Feasibility Pump Heuristics for Column Generation Approaches

Pierre Pesneau, Ruslan Sadykov, and François Vanderbeck

Univ. of Bordeaux, INRIA team Realopt

Abstract. Primal heuristics have become an essential component in mixed integer programming (MIP). Generic heuristic paradigms of the literature remain to be extended to the context of a column generation solution approach. As the Dantzig-Wolfe reformulation is typically tighter than the original compact formulation, techniques based on rounding its linear programming solution have better chance to yield good primal solutions. However, the dynamic generation of variables requires specific adaptation of heuristic paradigms. We focus here on "feasibility pump" approaches. We show how such methods can be implemented in a context of dynamically defined variables, and we report on numerically testing "feasibility pump" for cutting stock and generalized assignment problems.

Keywords: Primal Heuristic, Dantzig-Wolfe Decomposition, Branch-and-Price Algorithms.

1 Introduction

Heuristics are algorithms that attempt to derive "good" primal feasible solutions to a combinatorial optimization problem. They include constructive methods that build a solution and iterative improvement methods such as local search procedure that starts with an incumbent. The term "primal heuristic" generally refers to methods based on the tools of exact optimization, truncating an exact procedure or constructing solutions from the relaxation on which the exact approach relies: techniques range from greedy constructive procedures to rounding a solution of the linear programming (LP) relaxation, using the LP solution to define a target, or simply exploiting dual information to price choices. Today's MIP solvers rely heavily on generic primal heuristics: progress in primal heuristics is quoted as one of the main source of commercial solver performance enhancement in the last decade [6]. High quality primal values help pruning the enumeration by bound and by preprocessing (constraint propagation). They are also essential in tackling large scale real-life applications where the exact solver is given limited running time and a realistic ambition is to obtain good primal solutions.

Heuristics based on exact methods have found a new breath in the recent literature. The latest developments are reviewed in [5,12]. Here we focus on the so-called *feasibility pump algorithm* [1,11]. The method entails rounding the solution of the Linear Programming (LP) relaxation to the closest integer. It

R. Klasing (Ed.): SEA 2012, LNCS 7276, pp. 332–343, 2012.

might lead to an infeasible integer solution. But it defines a target for integer optimization, i.e., the LP is re-optimized with the objective of minimizing both the distance to that target and the original objective value. And the process iterates. Recently this paradigm has been extended from the context of binary integer programs to general integer programs [4,13]. Our purpose is to examine possible extensions of feasibility pump to the case where one works with a column generation approach for the Dantzig-Wolfe reformulation of the problem.

The column generation literature reports many application specific studies where primal heuristics are a key to success: some heuristics have been implemented either by taking decision in the space of the original compact formulation (using a so-called robust approach), others involve decision directly in the space of the Dantzig-Wolfe reformulation. Here, we focus on the latter direct approach that fully exploits the specificity of Dantzig-Wolfe decomposition: the price coordination mechanism that brings a global view and the more aggregate decisions that enable rapid progress in building a primal solution (fixing variables of the Dantzig-Wolfe reformulation has a much stronger impact than when fixing variables of the original formulation). Our aim is to extract generic methods for use in branch-and-price algorithms.

Making heuristic decision on the variables of the Dantzig-Wolfe (DW) reformulation requires particular attention to derive heuristics that are "compatible" with the pricing problem solver as highligthed herein. Otherwise, it may impair the effectiveness of the column generation approach that relied on the existence of an efficient pricing oracle. Hence, the generic primal heuristic paradigm must be adapted if one works with the column generation formulation. Alternatively, if one makes heuristic decision on the variables of the original compact formulation, then the generic primal heuristics for mixed integer programming apply directly. However, projecting the master solution in the original variable space is not always straightforward. First, the projection is not defined in the common case of identical subsystems. Second, because a modified original formulation (as induced by the primal heuristic) calls for a modified DW reformulation with possibly the same issues as when implementing the heuristic in the DW reformulation.

The rest of the paper is organized as follows. In Section 2, we review standard primal heuristics techniques based on mathematical programming and more specifically the feasibility pump heuristic. We then discuss in Section 3 the specific difficulties in implementing primal heuristics in a column generation context and we propose a simple strategy to get around these technical issues. In Section 4, we develop a general purpose feasibility pump heuristic for use in a column generation approach. This generic method is evaluated experimentally as reported in Section 5. In our conclusion, we analyze the results and discuss further work.

