Primal Heuristics for Branch-and-Price: the assets of diving methods

R. Sadykov (1,2), F. Vanderbeck (2,1), A. Pessoa (3), I. Tahiri (2,1), E. Uchoa (3)

(1) INRIA Bordeaux, team RealOpt

(2) Univ. of Bordeaux, Institute of Mathematics, CNRS UMR 5251

(3) Universidade Federal Fluminense, LOGIS

April, 2017

Abstract

Primal heuristics have become an essential component in mixed integer programming (MIP) solvers. Extending MIP based heuristics, our study outlines generic procedures to build primal solutions in the context of a branch-and-price approach and reports on their performance. Our heuristic decisions carry on variables of the Dantzig-Wolfe reformulation, the motivation being to take advantage of a tighter linear programming relaxation than that of the original compact formulation and to benefit from the combinatorial structure embedded in these variables. We focus on the so-called *diving* methods that use re-optimization after each LP rounding. We explore combinations with diversification-intensification paradigms such as *limited discrepancy search* , *sub-MIPing*, *local branching*, and *strong branching*. The dynamic generation of variables inherent to a column generation approach requires specific adaptation of heuristic paradigms. We manage to use simple strategies to get around these technical issues. Our numerical results on generalized assignment, cutting stock, and vertex coloring problems sets new benchmarks, highlighting the performance of diving heuristics as generic procedures in a column generation context and producing better solutions than state-of-the-art specialized heuristics in some cases.

Keywords: Matheuristics, Primal Heuristics, Rounding Heuristics, Diving Heuristics, Branch-and-Price, Column Generation, Generalized Assignment, Cutting Stock, Vertex Coloring.

1. Introduction

Matheuristics are algorithms that attempt to derive "good" primal feasible solutions to a combinatorial optimization problem, by combining mathematical optimization techniques with meta-heuristic paradigms. They make use of the tools of exact optimization, either by truncating an exact procedure or by constructing solutions from the relaxation on which the exact approach relies: techniques range from greedy constructive procedures, for instance by rounding a solution of the linear programming (LP) relaxation, to re-optimization to explore the neighborhood of a solution, to using the LP solution to define a target, or simply to 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 [10]. High quality primal values help pruning the enumeration by bound and by preprocessing (for instance, in reduced cost fixing techniques). 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 become key tools in the last decade. Their developments are reviewed in [8, 24]. Let us just mention a few landmarks: the *Large Scale Neighborhood Search* [3] (an exponential size neighborhood can be explored provided an efficient algorithm exists to search in it), the *Relaxation Induced Neighborhood Search* [17] (the components of the LP solution that are close to the best known integer solution are fixed to those integer values and the residual problem is then solved as a MIP of smaller size), the *local branching* heuristic [23] (the problem is restricted by bounding the number of variables that deviate from the incumbent solution), the *feasibility pump algorithm* [1, 22] (the LP solution is rounded to the closest integer; if infeasible, this solution is used as a target in re-optimizing the LP, and the process iterates). Meta-heuristic paradigms such as the oscillation between intensification and diversification of the search, and the use of historical memory contribute to enhance mathematical programming based heuristics.

Our aim is to extract generic methods that could be seen as natural black-box primal heuristics in branch-and-price algorithms. The column generation literature reports on many application specific studies where primal heuristics are a key to success: some heuristics have been implemented either by taking decisions in the space of the original compact formulation ([35, 37], f.i.), others involve decisions directly in the space of the Dantzig-Wolfe reformulation ([6, 50], f.i.). We make heuristic decisions by rounding variables of the Dantzig-Wolfe reformulation; but we rely on the original formulation to maintain the residual problem after fixing, to im-

plement the resulting proprocessing, and to check integratily for the current solution. This link allows us to implement such maintenance in a generic way. Our motivations for focussig on the Dantzig-Wolfe reformulation are: (i) To take advantage of the benefits of the Dantzig-Wolfe Model. According to Lagrangian duality theory, the Dantzig-Wolfe reformulation offers a tigher LP relaxation than the original formulation. Our experience suggests that stronger linear relaxation dual bounds typically lead to better performance of rounding heuristics. In addition, a Dantzig-Wolfe reformulation avoids the symmetry in the indexing of identical subproblems. While symmetry is less of an issue for primal heuristics, it can blur indications on variables that are close to being integer. (ii) The price coordination mechanism of a Dantzig-Wolfe decomposition brings a global view on the solution space that captures some of the combinatorial structure that is build into the subproblems; fixing variables of the Dantzig-Wolfe reformulation leads to making more aggregate decisions that enable rapid progress in building a primal solution (it has a much stronger impact than when fixing variables of the original formulation); this can be both an advantage (faster progress to a integer solution) and a drawback (of quickly painting yourself in a corner); however, in many applications making more aggregate fixing is more an advantage than a drawback. (iii) Our focus on applying primal heuristic in the Dantzig-Wolfe master is also justified by the fact that it is where the need for innovation lies.

Indeed, making heuristic decisions on the variables of the Dantzig-Wolfe reformulation requires particular attention to derive heuristics that are "compatible" with column generation. Enforcing additional constraints such as local branching or bounds on the master variables, or amending the objective such as required by the feasibility pump paradigm, can in some cases lead to destroying the special structure that made pricing tractable. This explains why the above mentioned landmark heuristics have not been applied directly in combination with column generation. In comparison, if one takes heuristic decisions on the variables of the original compact formulation, then the generic primal heuristics for mixed integer programming seem to apply directly. However, as we shall see, if one aims to rely on the Dantzig-Wolfe reformulation LP solution to take heuristic decisions in fixing variables of the original formulation, one is bound to face some of the same technical issues when re-optimizing the master after the rounding operation. This is so because the original solution space and the Dantzig-Wolfe reformulation integer solution space are not in a bijective relationship in the common case of identical subsystems, and because a modified original formulation (as induced by the primal heuristic) calls for a modified DW reformulation.

The rest of the paper is organized as follows. In Section 2, we review standard primal heuristic paradigms of Mixed Integer Programming that one could consider for extension in a column

generation context. In Section 3, we review the established use of primal heuristic techniques in the column generation literature. In Section 4, we examine the specific technical issues that arise when trying to extend primal heuristic paradigms to the context of a column generation approach. In Section 5, we derive ways to get around these difficulties and show how to adapt the implementation of various generic paradigms in the column generation context. Our procedures have in common to be based on the *diving* heuristic paradigm [8] which combines rounding with linear programming re-optimization by column generation. Re-optimization along with pre-processing techniques are essential to obtain feasible solutions. The core diving heuristic is combined with generic heuristic paradigms such as intensification and diversification. Section 6 provides a comparison of the relative performances of our heuristic implementations on instances of generalized assignment, cutting stock, and vertex coloring problems. Our best performance is compared with heuristics of the literature. In our conclusion, we analyze the performance of these generic primal heuristics, highlight the limitation in their applicability, and discuss further research questions.

The paper builds on our preliminary results in this domain that appeared in extended abstracts of conference proceedings [33, 41, 48]. The method review and the numerical study are quite more comprehensive in the present paper, so is the benchmarking with results of the literature. The computational progress over our preliminary work is important. It results from some key features: (i) a full-blown integration with pre-processing techniques at each node of the diving tree (pre-processing being essential to avoid fixings that lead to being trapped into an infeasible residual problem); (ii) a combination with automated stabilization schemes [42] that reduce significantly the master re-optimization cost after each rounding (essential for performance in terms of CPU time and master problem size); (iii) diversification strategies that aim to cope with infeasibilities rather than sub-optimality; (iv) a wider spectrum of implementation strategies, exploiting paradigms such as strong branching that leads to improved solution quality.

2. Generic mathematical programming heuristics

Basic instruments of standard primal heuristic based on mathematical programming approaches are *rounding*, *diving* and *sub-MIPing*, along side selection rules such as *greedy* or *guided search*. Assume a pure bounded integer program with n variables:

$$z := \min\{c\,x : A\,x \geq a,\ x \in I\!N^n\}\ . \tag{1}$$

Let x^{LP} be a solution to its Linear Programming (LP) relaxation. A *rounding heuristic* consists in iteratively selecting a fractional component of x^{LP} and setting its lower bound to the rounded-up fractional value or its upper bound to the rounded-down value. The method is defined by a rule for selecting the component and the rounding direction. Classical selection rules in this context are *least fractional*, *guided search*, *least pseudo-cost*, or *least infeasibility* among the rules reviewed in [8].

Diving differs from simple rounding heuristic by the fact that the LP is re-optimized after bounding or fixing one or several variables. A diving heuristic can be understood as a heuristic search in an LP-based branch-and-bound tree: the search plunges deep into the enumeration tree by selecting a branch heuristically at each node, as illustrated in Figure 1a. The branching rule used in such context is typically quite different from the one used in an exact approach: in a diving heuristic, one is not concerned with balancing the tree, but one aims at finding good primal solutions quickly.

Sub-MIPing consists in restricting the MIP problem to a residual problem of smaller size by way of fixing (or bounding) some of its variable values; the restricted MIP is then solved exactly or approximately using a MIP solver. Thus, the method is essentially defined by a way of restricting the problem. For instance, one can restrict the problem by fixing some of its variables through a rounding or diving heuristic, as illustrated in Figure 1b; or by restricting the set of variables, implicitly fixing to zero variables of the complementary set.

