Math. Program., Ser. A
DOI 10.1007/s10107-006-0079-z

FULL LENGTH PAPER

Comparison of bundle and classical column generation

O. Briant · C. Lemaréchal · Ph. Meurdesoif · S. Michel · N. Perrot · F. Vanderbeck

Received: 21 January 2005 / Accepted: 31 October 2006

Abstract When a column generation approach is applied to decomposable mixed integer programming problems, it is standard to formulate and solve the master problem as a linear program. Seen in the dual space, this results in the algorithm known in the nonlinear programming community as the cutting-plane algorithm of Kelley and Cheney-Goldstein. However, more stable methods with better theoretical convergence rates are known and have been used as alternatives to this standard. One of them is the bundle method; our aim is to illustrate its differences with Kelley's method. In the process we review alternative stabilization techniques used in column generation, comparing them from both primal and dual points of view. Numerical comparisons are presented for five applications: cutting stock (which includes bin packing), vertex coloring, capacitated vehicle routing, multi-item lot sizing, and traveling salesman. We also give a sketchy comparison with the volume algorithm.

Keywords Lagrangian duality · Dantzig–Wolfe decomposition · Stabilized column generation · Cutting-plane algorithms · Bundle algorithm · Volume algorithm · Nonsmooth convex optimization

This research has been supported by Inria New Investigation Grant "Convex Optimization and Dantzig-Wolfe Decomposition".

O. Briant
Gilco, 46 avenue Félix Viallet, 38000 Grenoble, France

C. Lemaréchal (✉)
INRIA, 655 avenue de l'Europe, Montbonnot, 38334 Saint Ismier, France
e-mail: claude.lemarechal@inrialpes.fr

Ph. Meurdesoif · S. Michel · N. Perrot · F. Vanderbeck
University of Bordeaux 1; MAB, 351 cours de la Libération, 33405 Talence, France

Mathematics Subject Classification (2000) 66K05 · 90C25 · 90C27

1 Algorithms for column generation

This paper deals with optimization problems of the form

$$\min cx, \qquad Ax \geqslant b \in \mathbb{R}^m, \quad x \in X := \{x^i : i \in I\} \subset \mathbb{R}^n, \tag{1.1}$$

where the index set I is assumed finite (think of X as the set of integer points in a bounded polyhedron). An equivalent formulation is Dantzig-Wolfe's (integer) *master problem*

$$\min \sum_{i\in I} (cx^i)\lambda_i, \qquad \sum_{i\in I} (Ax^i)\lambda_i \geqslant b, \quad \sum_{i\in I} \lambda_i = 1, \quad \lambda_i \in \{0,1\}, \ \ i \in I,$$

in which the variable vector is $\lambda \in \{0,1\}^{|I|}$.

When solving such problems, it is standard to make use of a lower bound obtained by *relaxation*. Denoting by conv (X) the convex hull of the set X, the present paper is rather devoted to solving

$$\min cx, \qquad Ax \geqslant b \in \mathbb{R}^m, \quad x \in \text{conv}\,(X) \tag{1.2}$$

or its Dantzig–Wolfe formulation: the (linear) master problem

$$\min \sum_{i\in I} (cx^i)\lambda_i, \qquad \sum_{i\in I} (Ax^i)\lambda_i \geqslant b, \quad \sum_{i\in I} \lambda_i = 1, \quad \lambda_i \geqslant 0, \ \ i \in I. \tag{1.3}$$

Remark 1.1 Needless to say, the points in conv (X) have the form $\sum_i \lambda_i x^i$, with λ varying in the unit simplex.

Our framework could handle the unbounded case as well, considering a finite set of rays c^j: conv (X) would have the form $\sum_i \lambda_i x^i + \sum_j \mu_j c^j$, the μ_j being just nonnegative (not restricted to sum up to 1); X could also be a mixed-integer polyhedron; or also be the Cartesian product of several subsets, i.e. $X = \prod_\ell X^\ell$ (when the constraint matrix defining X has a block diagonal structure, say).

Some of these "extensions" shall be illustrated when we come to specific applications in Sect. 2. It is mainly for simplicity that here we restrict our presentation to a single-block bounded polyhedron. □

Our aim is to solve the Dantzig–Wolfe formulation by column generation. At the current step k of the process, a *restricted master problem* is solved, obtained by restricting I to some $I^k \subset I$ in (1.2) or (1.3); we will set correspondingly $X^k := \{x^i : i \in I^k\}$. This resolution provides essentially two outputs:

- a primal solution $\hat{x} = \sum \hat{\lambda}_i x^i$, which is a candidate to solving (1.2) or (1.3),
- a dual solution $(u^k, r^k) \in \mathbb{R}^m_+ \times \mathbb{R}$ associated with constraints $Ax \geqslant b$ and $\sum_i \lambda_i = 1$, respectively.