2 Primal Heuristics for MIP and Feasibility Pump

Assume a bounded integer program with n variables:

$$z := \min\{c\,x : A\,x \geq a,\ x \in I\!N^n\}\ .$$

Let x^{LP} be a solution to its Linear Programming (LP) relaxation, and let z^{LP} be its value. A "rounding heuristic" consists in iteratively selecting a fractional component of x^{LP} that must take integer value and fix it to an integer value or simply reset the lower (resp. upper) bound on this component. The method is defined by a rule for selecting the component and its fixed value or bound (such as *least fractional* or *guided search* among the rules reviewed in [5]). "Diving heuristics" differ from simple rounding heuristics by the fact that the LP is re-optimized after bounding or fixing one (or several) variables. Diving can be seen as a heuristic search in a LP-based branch-and-bound tree: the search plunges depth into the enumeration tree by selecting a branch heuristically at each node.

The *feasibility pump* heuristic can be seen as specific iterative rounding procedure [1,11]. It was originally developed for a binary integer program:

$$\min\{c\,x : A\,x \geq a,\ x \in \{0,1\}^n\}\ .$$

Feasibility pump entails exploring a sequence of trial solutions, $\tilde{x}$, obtained by rounding to the closed integer solution to the LP solution, x^{LP}, of a program with modified objective function. If the rounded trial integer solution is feasible the algorithm stops with this primal candidate. Otherwise, the rounded solution serves as a target to which one aims to minimize some distance measure. Assuming that the new objective function combines the original objective with the L_1 norm to the target solution, the modified problem takes the form:

$$\min\{c\,x + \epsilon(\sum_{j\in J^0} x_j + \sum_{j\in J^1}(1-x_j)) : A\,x \geq a,\ x \in [0,1]^n\}\ . \tag{1}$$

The index sets $J^0 \subset J$ and $J^1 \subset J$ form a partition of $J = \{1,\ldots,n\}$. They define respectively the components that take value 0 or 1 in the previous trial integer solution, $\tilde{x}$, obtained through rounding the LP solution of (1). Parameter ϵ controls the impact of the cost modifications relative to the original objective. Extending feasibility pump to a general integer program requires an adapted distance measure [4,13]. Assuming bounded integer variables $x_j \in I\!N$, with $l_j \leq x_j \leq u_j,\ j \in J$, the modified problem could take the form:

$$\min\{c\,x + \epsilon(\sum_{j:\tilde{x}_j=l_j}(x_j - l_j) + \sum_{j:\tilde{x}_j=u_j}(u_j - x_j) + \sum_{j:l_j<\tilde{x}_j<u_j} d_j) : A\,x \geq a, \tag{2}$$

$$d_j - \tilde{x}_j \leq x_j \leq d_j + \tilde{x}_j\ \forall j,\ x \in I\!R^n\ \} \tag{3}$$

where (3) are additional constraints needed to define auxiliary variables d_j.

3 Primal Heuristics Combined with Column Generation

In the context of a column generation approach, we assume a mixed integer program with decomposable structure (a subset of constraints is assumed to have a *block diagonal* structure):

$$[P] \quad \min\{cx : Ay \geq a, y = \sum_k x^k, \underbrace{B^k x^k \geq b^k\ \forall k,\ x^k \in I\!N^n}_{x^k \in X^k}\} \tag{4}$$

where $Ay \geq a$ represent "complicating constraints", while optimization over subsystems X^k defined by subsystem $B^k x^k \geq b^k$ is "relatively easy" (i.e., optimization over X^k is assumed to be tractable). In the sequel, to simplify the presentation, subsystems X^k are assumed to be bounded pure integer programs and G^k is an enumeration of the feasible solutions to X^k, i.e., $X^k = \{x^g\}_{g \in G^k}$. The structure of $[P]$ can be exploited to reformulate it as the master program:

$$\min\{ \sum_{k, g \in G^k} (cx^g)\lambda_g^k : \sum_{k, g \in G^k} (Ax^g)\lambda_g^k \geq a;\ \sum_{g \in G^k} \lambda_g^k = 1\ \forall k;\ \lambda_g^k \in \{0,1\}\ \forall g, k\}\ . \tag{5}$$