Some of the heuristic paradigms that are mentioned in the literature can be seen as a specific implementation of *Sub-MIPing*: so are *Relaxation Induced Neighborhood Search* (RINS) or *local branching*; others, like *feasibility pump*, reduce the set of solutions to be considered using soft-constraints (i.e., penalizing deviation). These methods were originally developed for a binary integer program:

$$\min\{c\,x : A\,x \geq a,\ x \in \{0,1\}^n\}\ . \tag{2}$$

In the *Relaxation Induced Neighborhood Search* the restriction of the solution set is defined using a guided search rounding, i.e., fixing the variables that are the closest to their value in the incumbent solution. The *crossover heuristic* [8] is similar, except that one fixes variables which are at the same values in several available primal solutions. This crossover method was extended in [37] to a hybrid primal heuristic relying on both compact and extended Dantzig-Wolfe reformulation: fixing is performed in the original compact formulation based on the frequency of

values appearing in the columns generated while solving the master LP.

The *local branching* heuristic [23] entails exploring the neighborhood of the incumbent solution by solving a constrained problem:

$$\min\{c\,x : A\,x \geq a,\ \sum_{j\in J^0} x_j + \sum_{j\in J^1}(1-x_j) \leq \Delta,\ x \in \{0,1\}^n\}\ , \tag{3}$$

where J^0 and J^1 define respectively the components that take value 0 or 1 in the incumbent solution.

The *feasibility pump* heuristic [1, 22] entails exploring a sequence of trial solutions, $\tilde{x}$, obtained by rounding to the closest integer 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:

$$\min\{c\,x + \epsilon(\sum_{j\in J^0} x_j + \sum_{j\in J^1}(1-x_j)) : A\,x \geq a,\ x \in [0,1]^n\}\ . \tag{4}$$

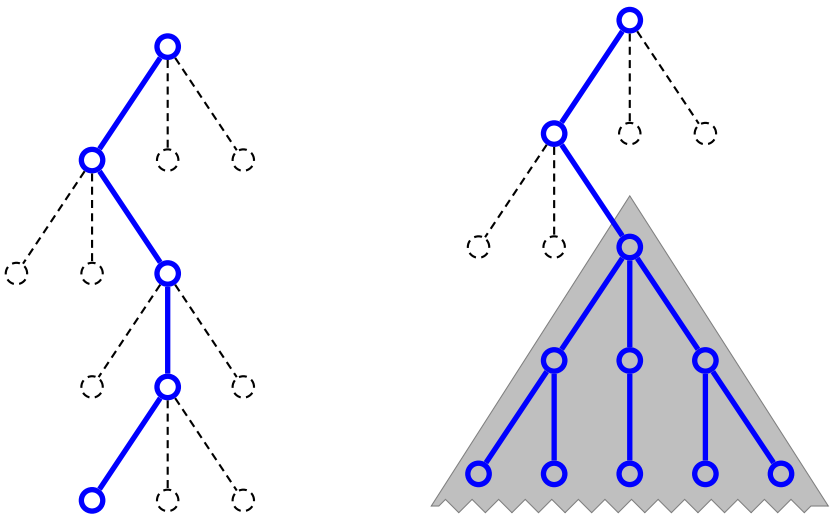
Sets J^0 and J^1 define respectively the components that take value 0 or 1 in the previous trial integer solution, $\tilde{x}$, obtained through rounding the LP solution of (4) as it was defined at the previous iterate. Parameter ϵ controls the impact of the cost modifications relative to the original objective. Extending the above heuristics to a general integer program is possible. For instance, for feasibility pump, it requires an adapted distance measure [7, 25].

3. Review of column generation based primal heuristics

In the context of a column generation approach, we assume a mixed integer program with decomposable structure. A subset of constraints is identified as a tractable subproblem, it typically has a *block diagonal* structure:

$$[P] \quad \min\{cx : Ay \geq a, y = \sum_k x^k, \underbrace{B^k x^k \geq b^k\ \forall k,\ x^k \in I\!N^n\ \forall k}_{x^k \in X^k\ \forall k}\} \tag{5}$$

where $Ay \geq a$ represent "complicating constraints", while optimization over the mixed-integer polyhedron, X^k, defined by subsystem $B^k x^k \geq b^k$, is "relatively easy". To simplify the presentation, X^k are assumed to be bounded pure integer programs and G^k is an enumeration of the



(a) Heuristic dive in an enumeration tree

(b) Exact solution of a Sub-MIP after a heuristic partial fixing of the solution

Figure 1: Illustration of two heuristic paradigms.

feasible solutions to X^k, i.e., $\{x^g\}_{g \in G^k}$ defines the feasible solutions of X^k. The structure of $[P]$ can be exploited to reformulate it as the master program:

$$[M'] \equiv \min\{ \sum_{k, g \in G^k} (cx^g)\lambda_g^k : \sum_{k, g \in G^k} (Ax^g)\lambda_g^k \geq a;\ \sum_{g \in G^k} \lambda_g^k = 1\ \forall k;\ \lambda_g^k \in \{0,1\}\ \forall g, k\}\,. \quad (6)$$

When each block is identical, as in the case of many applications, i.e., when $X^k = X = \{Bx \geq b, x \in I\!N^n\}\ \forall k$, where $Bx \geq b$ denotes the identical constraint set defining one block, then, the master takes the form:

$$[M''] \equiv \quad \min\{\sum_{g \in G} (cx^g)\lambda_g : \sum_{g \in G} (Ax^g)\lambda_g \geq a;\ \sum_{g \in G} \lambda_g = K;\ \lambda_g \in I\!N \quad \forall g\}\,, \quad (7)$$

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 (7), i.e., $[M] = [M'']$, unless specified otherwise.

The so-called master program $[M]$ is known as the Dantzig-Wolfe reformulation of $[P]$. It 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:

$$\min\{(c - \pi A)\, x\ :\ Bx \geq b, x \in I\!N^n\}\,. \quad (8)$$

Some primal heuristics have been routinely used in this column generation context. Beyond simple *greedy heuristics* (iterative greedy selection of a column into the partial solution) that are application specific, the most widely used heuristic is the so-called *restricted master heuristic*. This generic scheme can be seen as a sub-MIPing 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 restricted set $\overline{G}$ is typically defined as the set of columns generated during the master LP solution. Hence, this method is also called *price-and-branch*. The main drawback of this approach is that the resulting restricted master integer problem may be infeasible in many cases (as reported in our tests), i.e., often the columns of $\overline{G}$ may not be combined into an integer solution. In an application specific context, an ad-hoc recourse can be designed to "repair" infeasibility. Another typical implementation of such sub-MIPing heuristic is to define set $\overline{G}$ using the columns of several constructive heuristic solutions, possibly augmented with the LP solution columns. This guarantees feasibility of a restricted master heuristic. The method can then be viewed as the search for improving integer solutions in a neighborhood of the initial heuristic solutions (the neighborhood is defined by the columns set). However, the method often produces no better solutions than that of the initial heuristic solutions. Implementations of restricted master heuristics have been developed, for instance, for production planning [4], interval scheduling [5], network design [14], vehicle routing [2, 9, 13, 51] and delivery [49] problems.

Close to being generic, but not quite, *rounding heuristics* procedures have been applied to the master LP solution. In the common case where the master formulation defines a set covering problem, a standard rounding strategy consists of 3 steps: (i) an initial partial solution is obtained by rounding downwards the master LP solution; (ii) then, columns whose LP values are closest to the next integer are then considered for round up while feasible; (iii) finally, an ad-hoc constructive procedure is used to complete the solution. *Local search* can be used to improve the solutions while implementing some form of diversification, but this is again typically application specific: one can remove some of the columns selected in the primal solution and reconstruct a solution with one of the above techniques. Rounding heuristics (sometimes coupled with local search) have been successfully applied on cutting stock, planning and vehicle routing problems [6, 12, 15, 21, 40]. However, reaching feasibility remains a difficult issue that is handled in an application specific manner.

Diving heuristics are generic ways of "repairing" infeasibilities. The residual master problem that remains after a rounding operation must be "cleaned up" before re-optimization, deleting all columns that could not be part of an integer solution to the residual problem (and hence

would lead to an infeasibility if selected). Such preprocessing that is specific to a column generation context is presented in [55]; it leads to the definition of *proper columns*, i.e., columns that could take a non-zero integer value in an optimal solution to the residual master problem. We eliminate columns that become unproper after rounding. This preprocessing is a key feature in diving heuristics. It helps to avoid the primal heuristic dead-ending with an infeasible solution. Furthermore, we focus on generating proper columns in future pricing: if the pricing problem solver can handle the bounded version of the column generation sub-problem, one can tighten lower and upper bounds on pricing problem variables to generate proper columns. Note that the re-optimization of the residual master 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 through preprocessing. Although it is an important feature for the success of the approach, re-optimization of the master LP after fixing can be time consuming. Tuning the level of approximation in this re-optimization allows one to control the computational effort. Diving heuristics were successfully used, for instance, on vehicle routing [11, 29, 45, 50], inventory routing [38], crew rostering [27, 28], bin packing [47], cutting stock [35, 40], graph partitioning [16], machine scheduling [34], freight rail scheduling [18, 46] and lot-sizing [19] problems.

4. Issues in combining column generation and heuristics

Deriving general purpose primal heuristics based on the Dantzig-Wolfe reformulation raises some difficulties. Bounding a master variable in $[M]$ or modifying its cost can be incompatible with the column generation approach in the sense that it can induce modifications to the pricing problem that are not amenable to the pricing solver. Basically the issues are:

- Setting an upper bound on an existing column, as one might wish to do in diving heuristics, i.e., enforcing $\lambda_g \leq u_g$, yields an associated dual variable that must be accounted for in pricing (by modeling an extra cost for the specific solution x^g). If constraint $\lambda_g \leq u_g$ are ignored when pricing, the column x^g might be wrongly regenerated as the best price solution. One could restrict the pricing problem to avoid regenerating x^g, but this induces significant modifications to the pricing procedure that are as bad as accounting for the additional dual price.

