ejor.2015.05.078.

Ericsson, M., Resende, M. G. C., & Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization, 6*(3), 299–333. doi:10.1023/A:1014852026591.

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming, 104*(1), 91–104. doi:10.1007/s10107-004-0570-3.

Fischetti, M., & Salvagnin, D. (2009). Feasibility pump 2.0. *Mathematical Programming Computation, 1*(2–3), 201–222. doi:10.1007/s12532-009-0007-3.

Glover, F., & Laguna, M. (1997a). General purpose heuristics for integer programming - Part I. *Journal of Heuristics, 2*(4), 343–358. doi:10.1007/BF00132504.

Glover, F., & Laguna, M. (1997b). General purpose heuristics for integer programming - Part II. *Journal of Heuristics, 3*(2), 161–179. doi:10.1007/BF00132504.

Gonçalves, J. F., & de Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics, 8*(6), 629–642. doi:10.1023/A:1020377910258.

Gonçalves, J. F., & Resende, M. G. C. (2011a). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*, 487–525. doi:10.1007/s10732-010-9143-1.

Gonçalves, J. F., & Resende, M. G. C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization, 22*(2), 180–201. doi:10.1007/s10878-009-9282-1.

Hanafi, S., Lazić, J., & Mladenović, N. (2010). Variable neighborhood pump heuristic for 0–1 mixed integer programming feasibility. *Electronic Notes in Discrete Mathematics, 36*, 759–766. doi:10.1016/j.endm.2010.05.096.

Hansen, P., Mladenović, N., & Urošević, D. (2006). Variable neighborhood search and local branching. *Computers & OR, 33*(10), 3034–3045. doi:10.1016/j.cor.2005.02.033.

Li, Y., Ergun, O., & Nemhauser, G. L. (2015). A dual heuristic for mixed integer programming. *Operations Research Letters, 43*(4), 411–417. doi:10.1016/j.orl.2015.05.007.

Papageorgiou, D. J., Nemhauser, G. L., Sokol, J., Cheon, M.-S., & Keha, A. B. (2014). MIRPLib – a library of maritime inventory routing problem instances: Survey, core model, and benchmark results. *European Journal of Operational Research, 235*(2), 350–366. doi:10.1016/j.ejor.2013.12.013.

Santis, M. D., Lucidi, S., & Rinaldi, F. (2014). Feasibility pump-like heuristics for mixed integer problems. *Discrete Applied Mathematics, 165*(0), 152–167. doi:10.1016/j.dam.2013.06.018.

Stefanello, F., Aggarwal, V., Buriol, L. S., Gonçalves, J. F., & Resende, M. G. C. (2015). A biased random-key genetic algorithm for placement of virtual machines across geo-separated data centers. In *Proceedings of the 2015 on genetic and evolutionary computation conference GECCO '15* (pp. 919–926). New York, NY, USA: ACM. doi:10.1145/2739480.2754768.