When each block is identical, as is the case in many applications, let $Bx \geq b$ be the constraint set defining one block subsystem $X^k = X = \{Bx \geq b, x \in I\!N^n\}\ \forall k$, and the master takes the form:

$$[\mathrm{M}] \equiv \min\{\sum_{g \in G} (cx^g)\lambda_g : \sum_{g \in G} (Ax^g)\lambda_g \geq a;\ \sum_{g \in G} \lambda_g = K;\ \lambda_g \in I\!N\ \forall g\} \tag{6}$$

where $\lambda_g = \sum_k \lambda_g^k$, G is the set of *generators* of X, and K is the number of identical blocks. In the sequel, we assume identical subproblems and hence master formulation (6) unless specified otherwise. The master program $[M]$ is solved by Branch-and-Price: at each node of the Branch-and-Bound tree the linear relaxation of $[M]$ is solved by column generation. The reduced cost of a column $g \in G$ takes the form $(c - \pi A)\, x^g - \sigma$, where (π, σ) are the dual solution associated with constraints $Ax \geq a$ in $[M]$ and $\sum_{g \in G} \lambda_g = K$ respectively. Thus, the pricing problem is of the form: $\min\{(c - \pi A)\, x\ :\ Bx \geq b, x \in I\!N^n\}$.

The most commonly used generic primal heuristic in this column generation context is the so-called *restricted master heuristic*. The column generation formulation is restricted to a subset of generators $\overline{G}$ and associated variables, and it is solved as a static IP. The main drawback of this approach is that the resulting restricted master integer problem is often infeasible (the columns of $\overline{G}$ – typically defined by the LP solution – may not be combined to an integer solution). In an application specific context, an ad-hoc recourse can be designed to "repair" infeasibility. Such implementation has been developed for network design [8], vehicle routing [2,9,20] and delivery [19] problems. Alternatively, one can use the master LP solution as a base for column selection giving rise to a *rounding heuristics*. Such heuristics (sometimes coupled with local search) have been successfully applied [3,7,10,15] (on cutting stock, planning, and vehicle routing problems). However, reaching feasibility remains a difficult issue that is often handled in an application specific manner. Diving heuristics are generic ways of 'repairing" infeasibility as we highlight it in [14].

Deriving primal heuristics based on rounding the Dantzig-Wolfe reformulation LP solution raises some difficulties. Bounding a master variable or modifying its cost can be incompatible with the column generation approach in the sense that it can induce modifications to the subproblem that are not amenable to the pricing oracle. Depending on the application, the oracle may or may not still be applicable after some modifications to the problem structure. In the context

of this paper, our aim is to derive a generic algorithm that applies without any required modifications to the pricing subproblem formulation. To achieve this goal, we restrict our problem modifications to operations that are easy-to-implement in the master and do not induce subproblem modifications: namely master variable lower bound setting and cost reduction.

Indeed, setting an upper bound on an existing column, i.e., enforcing $\lambda_g \leq u_g$, or a slightly more general constraint of the form (3) in the master, yields an associated dual variable that must be accounted for in pricing (by modeling an extra cost for a specific solution x^g). Alternatively, one must restrict the pricing problem to avoid regenerating x^g; indeed, if $\lambda_g \leq u_g$ is ignored, the column x^g might otherwise be wrongly regenerated as the best price solution. Both approaches induce significant modifications to the pricing procedure. However, setting a lower bound on an existing column of the form $\lambda_g \geq l_g$ is trivial: this constraint can safely be ignored when pricing. Indeed, ignoring the dual price "reward" for generating this column, means that the pricing oracle overestimates its reduced cost and might not generate it; but the column needs not be generated since it is already in the master. Similarly, we cannot increase the cost c_g of a variable λ_g beyond its true cost, because then λ_g will price out negatively according to the original pricing oracle, and hence it can be regenerated as the best subproblem solution, unless we modify the pricing problem to avoid this. On the other hand, decreasing the cost c_g of a variable λ_g is amenable to the unmodified column generation scheme, as the pricing oracle shall simply overestimate the reduced cost of such already included column.