- 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.

- Adding slightly more general constraints as that of the form (3) for a local branching heuristic (and the like in a feasibility pump heuristic applied in the general integer case), results in incompatibility issues of the same nature than adding upper bounds of the form $\lambda_g \leq u_g$.

- Increasing the cost c_g of a variable λ_g, as needed in the feasibility pump paradigm, means that λ_g will price out negatively according to the original pricing oracle, and hence it can be wrongly regenerated by the original oracle (which models the initial cost) as the best pricing problem solution.

- Decreasing the cost c_g of a variable λ_g, however, is amenable to the unmodified column generation scheme, as using the original pricing oracle shall simply lead to overestimate the reduced cost of such already included column.

Thus, the only trivial operations in the Dantzig-Wolfe master are lower bound setting on master variables and cost reductions.

Observe that applying primal heuristic in the space of the original formulation (5) can also raise some technical questions. Having solved the master LP relaxation of (6), one can project its solution in the original space, defining

$$x^k = \sum_{g \in G^k} x^g \lambda_g^k \ , \tag{9}$$

and apply the procedures of Section 2 on this projected solution. There remain two issues however:

- The projection is not uniquely defined in the case where there are K identical subproblems. Thus, the variable fixing done by the primal heuristic can be ignored by producing another symmetric LP solution where the indexing of subproblem is permutated. To avoid this issue, one can break symmetries by defining a unique disaggregation of the LP solution of the aggregate master (7), using for instance a lexicographic disaggregation, as defined by a recursive rule:

$$\lambda_g^k = \min\{1, \lambda_g - \sum_{\kappa=1}^{k-1} \lambda_g^k, (k - \sum_{\gamma \prec g} \lambda_\gamma)^+\} \quad \forall k = 1, \ldots, K, g \in G \ , \tag{10}$$

where $\prec$ defines a lexicographic ordering of columns $g \in G$ (see [56] for details). One can then use projection (9), but the reverse relation cannot be properly enforced, i.e., a restriction on x_j^k variables cannot be modelled in (7).

- The problem modifications must typically be relaxed in a Lagrangian way to avoid any modifications to the pricing problem. For instance, restrictions of the form $x_j^k \leq u_j$ shall be added to the master and hence dualized in the same way as the "complicating constraints", $Ay \leq a$. If one wants a stronger enforcement in the pricing problem, this leads to structural modifications that might not be amenable in the pricing solver.

To get around these technical issues raised by the compatibility with column generation, we restrict our methods to using lower bound settings and cost reductions. A key observation is that defining lower bounds on master variables can be seen as a procedure to define a partial solution, $\hat{\lambda}$. The **residual master problem** that remains at iteration t, once the partial solution $\hat{\lambda}^t$ is extracted, takes the form:

$$[M^t] \equiv \min\{\sum_{g \in G^t} c_g^t \lambda_g : \sum_{g \in G^t} (Ax^g)\lambda_g \geq a^t;\ \sum_{g \in G^t} \lambda_g = K^t;\ \lambda_g \in I\!N \quad \forall g \in G^t\} \tag{11}$$

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 (7) by column generation; while $a^0 = a$, $K^0 = K$, and $c_g^0 = cx^g$ for all $g \in G^0$. At iteration t, $a^t = a - \sum_g (Ax^g)\hat{\lambda}_g^t$ and $K^t = K - \sum_g \hat{\lambda}_g^t$. The columns included in the partial solution shall remain in the residual problem for further selection if they are suitable (i.e., if they define proper columns for the residual problem [55]). Note that the residual master can be tackled in the same way as the original master LP, with the same pricing oracle, as no specific bounds on master variables have been set explicitly. Thus, the original column generation approach is applied to the residual problem (11) in a the re-optimization that is used at each step of the diving procedure.

5. Heuristic paradigms for column generation

Adapting MIP paradigms to the column generation context, we describe a diving heuristic that exploits the Dantzig-Wolfe reformulation of a problem. The variants that we consider have in common a procedure that incrementally updates a partial solution by incorporating master columns at a value obtained by rounding their LP value. After such update of the partial solution and redefinition of the residual master problem, the latter is re-optimized by column generation. To recap the interest of such *diving* heuristic approach, first note that it can be implemented in a

way that avoids the above discussed difficulties on bounding or fixing variables: each heuristic decision corresponds implicitly to defining a lower bound on the associated variable. Secondly, the method is less likely to dead-end with an infeasible residual master than rounding or restricted master heuristics: re-optimization either yields the "missing" complementary columns to a build feasible solution, or it allows to detect early the infeasibility of the residual problem (Simplex Phase 1 can provide such proof). The basic *diving* paradigm is then combined with partial backtracking (using *Limited Discrepancy Search*), *strong branching*, *sub-MIPing*, *local search*, and *local branching* features to intensify or diversify the search.

5.1 Pure diving

A pure *diving heuristic* is a simple depth-first heuristic search in the branch-and-price tree. At each tree node, a branch is heuristically selected based on a rounding strategy: it corresponds to rounding up or down a variable λ_g of the master LP solution. We denote this rounding as $\lceil \bar{\lambda}_g \rfloor$. To ensure compatibility with column generation, we translate such rounding operations into fixing a partial solution: rounding down variable λ_g means taking $\lfloor \lambda_g \rfloor$ copies of this column in the partial solution, while a round-up corresponds to taking $\lceil \lambda_g \rceil$ copies of this column. Thus, both round-up and round-down are implemented as setting a lower bound on the column use, and the column remains in the formulation. After such rounding operation, the *residual master problem* (11) and the pricing problem are modified by applying preprocessing techniques before re-optimization.

A generic template of a pure diving procedure is given in Table 1 in a recursive form where t is the iteration counter. The parameters of this procedure are elements: $G^{t-1}, a^{t-1}, K^{t-1}$, which define the previous residual master $[M^{t-1}]$. Other parameters are the current partial solution, $\hat{\lambda}$, and the current rounding, $\tilde{\lambda}$, of the LP solution to the residual master $[M^{t-1}]$: $\tilde{\lambda}$ defines variables and their values which should be added to the current partial solution $\hat{\lambda}$. To start the diving procedure, we call PUREDIVING$(G^0, a, K, \emptyset, \emptyset)$, where G^0 is the set of columns obtained while solving the linear relaxation of the initial master problem $[M]$.

In *Step 2*, set G^t denotes the set of columns that are suitable for the current residual master program (11):

$$G^t := \{g \in G : \ \hat{x}_j^l \leq x_j^g \leq \hat{x}_j^u \quad \forall j = 1, \ldots, n\} \tag{12}$$

where $\hat{x}_j^l$ and $\hat{x}_j^u$ are valid lower and upper bounds on pricing problem solution defining "proper" columns [55]. Columns in $G^{t-1} \setminus G^t$ are removed from residual master problem $[M^t]$.

PUREDIVING$(G^{t-1}, a^{t-1}, K^{t-1}, \hat{\lambda}, \tilde{\lambda})$

Step 1: Update the master: $a^t \leftarrow a^{t-1} - \sum_{g \in G} Ax^g \tilde{\lambda}_g, \quad K^t \leftarrow K^{t-1} - \sum_{g \in G} \tilde{\lambda}_g$.
Update partial solution: $\hat{\lambda} \leftarrow \hat{\lambda} + \tilde{\lambda}$.

Step 2: Apply pre-processing to residual master (11) and the associated pricing problem. Let G^t denote the set of columns that remain proper. If residual master problem is shown infeasible, return.

Step 3: Solve the LP relaxation of the current residual master $[M^t]$ (11), let λ^t denote its LP solution. If the problem is shown infeasible through Phase I of the Simplex algorithm, return.

Step 4: Let $F = \{g \in G^t : \lfloor \lambda^t_g \rfloor < \lambda^t_g < \lceil \lambda^t_g \rceil\}$.
If $\hat{\lambda} + \{\lambda^t_g\}_{g \in G^t \setminus F}$ defines a feasible primal solution to (7), record this solution.

Step 5: $\tilde{\lambda} \leftarrow 0$. If $F = \emptyset$, return.
Heuristically choose a column set $R \subseteq F$ and heuristically round their values: $\tilde{\lambda}_g \leftarrow \lceil \lambda^t_g \rfloor$ such that $\tilde{\lambda}_g > 0$ for $g \in R$.
Recursively call PUREDIVING$(G^t, a^t, K^t, \hat{\lambda}, \tilde{\lambda})$.

Table 1: Pure diving heuristic

A central element of the procedure that drives the heuristic is *Step 5* in which we choose columns for rounding. Note that several columns can be taken into the solution simultaneously. After solving the current residual master LP, one selects in the partial solution one or several columns $g \in F = \{g \in G^t : \lfloor \lambda^t_g \rfloor < \lambda^t_g < \lceil \lambda^t_g \rceil\}$, at heuristically set values $\lceil \bar{\lambda}_g \rfloor$. One then checks whether the current partial solution defines a solution to the full-blown problem. The important feature of the diving procedure is preprocessing, master and pricing problems are updated to "proper" columns and the process reiterates while the residual master problem is not proven infeasible.

In our implementation of *Step 5*, we choose one column $g \in F$, whose value λ^t_g is closest to its nearest non-zero integer. Then, we round λ^t_g to its nearest non-zero integer. In our experiment, we noted that rounding one column at a time yields typically better results. This is probably because we make less macroscopic fixing and take the benefit of re-optimization and preprocessing of the modified master.