An alternative approach is to perform the feasibility pump problem modifications in the space of the original formulation (4). Having solved the LP relaxation of (5), one can project its solution in the original space, defining

$$x^k = \sum_{g \in G^k} x^g \lambda_g^k \,, \tag{7}$$

and apply the primal heuristic procedures of Section 2 on this projected solution. There remains a key issue however. The projection is not uniquely defined in the case where there are K identical subproblems. Then, the LP solution of the aggregate master (6) can be disaggregated using for instance:

$$\lambda_g^k = \min\{1, \lambda_g - \sum_{\kappa=1}^{k-1} \lambda_g^k, (k - \sum_{\gamma \prec g} \lambda_\gamma)^+\} \quad \forall k = 1, \ldots, K, g \in G \,, \tag{8}$$

where $\prec$ defines a lexicographic ordering of columns $g \in G$ (see [23] for details). With such disaggregated λ_g^k values, one can use projection (7), but the reverse relation cannot be properly enforced, i.e., a restriction on x_j^k variables cannot be modeled in (6).

It is also important to observe that acting on the variables of the original formulation is a completely different decision space than acting on the DW reformulation variables. In particular, variable bounding or cost modification decisions have a more macroscopic effect when done in the DW reformulation. This

can be both an advantage (faster progress to a integer solution) and a drawback (of quickly painting yourself in a corner). This study focus on primal heuristics for the DW reformulation because in many applications making more aggregate fixing is more an advantage than a drawback and secondly because that is where the need for innovation lies (primal heuristics in the original formulation can implemented as defined in the standard paradigm for MIPs).

From the above discussion, we conclude that to implement the feasibility pump paradigm, we shall restrict the method to using lower bound setting and cost reduction. Our algorithm shall combine rounding and diving paradigm: defining lower bound on master variables is implemented by defining a partial solution. Indeed, rounding down variable λ_g amounts to taking $\lfloor \lambda_g \rfloor$ copies of this column in the partial solution. The *residual master problem* that remains once the partial solution, denoted $\tilde{\lambda}$, is extracted takes the form:

$$[RM] \equiv \min\{\sum_{g\in G}(cx^g)\lambda_g : \sum_{g\in G}(Ax^g)\lambda_g \geq \tilde{a};\ \sum_{g\in G}\lambda_g = \tilde{K};\ \lambda_g \in I\!N \quad \forall g\} \quad (9)$$

where $\tilde{a} = a - \sum_g (Ax^g)\tilde{\lambda}_g$ and $\tilde{K} = K - \sum_g \tilde{\lambda}_g$. The columns that are part of the partial solution remain in the residual problem for further selection if suitable. Thus, the master variable lower bounds are implemented implicitly by the definition of the partial solution.

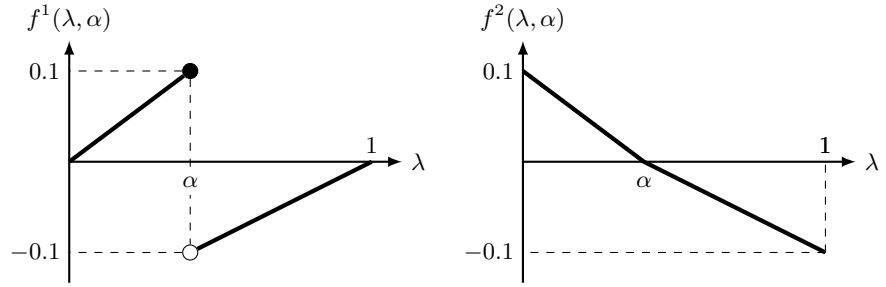
A key feature in this primal heuristic is preprocessing: it is important to "cleanup" the residual problem after fixing a partial solution, deleting all columns that could not be part of an integer solution to the residual problem (and hence would lead to infeasibility if selected). Thus, preprocessing helps to avoid the primal heuristic dead-ending with a unfeasible solution. In this context, so-called "proper columns" are columns that could take integer value in an optimal solution to the residual master problem [22]. If the oracle assumes a bounded subproblem, one can tighten lower and upper bounds on subproblem variables to generate proper columns. In that case, we refine the bounds on subproblem variables by constraint propagation after fixing a partial master solution.

Observe that, as the *residual master problem* (9) that remains after a rounding operation might be modified by preprocessing, its re-optimization might not necessarily be trivial and it can lead to generating new columns. This mechanism yields the "missing" complementary columns to build feasible solutions. If the residual master is however infeasible for a given partial solution, re-optimization can be a way to prove it early through a Simplex phase 1 and/or preprocessing. Re-optimization of the master LP after fixing, however important feature for the success of the approach, can be time consuming. Tuning the level of approximation in this re-optimization allows one to control the computational effort.

4 A Generic Feasibility-Pump Algorithm

In Table 1, we propose a generic heuristic for use in a column generation context that exploits the main ideas of the feasibility pump paradigm. A target solution $\tilde{\lambda}$ is defined by rounding each component of the LP solution of the master (6) to the

closest integer. If $\tilde{\lambda}$ defines a feasible solution to (6), we stop. Otherwise we use it as a target point. To induce a new master LP solution "closer" to target solution $\tilde{\lambda}$, we decrease the cost of columns that were rounded-up to define $\tilde{\lambda}$ and increase the cost of those that were rounded down. However, we do not increase column costs beyond their original value; otherwise, this would induce the regeneration of the same column at its initial cost. The two cost modification factor functions that we considered are presented in Figure 1. The first one is a direct adaptation of the cost modification arising in (1). The second is a complementary variant that aims at stabilizing the part of the solution that is currently integer, using the cost modification to make the current integer solution even more attractive.



$$f^1(\lambda,\alpha) = \begin{cases} 0.1\ \frac{\lambda}{\alpha} & \text{if } \lambda \le \alpha \\ -0.1\ \frac{(1-\lambda)}{(1-\alpha)} & \text{if } \lambda > \alpha \end{cases} \qquad f^2(\lambda,\alpha) = \begin{cases} 0.1\ (1-\frac{\lambda}{\alpha}) & \text{if } \lambda \le \alpha \\ -0.1\ \frac{(\lambda-\alpha)}{(1-\alpha)} & \text{if } \lambda > \alpha \end{cases}$$

Fig. 1. Two cost modification factor functions of the form $f(\lambda,\alpha)$ where $\lambda \in [0,1]$ is the factional part of a column value in the master LP solution and $\alpha \in (0,1)$ is a fractionality threshold parameter

At iteration t, the modified master program becomes:

$$[M^t] \equiv \min\{ \sum_{g\in G^t} c_g^t \lambda_g : \sum_{g\in G^t} (Ax^g)\lambda_g \ge a^t;\ \sum_{g\in G^t} \lambda_g = K^t;\ \lambda_g \in I\!N \quad \forall g \in G^t\} \tag{10}$$

where G^t, a^t, K^t, and c_g^t are updated in the course of the algorithm. Initially, G^0 is the set of columns generated in the course of solving the linear relaxation of the master program, [M] given in (6), by column generation, $a^0 = a$, $K^0 = K$, and $c_g^0 = cx^g$ for all $g \in G^0$. Then, at iteration t, we compute the LP solution λ^t to (10), and its rounded value $\tilde{\lambda}$. If the later is not feasible for (6), we iterate the feasibility pump procedure. In our implementation, we define an initial partial solution by rounding down the LP solution to the master (6) before going into the feasibility pump heuristic (see *Step 2* in Table 1). In this way, our residual master program is close to a 0 - 1 problem and we omit the subtleties required to handle general integer problems. Finally, observe that the algorithm cycles with the same master LP solution if no columns are rounded up in the target solution and hence no column cost are decreased. To avoid this situation, we then

Table 1. Feasibility Pump heuristic. Let G^f denote the current set of fractional value columns, i.e., $G^f = \{g \in G^t : \lfloor \lambda_g^t \rfloor < \lambda_g^t < \lceil \lambda_g^t \rceil\})$, while $\tilde{\lambda}_g$ denotes the rounded value of λ_g^t, which can be obtained by rounded up or down the current LP value. $\hat{\lambda}$ denotes the current partial solution, while $f(.)$ is one of the function of Figure 1.