5.2 Diving with limited backtracking

Here, we consider a limited backtracking scheme. It relies on using the *Limited Discrepancy Search (LDS)* paradigm of [30]. This original feature defines a scheme to diversify the search. It has a great impact in improving the performance of the pure diving method. We call this algorithm "Diving with LDS". Furthermore, we developed a specific implementation of a limited backtracking scheme for use when it is hard to find a feasible solution to the problem using pure diving. In this heuristic, that we call "Diving for Feasibility", backtracking is used to intensify the search towards the leaves of the search tree, exploring the neighborhood of the first dive solution, until a feasible solution is identified.

DIVINGWITHLDS$(G^{t-1}, a^{t-1}, K^{t-1}, \hat{\lambda}, \tilde{\lambda}, T, d)$

Steps 1–4 are the same as in PUREDIVING

Step 5: **Repeat** the following

1. $\tilde{\lambda} \leftarrow 0$. If $F \setminus T = \emptyset$, return.
2. Heuristically choose a column set $R \subseteq F \setminus T$, and heuristically round their values: $\tilde{\lambda}_g \leftarrow \lceil \lambda_g^t \rfloor$ such that $\tilde{\lambda}_g > 0$ for $g \in R$.
3. Recursively call DIVINGWITHLDS$(G^t, a^t, K^t, \hat{\lambda}, \tilde{\lambda}, T, d+1)$.
4. $T \leftarrow T \cup R$.

While $|T| \leq$ *maxDiscrepancy* **and** $d \leq$ *maxDepth*.

Table 2: Diving heuristic with limited backtracking

The scheme is applied from the root node of the search tree. Backtracking is performed up to depth *maxDepth*. When a backtrack is performed back to a node, the backtracked branching decision is forbidden for other branches from that node and in the subtrees "rooted" at these branches; i.e., the column that was selected for rounding in the backtracked branch cannot be selected as a candidate for rounding in other branches. This is implemented by using a tabu list that includes forbidden branching decisions: the tabu list is a set of columns, denoted T, that are forbidden to be chosen for rounding. Backtracking branches are considered as long as the number of columns in the tabu list does not exceed *maxDiscrepancy* which is the second parameter of the scheme. The template for the diving heuristic with limited backtracking (named "diving with LDS") in a recursive form is presented in Table 2. This procedure is derived from

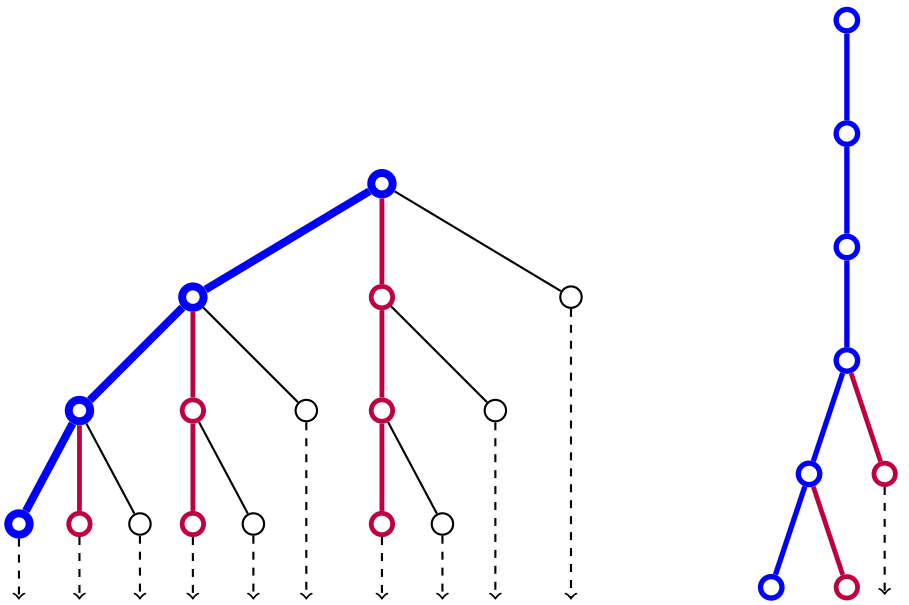
the pure diving procedure with a modified *Step 5* and two additional parameters: the tabu list T and the current search tree depth d. To start the diving heuristic with limited backtracking, we call DIVINGWITHLDS$(G^0, a, K, \emptyset, \emptyset, \emptyset, 1)$.

Our implementation of *Step 5* of the diving heuristic with limited backtracking is similar to the one for the pure diving heuristic. We choose a column $g \in F \setminus T$, whose value λ_g^t is closest to its nearest non-zero integer, and we round λ_g^t to its nearest non-zero integer. Thus, our set R has cardinality one and our tabu set increases by one unit at the time. An example of the search tree for the diving heuristic with limited backtracking is depicted in Figure 2-(a). In this example, the instantiation of the parameters is *maxDepth* $= 3$, *maxDiscrepancy* $= 2$, and, as in our implementation, we round one column at a time. In this figure, the boldest branches have an empty tabu list, less bold branches have tabu list with one column, and thin branches have the tabu list with two columns.

When the main goal is to find a feasible solution, we consider using backtracking toward the leaves of the depth-first-search dive in the branch-and-price tree. This algorithm that we call "Diving for Feasibility", entails a specific parametrization of the above DIVINGWITHLDS procedure: *maxDepth* $= \infty$; additionally, the procedure stops as soon as a feasible solution is found at *Step 4* of PUREDIVING. Observe that given that we use a depth-first-search strategy, the backtracking will take place towards the leaves, by reconsidering in priority the last variable fixings. As the procedure stops as soon as a feasible solution is found, it is unlikely that the search backtracks up to the root. In our implementation we set *maxDiscrepancy* $= 1$, the resulting search scheme is illustrated in Figure 2-(b). This heuristic tends to yield a feasible solution in very small additional time compared to pure diving heuristic when the latter cannot find one.

5.3 Strong diving

Strong branching [36] is a strategy to make "intelligent" look-ahead branching decisions in an effort to reduce the size of the branch-and-bound tree when solving a mixed integer program. The idea is to select among several possible branching decisions by comparing the impact they have on improving the dual bound. For this, one performs a (possibly approximate) evaluation of the dual bound of the child nodes, i.e., one solves (approximately) the linear programming relaxation of the child nodes with the branching decision temporarily applied. Then, a branching decision is chosen which generates the best improvement in the local dual bound computed from the child node's (approximate) dual bounds.



(a) Diving with LDS

(b) Diving for Feasibility

Figure 2: (a) is a "Diving with LDS" example for *maxDepth* $= 3$ and *maxDiscrepancy* $= 2$; (b) is a "Diving for Feasibility" example for *maxDiscrepancy* $= 1$, assuming a feasible solution was found after 2 back-tracks

Our use of the strong branching paradigm in the diving procedure goes as follows. We choose several columns as candidates for rounding. The number of candidates is limited by the parameter *maxCandidates*. Then, for each candidate, we apply temporarily the rounding and compute the resulting dual bound for the associated residual master problem (making use of the column generation procedure). Contrary to the "classic" strong branching applied in branch-and-bound, the best candidate is that which generates the child node with the smallest deterioration of the dual bound. Indeed, the heuristic explores a single branch in search for the best possible primal bounds. This is different from the aim of getting the best dual bound improvement that is pursued in a "classic" branch-and-bound procedure.

The template for the strong diving heuristic in a recursive form is presented in Table 3. Our procedure is combined with the paradigm of the diving heuristic with limited backtracking. In *Step 2* and *Step 3*, we choose a candidate set of columns, and then each candidate is evaluated. *Step 5* of the DIVINGWITHLDS is modified into *Step 4* in STRONGDIVING: for the next diving decision, we choose a column which rounding results in the smallest increase of the dual bound

STRONGDIVING$(G^t, a^t, K^t, \hat{\lambda}, F, T, d)$

Step 1: If $d = 1$, run Steps 2–4 of PUREDIVING.

Step 2: Heuristically choose a column set $C \subseteq F$, such that $|C| = \min\{$*maxCandidates*$, |F|\}$.

Step 3: **For each** candidate $g \in C$

1. $\tilde{\lambda} \leftarrow 0$. Heuristically round λ_g: $\tilde{\lambda}_g \leftarrow \lceil \lambda^t_g \rfloor$ such that $\tilde{\lambda}_g > 0$.
2. Call Steps 1–4 of PUREDIVING$(G^t, a^t, K^t, \hat{\lambda}, \tilde{\lambda})$. At the end of this call, save the following: $G^{t,g} \leftarrow G^t$, $a^{t,g} \leftarrow a^t$, $K^{t,g} \leftarrow K^t$, $\hat{\lambda}^g \leftarrow \hat{\lambda}$, $F^g \leftarrow F$. Also, let db^g be the value of the LP relaxation of $[M^t]$ in Step 3 of this call.

Step 4: **Repeat** the following

1. Choose $g \in C$ with the smallest value db^g.
2. Recursively call STRONGDIVING$(G^{t,g}, a^{t,g}, K^{t,g}, \hat{\lambda}^g, F^g, T, d + 1)$.
3. $T \leftarrow T \cup \{g\}$, $C \leftarrow C \setminus \{g\}$.

While $|T| \leq$ *maxDiscrepancy* **and** $d \leq$ *maxDepth*.

Table 3: Strong diving heuristic

of the master problem linear relaxation. In our implementation, we define C to be the columns with values closest to their non-zero integers; their number is bounded by *maxCandidates*. In *Step 3*, we round λ_g to its nearest non-zero integer. To start the strong diving heuristic, we call STRONGDIVING$(G^0, a, K, \emptyset, \emptyset, \emptyset, 1)$.