Step 1: Solve the LP relaxation of the master (6) by column generation. Set the iteration counter $t \leftarrow 0$. Initialize the column set G^0 to the columns generated in solving the master LP, the right-hand-sides to $a^0 \leftarrow a$ and $K^0 \leftarrow K$. Let λ^0 be the master LP solution. Initialize the master column current costs, $c_g^0 \leftarrow c_g$, and the current partial solution, $\hat{\lambda}_g \leftarrow 0$, for all $g \in G^0$. The fractionality threshold is initially set to $\alpha \leftarrow 0.5$.

Step 2: Fix a partial IP solution: $\hat{\lambda}_g \leftarrow \hat{\lambda}_g + \lfloor \lambda_g^t \rfloor$ for all $g \in G^t$. Reset $\tilde{\lambda} \leftarrow 0$; then set $\tilde{\lambda}_g \leftarrow \lfloor \lambda_g^t \rfloor$ and reset $\lambda_g^t \leftarrow \lambda_g^t - \lfloor \lambda_g^t \rfloor$.

Step 3: Update the master constraint right-hand-sides: $a^t \leftarrow a^t - \sum_{g \in G^t} Ax^g \tilde{\lambda}_g$, $K^t \leftarrow K^t - \sum_{g \in G^t} \tilde{\lambda}_g$. Apply preprocessing to the master, deleting non-proper columns from G^t and possibly update the pricing problems (possibly setting new bounds on subproblem variables in one aims to generate only proper columns). Using column generation, re-optimize the linear relaxation of the residual master program (10), possibly adding columns to G^t. If residual master problem is shown infeasible through preprocessing or Phase 1 of the Simplex algorithm, STOP. Else, reset λ^t to the current master LP solution and update G^f.

Step 4: Define the solution rounded to the closest integer: reset $\tilde{\lambda} \leftarrow 0$; then set $\tilde{\lambda}_g \leftarrow \lfloor \lambda_g^t \rfloor$ if $\lambda_g^t \leq \alpha$ and $\tilde{\lambda}_g = \lceil \lambda_g^t \rceil$ otherwise, for all $g \in G^t$.

Step 5: If no columns were rounded up, try to reset the fractionality threshold α. I.e., let $\alpha_{\min} = \min_{g \in G^f} \{\lambda_g^t - \lfloor \lambda_g^t \rfloor\}$ and $\alpha_{\max} = \max_{g \in G^f} \{\lambda_g^t - \lfloor \lambda_g^t \rfloor\}$. If $\alpha_{\min} < \alpha_{\max}$, decrease the fractionality threshold: $\alpha \leftarrow \frac{\alpha_{\min} + \alpha_{\max}}{2}$ and return to step 4. Otherwise, if $\alpha_{\min} = \alpha_{\max}$, apply a diversification step by fixing a column from $g \in G^f$ to value $\tilde{\lambda}_g = \lceil \lambda_g^t \rceil$ and go-to *Step 3*.

Step 6: If $(\hat{\lambda} + \tilde{\lambda})$ defines a complete primal solution, i.e., is a feasible integer solution to (6), record this solution and STOP.

Step 7: Define the updated column costs using rule: $c_g^{t+1} \leftarrow c_g^t$ for $g \in G^t \setminus G^f$ and

$$c_g^{t+1} \leftarrow \min\{c_g, f(\lambda_g^t - \lfloor \lambda_g^t \rfloor, \alpha)\, c_g^t\} \quad \forall g \in G^f \ .$$

Step 8: Let $t \leftarrow t + 1$ and re-optimize the linear relaxation of the residual master program (10) with modified cost using column generation; record its solution λ^t and update G^f.

Step 9: If the maximum number, T, of feasibility pump iterations has been reached, go-to *Step 2*.

Step 10: Go-to *Step 3*.

decrease the fractionality threshold parameter (see *Step 5* in Table 1), unless all fractional columns have the same fractionality. In the later case, we arbitrarily fix a column to its rounded up value to induce a diversification.

5 Computational Results

We built the above feasibility pump heuristic into *BaPCod* [21], a generic Branch-and-Price Code that we developed. We tested these procedures on standard

models: the Cutting Stock Problem (CSP) and the Generalized Assignment Problem (GAP). For each model, we present average computational results on random instances similar to those of the literature. We tried both cost modification factor functions presented in Figure 1. In the numerical result tables, column "found" gives the fraction of instances for which the feasibility pump heuristic found a feasible primal solution. Tables report either the number of instances solved to optimality or the "gap" computed as the difference between the solution found and the column generation lower bound in per cent from the latter. This statistic is the average among instances for which a feasible solution has been found. Column "time" gives the average heuristic execution time in seconds over all instances, including those for which the heuristic fails to find a primal solution (those require typically larger computing time). For comparison, we present the performance of the diving heuristic presented in [14]: after fixing the lower integer part of the master solution, this heuristic rounds one column value selected according to the least fractional criteria, re-optimize the residual master, and reiterates until either finding a feasible solution or reaching infeasibility.

The results for the Cutting Stock Problem (CSP) are given in Table 2 for instances with 50 and 100 items respectively. For the CSP, the *master linear program* is:

$$\min \left\{ \sum_{g \in G} \lambda_g : \sum_{g \in G} x_i^g \lambda_g \geq d_i \ \forall i; \quad \sum_{g \in G} \lambda_g \leq K; \quad \lambda_g \geq 0 \ \forall g \right\} \tag{11}$$

where G is the set of feasible solutions to the *pricing subproblem*:

$$\max \left\{ \sum_i \pi_i x_i : \sum_i w_i x_i \leq W, \quad x_i \in I\!N, \ i = 1, \ldots, n \right\}$$

We solve the latter using the integer knapsack solver of Pisinger [17]. Note that for this application, the residual master problem is always feasible as one can use as many patterns as needed. Fifty random instances are generated using uniform distributions: $W = 10000$, $w_i \in U[500, 2500]$, $d_i \in U[1, 50]$. For these tests, the primal solution is always equal to the root dual bound or one unit above.

In the Generalized Assignment Problem (GAP), one searches for a minimum cost assignment of a set of jobs indexed by j to a set of machines indexed by m with limited capacity. The master linear program is:

Table 2. Results for Cutting Stock instances

n	max d_i	function	found	opt	gap	time
50	50	f^1	50/50	45/50	0.05	6.14
50	50	f^2	50/50	41/50	0.09	4.82
50	50	Pure Div	50/50	43/50	0.07	1.17
100	50	f^1	50/50	43/50	0.04	23.93
100	50	f^2	50/50	40/50	0.05	17.98
100	50	Pure Div	50/50	35/50	0.08	4.08

$$\min \left\{ \sum_{m,g \in G^m} c_g \lambda_g : \sum_{m,g \in G^m} x_j^g \lambda_g = 1 \, \forall j \in J; \; \sum_{g \in G^m} \lambda_g \leq 1 \, \forall m \in M; \; \lambda_g \geq 0 \, \forall g \right\},$$

where G^m is the set of feasible assignments to machine m solving the $0-1$ knapsack problem:

$$\max \left\{ \sum_{j \in J} (\pi_j - c_j^m) \, x_j^m : \sum_{j \in J} p_{jm} x_j^m \leq w_m; \quad x_j^m \in \{0,1\} \, \forall j \in J \right\}.$$

We solve the latter using the code of Pisinger [16]. Random instances were generated in the same way as the hard instances of type D in [18]. The results are shown in Table 3.

Table 3. Average results for 50 instances of type D for the Generalized Assignment Problem

m	n	function	found	gap	time
10	50	Pur Div.	34/50	1.00%	0.37
10	50	f^1	36/50	0.98%	1.81
10	50	f^2	48/50	1.14%	0.81
20	100	Pur Div.	35/50	0.65%	2.46
20	100	f^1	36/50	0.55%	14.56
20	100	f^2	42/50	0.75%	5.92

6 Conclusion

Our study demonstrates that the feasibility pump primal heuristics paradigm can be successfully extended to a column generation context. The key to such extension is to restrict problem modifications to those that are compatible with the column generation procedure: our implementation relies only on a cost reduction mechanism and implicitly on setting lower bound on master variables. As in the case for diving heuristics, the variable fixing done in our implementation of the feasibility pump heuristic can lead to a dead-end with an unfeasible residual master program. Compared to our previous experience using a pure diving heuristic, as reported in [14], our numerical preliminary experiment shows that feasibility pump leads to more feasible primal solutions and relatively good solutions. The diversification mechanisms that we experimented for diving heuristics in [14] (i.e., a limited backtracking when the heuristic dead-ends) could be adapted for feasibility pump. In future work, we also intend to test the approach on a larger scope of applications and to compare feasibility pump implementations in the master program, versus implementing cost modification in the space of the compact formulation, when this is feasible, i.e., in case such as the GAP where there are no multiple identical subproblems.