5.4 Diving with restarts

The paradigm considered here consists in intensifying the search around the incumbent solution initially obtained by the "diving for feasibility" heuristic. Our "local search" is fixing a part of the solution to what was the incumbent solution and restarts the pure diving heuristic for the residual master problem. It can be understood as an implementation of a *neighborhood search* paradigm when the part of the solution that one fixes defines the neighborhood around the previous dive solution. The parameters of this scheme are *fixRatio* which is the ratio of

columns participating to the incumbent solution that remain fixed, and *numIterations* which is the number of restarts of the procedure.

DIVINGWITHRESTARTS(G^0, a, K)

Step 1: Call DIVINGWITHLDS$(G^0, a, K, \emptyset, \emptyset, \emptyset, 1)$ with parameters *maxDepth* $= \infty$, *maxDiscrepancy* $= 1$, and stop it as soon as a feasible solution λ^* is obtained.

Step 2: **Repeat** *numIterations* times

1. Let $R = \{g \in G : \lambda^*_g > 0\}$.
2. Choose set $C \subset R$ such that $|C| = \lceil \textit{fixRatio} \cdot |R| \rceil$.
3. $\tilde{\lambda} \leftarrow \sum_{g \in C} \lambda^*_g$.
4. Call PUREDIVING$(G^0, a, K, \emptyset, \tilde{\lambda})$.
5. If a feasible solution λ is obtained during this call and $c\lambda < c\lambda^*$ then $\lambda^* \leftarrow \lambda$.

Table 4: Diving heuristic with restarts

The template for the diving heuristic with local search is presented in Table 4. In *Step 1*, we call diving for feasibility to obtain the initial feasible solution. In our implementation of *Step 2* of this procedure, the set C of columns to fix is chosen randomly (uniformly). An example of the search tree for the diving heuristic with restarts is depicted in Figure 3. In this example, parameters are set as *fixRatio* $= 0.5$ and *numIterations* $= 2$.

5.5 Diving combined with sub-MIPing

Sub-MIPing refers to applying a MIP solver to a restricted problem. The standard *restricted master heuristic* consists in limiting the set of variables in the master to the subset of columns, G^0, that were generated during the solution of the root node LP, and applying to it a MIP solver without any further column generation. As it is shown by our computational experiments below, the restricted master heuristic often fails to obtain a feasible solution. Moreover, even if a feasible solution is obtained, its quality is bad in many cases. The improved method considered here attempts to correct these drawbacks.

The first drawback is addressed by calling the diving for feasibility heuristic before hand, so as to obtain an initial feasible solution. The second drawback is partially taking care of by

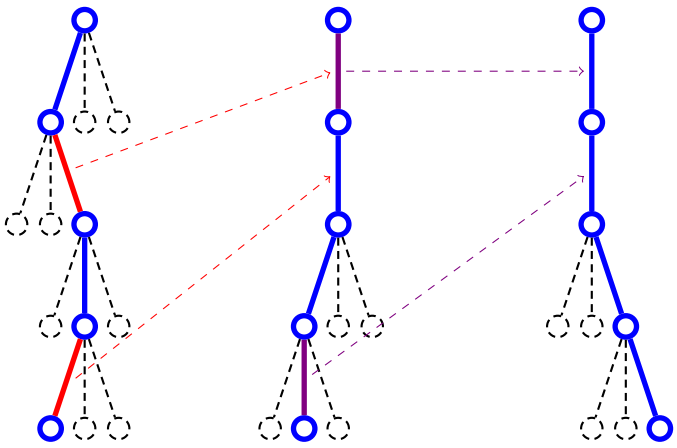


Figure 3: Diving heuristic with restarts: example with *fixRatio* $= 0.5$ and *numIterations* $= 2$ assuming the initial incumbent was obtained without any backtracking.

the same token, as collecting all the columns generated during the diving heuristic, in addition to those produced while solving the root node LP, provides a diversified set of columns and hence it gives a better chance to obtain better quality solutions. The method can be seen as a combination of two generic heuristics, and thus it is generic too.

5.6 Local branching

The local branching heuristic is a kind of sub-MIPing heuristic that can be adapted to the column generation context. The paradigm is to explore the neighborhood of the incumbent solution by adding a constraint that limits the deviation from this incumbent solution. To obtain an initial incumbent solution λ^* in a generic way, we use the diving for feasibility heuristic. Then, we add to the master the local branching constraint: let $R = \{g \in G : \lambda^*_g > 0\}$ and $r^* = \sum_{g \in R} \lambda^*_g$, and set the local branching constraint as

$$\sum_{g \in R} \lambda_g \geq r^* - \lceil r^* \cdot \textit{deviationRatio} \rceil, \tag{13}$$

where *deviationRatio* is a parameter specifying what is the allowed deviation ratio from the initial solution. Thus, the local branching heuristic is similar to the procedure of Section 5.5, except that instead of fixing an initial set of variables in *Step 2.3*, we add cut (13) to the master program. The latter constraint permits a significant reduction in the running time and thus alleviates a drawback of the "restricted master heuristic". However, due to the local branching constraint, the set of feasible solutions becomes smaller and the resulting solution can be worse. A variant of the local branching heuristic would be to use the full blown branch-and-price

procedure for re-optimization after adding the local branching cut. However, our tests show that this variant is too computationally demanding to be competitive.

5.7 Feasibility pump

The feasibility pump heuristic can be partially adapted to the column generation context. As indicated in Section 4, cost c_g of a master variable λ_g can be decreased without an impact on the pricing problem. However, increasing the cost above the true cost of a variable is not possible. Therefore, changing the cost of master variables as required by the feasibility pump paradigm is possible, but with the limitation not to exceed the true cost. The details on a possible implementation and the associated numerical tests can be found in our conference paper [41]. They are not reported here since we have not found this approach competitive in regards to the above methods. The only potential advantage underlined in the method of [41] is that a feasible solution is obtained more frequently than when using a pure diving heuristic of Section 5.1. However, the diving for feasibility heuristic of Section 5.2 shows better performance than feasibility pump: it yields a feasible solution more frequently and in less time when compared to our implementation of a feasibility pump heuristic.

6. Numerical study

In implementing the above described heuristics, there are practical considerations that can have a significant impact on the method performance. We emphasize in particular the role of stabilization techniques in reducing the number of iterations of the column generation procedure (in the initial master LP optimization and its re-optimization): we use the methods presented in [42]. Speeding up the master solution time is even more important in the context of heuristics where one trades optimality for response time. Our benchmarking is against other quick heuristic methods. Other important factors are pre-processing and master "clean-up" strategies (removing "unattractive" columns from the restricted master) and column generation strategies (such the generation of multiple columns at each iteration).

Let us emphasize the importance of preprocessing. After updating the partial solution and the master problem (Step 1 of PUREDIVING), preprocessing acts to remove columns that would violate master constraints if taken at a positive integer value (Step 2 of PUREDIVING). Indeed, if these columns were selected for rounding, they would lead to dead-ending with an infeasible master solution. Thus, using proper columns is key to avoid infeasibility in the diving heuristic. To avoid generating non-proper columns, bounds, defining proper columns in (12), are enforced

on pricing problem variables if the pricing oracle can account for these. Our procedure for subproblem variable bound tightening is described in [53]: bounds are tightened not only based on subproblem constraints but also on the basis of the subproblem variable input in the master constraints of the residual problem.

It is important to be aware that the performance of the heuristics is sensitive to the master problem initialization, the LP solver used, and any feature that has an influence on the master LP solution at the root node. Indeed, due to degeneracy, the master LP admits typically many alternative solutions, each of which shall provide a different seed for the heuristic with a different outcome. Hence, when benchmarking against best solutions of the literature, we perform several runs to exploit different seeds.

We have tested our heuristics on three applications: Generalized Assignment, Cutting Stock, and Vertex Coloring as reported in the following subsections. The specific heuristic procedures that we compared are:

1. Pure Diving.
2. Diving for Feasibility with parameter *maxDiscrepancy* $= 1$.
3. Diving with Restarts with parameters *fixRatio* $= 0.5$, *numIterations* $= 10$.
4. Local Branching with parameter *deviationRatio* $= 0.3$.
5. Diving with LDS with parameters *maxDiscrepancy* $= 3$, *maxDepth* $= 2$.
6. Strong Diving with parameters *maxDiscrepancy* $= 3$, *maxDepth* $= 2$, *maxCandidates* $= 10$.
7. A pure Restricted Master.
8. Diving with SubMIPing.

The parameter settings have been selected through a pre-study: allowing more diversification increases computing time without much gain in terms of solution quality. Note that "Diving with LDS" builds on "Diving for Feasibility", while "strong branching" is built over the diving with LDS procedure.

All experiments were run on a Dell PowerEdge 1950 (32Go, Intel Xeon X5460, 3.16GHZ). To solve the restricted master LP, Cplex 12.4 was used as an LP solver: using Cplex default

option that automatically selects between primal and dual Simplex. Unless said otherwise, the global time limit is set to 10 minutes per instance.