References

1. Achterberg, T., Berthold, T.: Improving the feasibility pump. Discrete Optim. 4(1), 77–86 (2007)
2. Agarwal, Y., Mathur, K., Salkin, H.M.: A set-partitioning-based exact algorithm for the vehicle routing problem. Networks 19(7), 731–749 (1989)
3. Belov, G., Scheithauer, G.: A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. European J. Oper. Res. 141(2), 274–294 (2002)
4. Bertacco, L., Fischetti, M., Lodi, A.: A feasibility pump heuristic for general mixed-integer problems. Discrete Optimization 4(1), 63–76 (2007)
5. Berthold, T.: Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin (2006)
6. Bixby, B.: Presentation of the gurobi optimizer. In: Integer Programming Down Under: Theory, Algorithms and Applications, Workshop at Newcastle NSW (2011)
7. Ceselli, A., Righini, G., Salani, M.: A column generation algorithm for a vehicle routing problem with economies of scale and additional constraints. In: Proceedings TRISTAN, Phuket, Thailand (June 2007)
8. Chabrier, A.: Heuristic branch-and-price-and-cut to solve a network design problem. In: Proceedings CPAIOR, Montreal, Canada (May 2003)
9. Chabrier, A., Danna, E., Le Pape, C.: Coopération entre génération de colonnes et recherche locale appliquées au problème de routage de véhicules. In: Huitièmes Journées Nationales sur la résolution de Problèmes NP-Complets (JNPC), Nice, France, pp. 83–97 (May 2002)
10. Dobson, G.: Worst-case analysis of greedy heuristics for integer programming with nonnegative data. Math. Oper. Res. 7(4), 515–531 (1982)
11. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. 104(1, ser. A), 91–104 (2005)
12. Fischetti, M., Lodi, A.: Heuristics in Mixed Integer Programming. Wiley Encyclopedia of Operations Research and Management Science, J.J. Cochran Edt., vol. 3, pp. 2199–2204. Wiley (2011)
13. Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming Computation 1, 201–222 (2009)
14. Joncour, C., Michel, S., Sadykov, R., Sverdlov, D., Vanderbeck, F.: Column generation based primal heuristics. Electronic Notes in Discrete Mathematics 36, 695–702 (2010)
15. Perrot, N.: Integer Programming Column Generation Strategies for the Cutting Stock Problem and its Variants. PhD thesis, Université Bordeaux 1, France (2005)
16. Pisinger, D.: A minimal algorithm for the 0-1 knapsack problem. Operations Research 45(5), 758–767 (1997)
17. Pisinger, D.: A minimal algorithm for the bounded knapsack problem. INFORMS Journal on Computing 12(1), 75–82 (2000)
18. Savelsbergh, M.: A branch-and-price algorithm for the generalized assignment problem. Operations Research 45(6), 831–841 (1997)
19. Schmid, V., Doerner, K.F., Hartl, R.F., Savelsbergh, M.W.P., Stoecher, W.: An effective heuristic for ready mixed concrete delivery. In: Proceedings TRISTAN, Phuket, Thailand (June 2007)
20. Taillard, É.D.: A heuristic column generation method for the heterogeneous fleet VRP. RO Oper. Res. 33(1), 1–14 (1999)

21. Vanderbeck, F.: Bapcod - a generic branch-and-price code (2008), http://wiki.bordeaux.inria.fr/realopt/
22. Vanderbeck, F., Savelsbergh, M.W.P.: A generic view of dantzig-wolfe decomposition in mixed integer programming. Operations Research Letters 34(3), 296–306 (2006)
23. Vanderbeck, F., Wolsey, L.: Reformulation and decomposition of integer programs. In: 50 Years of Integer Programming 1958-2008, pp. 431–502. Springer, Berlin Heidelberg (2010)