6.1 Generalized assignment

In the generalized assignment problem (GAP), we are given n tasks to be assigned to m machines (or agents), where each machine i has capacity u_i, and each task j, when assigned to machine i, uses d_{ij} units of resource and costs c_{ij}, for $j = 1, \ldots, n$, and $i = 1, \ldots, m$. The problem consists in assigning each task to exactly one machine, such that the total resource usage on each machine does not exceed its capacity, and the total assignment cost is minimized. This application involves **multiple distinct subproblems**. A solution of the pricing subproblem consists of a set of tasks to be assigned to one of the machines, that satisfies the capacity constraint. To solve this binary knapsack pricing subproblem, we use the *minknap* solver of Pisinger [43]. This oracle can by downloaded from `http://www.diku.dk/~pisinger/minknap.c`. As the solver assumes integer data, our input is transformed by rounding after multiplying by a suitable factor. Hence our oracle might be heuristic rather than exact, but this is acceptable since our aim is to produce a heuristic solution. We use the "triple" stabilization procedure of [42], combining Wentges smoothing, directional smoothing, and 3-piecewise linear penalty function to improve convergence of our column generation procedure.

For the GAP, there exists a library of "classic" instances of types A, B, C, D, E with up to $n = 1600$ tasks and $m = 80$ machines. They can be downloaded from `http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap/`. These instances were used, for example, in [57]. As for a pair (m, n) there is only one instance in the library, this was not enough to run an experiment to compare heuristics between them. Hence, we randomly generated instances of the two most difficult types D and E. We applied the same generation procedure as that used for the instances in the library. We generated instances of type D for sizes $(n, m) = \{(90, 18), (160, 8)\}$, and of type E for sizes $(m, n) = \{(200, 40), (400, 20)\}$. For each type and each pair (n, m), we generated 10 instances. For such instance sizes, we were able to obtain, for comparison purposes, an optimal solution in a reasonable time using our own Branch-and-Price code.

Our numerical results are presented in Table 5: "Time" is the average running time of the heuristic in seconds; "Found" is the percentage of instances for which a feasible solution was found by the heuristic, "Opt" is the percentage of instances for which an optimal solution was found by the heuristic, and "RelGap" is the average relative gap between the solution value

found by the heuristic and the optimal solution value. Note that "Time" includes both the time to initially solve the master LP and the time to perform the heuristic diving. Also observe that "RelGap" appears to be small, but this is due to the large cost of any feasible solution: the scope for optimization concerns only a marginal part of the global cost.

		Instances of type D				Instances of type E			
	Heuristic	Time	Found	Opt	RelGap	Time	Found	Opt	RelGap
1.	Pure Diving	0.80	70%	0%	0.37%	9.77	90%	15%	0.053%
2.	Diving for Feasibility	0.81	100%	0%	0.39%	9.84	100%	15%	0.056%
3.	Diving with Restarts	1.52	100%	0%	0.24%	12.31	100%	15%	0.029%
4.	Local Branching	1.90	100%	0%	0.38%	25.78	100%	20%	0.032%
5.	Diving with LDS	4.21	100%	5%	0.10%	36.84	100%	45%	0.010%
6.	Strong Diving	33.45	100%	15%	0.05%	375.84	100%	45%	0.009%
7.	Restricted Master	26.50	55%	0%	11.00%	388.86	100%	0%	3.221%
8.	Diving + SubMIPing	40.22	100%	0%	0.38%	259.29	100%	25%	0.017%

Table 5: Results for the generated instances of the Generalized Assignment Problem

In Table 5, the worst performance is clearly that of the restricted master heuristic: it found the least number of feasible solutions, and where a solution was obtained, the average gap is much larger than for other heuristics. The fastest heuristic is pure diving, but it does not always find a feasible solution. Diving for feasibility manages to "correct" this drawback of the pure diving heuristic with almost no additional time. Diving with restarts and local branching heuristics improve over diving for feasibility but require more running time. Diving with LDS improves significantly the quality of obtained solution but requires even more running time. The best solution quality is obtained by the strong diving heuristic, but it consumes an order of magnitude more time than the previous heuristic. Diving with subMIPing does not compare favourably here with other heuristics, as the quality of solutions is worse than by diving with LDS, but the running time is significantly larger. Overall, we conclude that the diving with LDS heuristic offers the best tradeoff between quality and running time. However, if one needs a fast heuristic, diving for feasibility can provide high quality solutions in small time.

In the next experiment, we tested the diving heuristic with LDS on the "classic" instances from Yagiura's library. Only instances of types C, D, and E were used in previous literature, as, f.i., in [44], because instances of types A and B are very easy. In Table 6, we compare our heuristic with the path relinking algorithm with ejection chains by Yagiura et al. [57], which

is the best available specialized heuristic for the problem (according to our knowledge of the literature). The results are grouped by instances type, by instance size (the number of tasks), and by ratio n/m between the number of tasks and the number of machines. High ratio instances are with $(n, m) \in \{(100, 5), (200, 5), (400, 10), (900, 15), (1600, 20)\}$, medium ration instances are with $(n, m) \in \{(100, 10), (200, 10), (400, 20), (900, 30), (1600, 40)\}$, and low ratio instances are with $(n, m) \in \{(100, 20), (200, 20), (400, 40), (900, 60), (1600, 80)\}$. In Table 6, "Time" gives the geometric mean of the running time, "Opt" is the percentage of optimal solutions found. The columns "RelGap" and "AbsGap" report the average relative and absolute gaps between the solution value found by the heuristic and the optimal solution value (or the best known lower bound reported in [44] if the optimal value is not known). Note that the computer used by Yagiura et al. is approximately 16 times slower than ours [1]. Therefore, the running time of their algorithm reported in the table is their actual running time that we have divided by 16.

	Yagiura et al. [57]				Diving heuristic with LDS			
Group	Time	Opt	RelGap	AbsGap	Time	Opt	RelGap	AbsGap
Type C	145.1	53%	0.010%	0.9	30.0	47%	0.015%	0.7
Type D	145.1	7%	0.103%	21.1	69.5	7%	0.047%	8.5
Type E	145.1	33%	0.013%	6.7	38.1	47%	0.014%	3.2
$n = 100$	9.4	67%	0.073%	4.8	1.4	44%	0.045%	3.4
$n = 200$	18.8	44%	0.045%	5.3	6.1	11%	0.054%	6.0
$n = 400$	187.5	33%	0.051%	12.8	40.2	44%	0.017%	4.1
$n = 900$	625.0	0%	0.029%	14.7	291.1	33%	0.006%	3.0
$n = 1600$	3125.0	11%	0.011%	10.2	1500.7	33%	0.006%	4.1
high n/m	145.1	47%	0.006%	2.5	19.1	33%	0.023%	3.3
med n/m	145.1	27%	0.031%	7.1	46.7	27%	0.025%	3.7
low n/m	145.1	33%	0.089%	19.1	88.7	40%	0.029%	5.3
All	145.1	31%	0.042%	9.6	43.0	33%	0.026%	4.1

Table 6: Comparison with the literature on the library instances of the Generalized Assignment Problem.

From the result summary reported in the last line of Table 6, we can conclude that our algorithm is faster and obtains better solutions on average. Of course, the comparison of running times is very rough. Results grouped by different instance characteristics show that the diving

[1] according to http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance

heuristic outperforms significantly the algorithm by Yagiura et al. 1) for instances of the most difficult type D, 2) for large-size instances, and 3) for instances with low n/m ratio.

According to [44], there are ten open instances in this set. We have tried to improve the best known solutions for these instances. For this, we did seven runs of the diving heuristic with LDS. For each run, we set different parameters *maxDiscrepancy*, *maxDepth* (always ensuring that the total number of dives is equal to ten), different maximum number of columns in the master (master clean-up), and different parameter κ for the penalty function stabilization (see [42]). It was already mentioned above that the master LP usually admits many alternative solutions. Therefore, a change of the master clean-up and stabilization parameters typically change the master LP solution obtained by column generation. Thus, the diving heuristics explore different parts in the search tree.

	Best known ([44])		Best run (instance specific)			Average (seven runs)	
Instance	Bound	Solution	Solution	Time	Reduced gap	Time	Reduced gap
D-20-200	12235	12244	12238	18	66%	22	3%
D-20-400	24563	24585	24567	82	82%	82	56%
D-40-400	24350	24417	24356	134	89%	145	72%
D-15-900	55404	55414	**54404**	80	100%	179	43%
D-30-900	54834	54868	54838	529	88%	505	61%
D-60-900	54551	54606	54554	1445	95%	1490	83%
D-20-1600	97824	97837	97825	744	92%	665	69%
D-40-1600	97105	97113	**97105**	3158	100%	7314	38%
D-80-1600	97034	97052	97035	10852	94%	10856	-48%
C-80-1600	16284	16289	16285	2186	80%	2572	80%

Table 7: Improved best known solutions for open instances of the Generalized Assignment Problem.

In Table 7, for each instance, from left to right we report the instance name, the best known lower bound and solution value reported by [44], the best solution value obtained by us, the time in seconds taken to obtain this value, the reduction in gap between the bound and the best known solution obtained by us, the average solution obtained over all seven runs, and the average running time. We have managed to improve the best known solutions for all ten open instances. Moreover, the average reduction of the gap between the best known solution and the best known lower bound is 89%. For two instances D-15-900 and D-40-1600, optimal solutions

were found for the first time. The average solution value obtained over the seven runs is smaller than the previously best known solution for all instances except one.

6.2 Cutting stock

In the cutting stock problem (CSP), we are given n items to be put to bins of size S, each item i has d_i copies of size s_i, for $i = 1, \ldots, n$. The problem consists in assigning each item copy to exactly one bin such that the total size of all item copies assigned to a bin does not exceed S, and the number of bins used is minimized. This application involves **multiple identical subproblems**. A solution of the pricing subproblem prescribes the number of copies of each item assigned to one bin so as to obey the size constraint. To solve this integer knapsack pricing subproblem, we use the *minknap* solver of Pisinger [43], having transformed the instance of the bounded integer knapsack problem into an equivalent instance of a binary knapsack problem (by binary expansion). The "double" stabilization of [42] combining Wentges smoothing and directional smoothing was applied to improve convergence of our column generation procedure.

For the experiment, we used instances collected in the recent survey on the bin packing and cutting stock problems [20]. The instances can be downloaded from `http://or.dei.unibo.it/library/bpplib`. We grouped instances in four sets: our set "Falkenauer" contains 80 instances of set "Falkenauer T" in [20]; our set "Hard" contains 55 instances of sets "Scholl 3", "Wäscher", and "Hard28" in [20]; our set "AI" contains 100 instances of set "AI" with $n = 201$ and $n = 402$ in [20]; our set "ANI" contains 50 instances of set "ANI" with $n = 201$ in [20]. According to [20], for all instances except for the set "ANI", the optimal solution value is equal to the lower bound value obtained by rounding up the column generation master LP bound. For the instances in set "ANI", the gap between the optimal solution value and this lower bound is one.

The results are presented in Table 8, except for the Diving for feasibility, given that feasibility is not an issue in the CSP with a free number of bins. Also, there is no need to report on "Found" as all the heuristics found a feasible solution for all instances. The columns have the same meaning as in Table 5, except the column "AbsGap" which is the average absolute gap between the solution value found by the heuristic and the optimal solution value. One can observe that instances in set "ANI" are easy for all heuristics, as optimal solutions were obtained by all of them. However, it is very hard to prove optimality for them. For other instances, the restricted master heuristic has worse performance than diving based heuristics. The diving with restarts, local branching and diving with subMIPing heuristics showed better results than the

pure diving heuristic, but not significantly better. Among these three heuristics, the diving with restarts heuristic is the fastest one and seems to offer the best trade-off between solution quality and running time. The best performance are obtained with the diving with LDS heuristic and and the strong diving heuristic. However, the latter requires larger times for instances in sets "Hard" and "AI". Note that the strong diving heuristic solved all the instances in set "Falkenauer" to provable optimality (as they all have the zero gap) in 7 seconds which is faster than any known column generation based algorithm according to [20]. Only 10 instances in set "AI" could not be solved in 10 minutes by the diving heuristic, whereas the best known algorithm could not solve 5 instances in this set in 1 hour according to [20].

		Instances "Falkenauer"			Instances "Hard"		
	Heuristic	Time	Opt	AbsGap	Time	Opt	AbsGap
1.	Pure Diving	5.16	57%	0.42	3.68	51%	0.49
3.	Diving with Restarts	5.47	68%	0.33	3.84	56%	0.44
4.	Local Branching	78.50	66%	0.34	15.28	60%	0.40
5.	Diving with LDS	7.94	99%	0.01	4.67	76%	0.24
6.	Strong Diving	7.28	100%	0.00	14.90	76%	0.24
7.	Restricted Master	302.71	20%	3.30	137.93	4%	1.22
8.	Diving + SubMIPing	145.49	68%	0.33	15.88	65%	0.35
		Instances "AI"			Instances "ANI"		
	Heuristic	Time	Opt.	AbsGap	Time	Opt.	AbsGap
1.	Pure Diving	13.71	46%	0.54	2.64	100%	0.00
3.	Diving with Restarts	14.83	51%	0.49	2.93	100%	0.00
4.	Local Branching	44.40	52%	0.48	3.01	100%	0.00
5.	Diving with LDS	27.44	89%	0.11	4.75	100%	0.00
6.	Strong Diving	67.42	90%	0.10	33.34	100%	0.00
7.	Restricted Master	224.37	5%	1.22	2.70	100%	0.00
8.	Diving + SubMIPing	85.49	53%	0.47	3.03	100%	0.00

Table 8: Results for the Cutting Stock Problem

6.3 Vertex coloring

In the vertex coloring problem, we are given a simple undirected graph with n vertices. We need to label each vertex of the graph with a color such that no two adjacent vertices share the

same color. The objective is to use a minimum number of colors. This application involves **multiple identical subproblems**. A solution of the pricing problem consists in choosing a set of vertices which are non-adjacent to each other. So, the pricing problem here is a weighted stable set problem, or the weighted clique problem once seen in the complementary graph. To solve the pricing problem, we used the *cliquer* solver by Östergård [39] (which can be downloaded from `http://users.aalto.fi/~pat/cliquer.html`).

To compare the heuristics between them, we generated random instances with $n \in \{50, 60, 70, 80, 90\}$ and with density $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. When generating a random graph with density p, each edge is added to the graph with probability p. For each pair (n, p), we generated 5 instances. This makes the total number of generated random instances equal to 125. The second set of instances contains random regular instances. A random regular instance generated with parameters (n, p) contains n vertices, each of which has degree d, where $d = \lfloor p \cdot (n-1) \rfloor$. Again, we generated 5 instances for each pair (n, p) such that $n \in \{50, 60, 70, 80, 90\}$ and $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ making the number of generated random regular instances equal to 125. Optimal solutions for all generated instances were obtained using our own Branch-and-Price code.

Table 9 reports our results. The columns have the same meaning as above. Again, all the heuristics found a feasible solution for all instances, and thus results for diving for feasibility are not presented. The results are somewhat similar to the ones for the cutting stock problem. The worst performances are those of the restricted master heuristic. The two best heuristics are the diving with LDS and the strong diving. The diving heuristics with restarts and the local branching heuristic have a marginal improvement over the pure diving heuristic. The diving heuristic with subMIPing offers a larger improvement but takes more time than the diving with LDS. Note that there is a small difference between the running times of different heuristics for the random regular instances. The reason is that most of the running time is spent for the initial optimization of the master LP.

In the next experiment, we compared the diving heuristic with LDS with recent specialized heuristics available in the literature. In [26], the best such heuristics were reviewed and compared on a set of difficult instances. We run the diving heuristic with LDS on the same set of instances with up to $n = 1000$ vertices. Note that, due to a large size of instances, an exact algorithm cannot be always applied to solve the pricing problem. Therefore, in this experiment, we used the local search heuristic from [31] to solve the pricing problem. Also, the Wentges dual price smoothing technique with automatic parameter setting [42] was applied to improve

		Random instances			Random regular instances		
	Heuristic	Time	Opt	AbsGap	Time	Opt	AbsGap
1.	Pure Diving	0.94	71%	0.29	23.85	59%	0.41
3.	Diving with Restarts	1.06	74%	0.26	23.92	66%	0.34
4.	Local Branching	1.00	74%	0.26	24.15	63%	0.37
5.	Diving with LDS	1.38	88%	0.12	24.34	81%	0.19
6.	Strong Diving	3.65	94%	0.06	27.36	88%	0.12
7.	Restricted Master	3.94	49%	0.54	26.09	46%	0.56
8.	Diving + SubMIPing	1.93	81%	0.19	25.34	70%	0.30

Table 9: Results for the Vertex Coloring Problem

the convergence of column generation. As it is common in the vertex coloring literature (see, for example, in [52]), we set the time limit to 5 hours for solving one instance.

In Table 10, for each instance, from left to right we report the instance name, the number of vertices n, the density p of the graph, the solution value found by the diving heuristic with LDS, its running time, the best known solution value in the literature. In the last three columns, using [26] we report the number of recent heuristics in the literature which found a better, an equal, and a worse solution than our heuristic.

From the table we can see that the diving heuristic with LDS is clearly not the best heuristic for the vertex coloring problem. However, its overall performance is quite satisfactory. For a large majority of instances, the solution found is not more than one unit away from the best known solution. On some instance, a majority of recent heuristics performed worse than the diving heuristic with LDS.

7. Conclusion

To benefit from the tighter linear relaxation resulting from a Dantzig-Wolfe reformulation, we developed diving heuristic schemes that are amenable to a column generation context. Additional diversification and intensification features that we built on the basic diving procedure have been shown to bring a much better chance to identify a feasible solution when this was an issue and/or to enhance the solution quality with limited extra computational burden. Our recommendation is to use diving with LDS as it proved to have good performance across applications; it relies on "diving for feasibility" to identify a first incumbent. If one is prepared to trade longer

Instance			Diving with LDS		Best	Literature heuristics are		
Name	n	p	Solution	Time	known	Better	Equal	Worse
dsjc1000.1	1000	0.10	21	5h00m	20	11	0	0
le450.25c	450	0.17	26	0h45m	25	9	4	0
le450.25d	450	0.17	26	0h43m	25	9	3	0
dsjr500.5	500	0.47	122	0h03m	122	0	5	6
r1000.5	1000	0.48	234	0h51m	234	0	2	7
flat300.28_0	300	0.48	29	0h05m	28	4	2	8
flat1000.76_0	1000	0.49	83	5h00m	82	6	1	7
dsjc500.5	500	0.50	49	1h08m	48	10	4	0
dsjc1000.5	1000	0.50	84	5h00m	83	7	2	6
latin_square_10	900	0.76	100	5h00m	97	6	0	3
dsjc500.9	500	0.90	127	0h41m	126	9	1	0
dsjc1000.9	1000	0.90	226	5h00m	222	11	1	0
dsjr500.1c	500	0.97	85	0h01m	85	0	8	1
r1000.1c	1000	0.97	98	1h15m	98	0	7	2

Table 10: Comparison with the literature on the library instances of the Vertex Coloring Problem

computing time for better solution quality, we recommend using our variant relying on strong branching, which builds over diving with LDS. It proved to be an asset on harder combinatorial problems such as Vertex Coloring. In any case, our diving heuristics are comparatively much better than the most commonly used heuristic of the column generation literature, namely the restricted master heuristic otherwise known as price-and-branch: our results show that the latter provides the worst performance in terms of trade-off between solution quality and running time.

Where we could compare our results to other column generation based heuristics, as for the cutting stock problem (referring to the review paper of [20]), we obtained better performance. Even more, our generic diving procedures are very competitive compared with tailored primal heuristics that were specifically designed for combinatorial applications. This is the case for Generalized Assignment. Or at least we tend to match existing performance, as is the case for Vertex Coloring. And last, but not least, using column generation based diving, we have managed to improve the best known solutions for several open Generalized Assignment instances of the literature.

The genericity of our primal heuristic procedures rely on exploiting the link between original

formulation and its Dantzig-Wolfe reformulation. We make the assumption that the pricing oracle is able to handle the bounded version of the pricing problem, in order to only generate proper columns. Note that enforcing bounds on subproblem variables can lead to a more complex subproblem in a few applications. For example, the 2 dimensional guillotine cutting-stock problem leads to a subproblem that can by solved by dynamic programming in pseudo-polynomial time [32], only if there are no upper bounds on the maximum number of copies of an item in a cutting pattern. Such limitation is similar to the assumption on which relies the generic branching scheme of [54]. A standard approach to bypass this issue is to rely on a heuristic oracle to generate proper columns during the diving heuristic phase, while using an exact oracle for the unbounded problem during the initial master LP solution phase. Also note that, in the presence of different subproblem types, there are branching priority issue that would not be generic.

There are further directions to explore using the methods presented herein. An issue for further research is to consider extending our heuristics to more complex master program involving not only subproblem solution selection variables but also additional decision variables (so-called pure master variables). Primal heuristics would then combine strategies developed herein with standard MIP diving procedures.

Aknowledments: This research was supported by the associated team program of Inria through the SAMBA project. The detailed comments of the referees have been greatly useful in pointing the need for further explanations in our original manuscript. We are very grateful to them for their involvement.

References

[1] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

[2] Y. Agarwal, K. Mathur, and H. M. Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989.

[3] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[4] Laurent Alfandari, Agnès Plateau, and Xavier Schepler. A branch-and-price-and-cut approach for sustainable crop rotation planning. *European Journal of Operational Research*, 241(3):872 – 879, 2015.

[5] Enrico Angelelli, Nicola Bianchessi, and Carlo Filippi. Optimal interval scheduling with a resource constraint. *Computers and Operations Research*, 51:268 – 281, 2014.

[6] G. Belov and G. Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141(2):274–294, 2002.

[7] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63 – 76, 2007.

[8] T. Berthold. Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.

[9] N. Bianchessi, R. Mansini, and M.G. Speranza. The distance constrained multiple vehicle traveling purchaser problem. *European Journal of Operational Research*, 235(1):73 – 87, 2014.

[10] B. Bixby. Presentation of the Gurobi Optimizer. Integer Programming Down Under: Theory, Algorithms and Applications, Workshop at Newcastle NSW, 2011.

[11] V. Cacchiani, V.C. Hemmelmayr, and F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 163, Part 1:53 – 64, 2014.

[12] A. Ceselli, G. Righini, and M. Salani. A column generation algorithm for a vehicle routing problem with economies of scale and additional constraints. In *Proceedings TRISTAN*, Phuket, Thailand, june 2007.

[13] A. Chabrier, E. Danna, and C. Le Pape. 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)*, pages 83–97, Nice, France, may 2002.

[14] Alain Chabrier, Emilie Danna, Claude Le Pape, and Laurent Perron. Solving a network design problem. *Annals of Operations Research*, 130(1-4):217–239, 2004.

[15] GF Cintra, Flávio Keidi Miyazawa, Yoshiko Wakabayashi, and EC Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191(1):61–85, 2008.

[16] Fabio Colombo, Roberto Cordone, and Marco Trubian. Column-generation based bounds for the homogeneous areas problem. *European Journal of Operational Research*, 236(2):695 – 705, 2014.

[17] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1, Ser. A):71–90, 2005.

[18] Toby Davies, Adrian Pearce, Peter Stuckey, and Harald Sondergaard. Fragment-based planning using column generation. In *24th International Conference on Automated Planning and Scheduling*, 2014.

[19] Z. Degraeve and R. Jans. A new Dantzig-Wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times. *Oper. Res.*, 55(5):909–920, 2007.

[20] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. Research report OR-15-1, DEI "Guglielmo Marconi", University of Bologna, 2015.

[21] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with non-negative data. *Mathemathics of Operations Research*, 7(4):515–531, 1982.

[22] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1, Ser. A):91–104, 2005.

[23] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3, Ser. B):23–47, 2003.

[24] M. Fischetti and A. Lodi. *Heuristics in Mixed Integer Programming*, volume 3 of *Wiley Encyclopedia of Operations Research and Management Science, J.J. Cochran Edt*, pages 2199–2204. Wiley, 2011.

[25] M. Fischetti and D. Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1:201–222, 2009.

[26] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. Recent advances in graph vertex coloring. In Ivan Zelinka, Václav Snášel, and Ajith Abraham, editors, *Handbook of Optimization*, volume 38 of *Intelligent Systems Reference Library*, pages 505–528. Springer Berlin Heidelberg, 2013.

[27] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47(2):247–263, 1999.

[28] Martin Grötschel, Ralf Borndörfer, and Andreas Löbel. Duty scheduling in public transit. In Willi Jäger and Hans-Joachim Krebs, editors, *Mathematics — Key Technology for the Future*, pages 653–674. Springer Berlin Heidelberg, 2003.

[29] O. Gunluk, T. Kimbrel, L. Ladanyi, B. Schieber, and G. B. Sorkin. Vehicle Routing and Staffing for Sedan Service. *Transportation Science*, 40(3):313–326, 2006.

[30] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proc. IJCAI-95, Montreal, Quebec*, pages 607–613. Morgan Kaufmann, 1995.

[31] Stephan Held, William Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

[32] JC Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16(5):462–469, 1972.

[33] Cédric Joncour, Sophie Michel, Ruslan Sadykov, Dmitry Sverdlov, and François Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695 – 702, 2010. International Symposium on Combinatorial Optimization.

[34] Daniel Kowalczyk and R. Leus. An exact algorithm for parallel machine scheduling with conflicts. Technical report, Social Science Research Network, 2015.

[35] Hugo H. Kramer, Vinicius Petrucci, Anand Subramanian, and Eduardo Uchoa. A column generation approach for power-aware optimization of virtualized heterogeneous server clusters. *Computers and Industrial Engineering*, 63(3):652 – 662, 2012.

[36] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[37] Marco Lübbecke and Christian Puchert. Primal heuristics for branch-and-price algorithms. In Diethard Klatte, Hans-Jakob Lüthi, and Karl Schmedders, editors, *Operations Research Proceedings 2011*, Operations Research Proceedings, pages 65–70. Springer Berlin Heidelberg, 2012.

[38] S. Michel and F. Vanderbeck. A column-generation based tactical planning method for inventory routing. *Operations Research*, 60(2):382–397, 2012.

[39] P. R. J. Östergård. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing*, 8:424–436, December 2001.

[40] N. Perrot. *Integer Programming Column Generation Strategies for the Cutting Stock Problem and its Variants*. PhD thesis, Université Bordeaux 1, France, 2005.

[41] Pierre Pesneau, Ruslan Sadykov, and François Vanderbeck. Feasibility pump heuristics for column generation approaches. In Ralf Klasing, editor, *Experimental Algorithms*, volume 7276 of *Lecture Notes in Computer Science*, pages 332–343. Springer Berlin Heidelberg, 2012.

[42] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. Technical Report hal-01077984, HAL Inria, 2014.

[43] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[44] Marius Posta, Jacques A. Ferland, and Philippe Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52:629–644, 2012.

[45] Ali Gul Qureshi, Eiichi Taniguchi, and Tadashi Yamada. Column generation-based heuristics for vehicle routing problem with soft time windows. *Journal of the Eastern Asia Society for Transportation Studies*, 8:827–841, 2010.

[46] Ruslan Sadykov, Alexander A. Lazarev, Artur Pessoa, Eduardo Uchoa, and François Vanderbeck. The prominence of stabilization techniques in column generation: the case of freight transportation. In *Sixth International Workshop on Freight Transportation and Logistics*, Ajaccio, France, 2015.

[47] Ruslan Sadykov and François Vanderbeck. Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013.

[48] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, and Eduardo Uchoa. Column generation based heuristic for the generalized assignment problem. In *XLVII Simpósio Brasileiro de Pesquisa Operacional*, Porto de Galinhas, Brazil, 2015.

[49] Verena Schmid, Karl F. Doerner, Richard F. Hartl, Martin W. P. Savelsbergh, and Wolfgang Stoecher. A hybrid solution approach for ready-mixed concrete delivery. *Transportation Science*, 43(1):70–85, 2009.

[50] Remy Spliet and Guy Desaulniers. The discrete time window assignment vehicle routing problem. *European Journal of Operational Research*, 244(2):379 – 391, 2015.

[51] É. D. Taillard. A heuristic column generation method for the heterogeneous fleet VRP. *RO Oper. Res.*, 33(1):1–14, 1999.

[52] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376 – 384, 2011.

[53] F. Vanderbeck. chapter Implementing Mixed Integer Column Generation, pages 331–358. Kluwer's series in Operations Research. Kluwer, 2005.

[54] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130:249–294, 2011.

[55] F. Vanderbeck and M. W. P. Savelsbergh. A generic view of dantzig-wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.

[56] François Vanderbeck and Laurence Wolsey. Reformulation and Decomposition of Integer Programs. In M. Jünger, Th.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*. Springer, 2010.

[57] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2):548 – 569, 2006.