Then, the dual solution is used to price out columns in $I \setminus I^k$. For this, an optimisation subproblem or *oracle* is called upon to provide new columns of negative reduced cost, if any: for the given $u = u^k$, we minimize (possibly approximately) $(c - u^k A)x^i$ for i ranging over the whole of I, i.e. we consider the problem

$$\min_{x \in X} (c - uA)x \tag{1.4}$$

at $u = u^k$, where X can equally be replaced by $\text{conv}\,(X)$.

From now on we will assume for simplicity that the oracle is exact and that $I^k = \{1, 2, \ldots, k\}$: for a given u, the oracle solves (1.4) exactly, and provides just one optimal column, call it $x(u)$, to be appended to X^k.

1.1 Standard column generation

The traditional—and most natural—restricted master problem is just (1.2) or (1.3) with I replaced by I^k: we solve

$$\min cx, \qquad Ax \geqslant b \in \mathbb{R}^m, \quad x \in \text{conv}\,(X^k), \tag{1.5}$$

which is the linear program

$$\min \sum_{i=1}^{k} \lambda_i c x^i, \qquad \sum_{i=1}^{k} \lambda_i A x^i \geqslant b, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad \lambda_i \geqslant 0, \;\; i = 1, \ldots, k. \tag{1.6}$$

Note that feasibility must be enforced: the initial (1.5) or (1.6) must contain a number of columns (possibly artificial) having in their convex hull some x satisfying $Ax \geqslant b$.

Suppose (1.5) or (1.6) has indeed a solution $\hat{x}$ or $\hat{\lambda}$ with

$$\hat{x} = \sum \hat{\lambda}_i x^i, \tag{1.7}$$

and multipliers (u^k, r^k). To check whether $\hat{x}$ solves (1.2), we implicitly do full pricing by solving (1.4) to obtain $x(u^k)$. Two cases can arise:

(i) Either $x^{k+1} := x(u^k)$ has negative reduced cost: $(c - u^k A)x^{k+1} + r^k < 0$; this new column is appended to the "bundle" $x^1, \ldots, x^k$ and the process is repeated, (note: with our definition of (u^k, r^k), a negative reduced cost implies that x^{k+1} does not lie in $\text{conv}\,(X^k)$).

(ii) Or $x(u^k)$ has zero reduced cost: then all columns in I have nonnegative reduced cost and (1.2) or (1.3) is solved.

Remark 1.2 Some tolerance can be inserted, to stop the algorithm when the smallest reduced cost is "not too negative", i.e. $(c - u^k A)x^{k+1} + r^k \geqslant -\varepsilon$. Early termination based on such tolerance results in approximate optimality; this is the subject of Remark 1.3 below, where the concept of ε-optimality shall be defined. □

Insofar as we aim at solving (1.2), $\hat{x}$ plays its role, of course. However, it is important to understand that the key role is actually played by u^k which, via the column x^{k+1}, really drives the algorithm toward convergence. Good u^k's are those that are close to optimal multipliers for (1.2) or (1.3). It is therefore of interest to analyze (1.5) or (1.6) in the dual space, the LP dual of (1.6) being the "dual restricted master"

$$\max ub - r, \qquad uAx^i - r \leqslant cx^i, \quad i = 1, \ldots, k, \quad (u, r) \in \mathbb{R}^m_+ \times \mathbb{R}. \tag{1.8}$$

1.2 Column generation seen in the dual space

For subsequent use, it is convenient to free oneself from the LP formalism and to use general duality (see for example [16], [22, Chap. XII], [38, Sect. 1], [37, Sect. 1.2]).

To (1.2) we associate the Lagrange function

$$\mathrm{conv}\,(X) \times \mathbb{R}^m \ni (x, u) \mapsto L(x, u) := cx + u(b - Ax) = (c - uA)x + ub, \tag{1.9}$$

and the so-called *dual function*

$$\begin{aligned} \mathbb{R}^m \ni u \mapsto \theta(u) &:= \min_{x \in X} L(x, u) = \min_{x \in \mathrm{conv}\,(X)} L(x, u) \\ &= (c - uA)x(u) + ub; \end{aligned} \tag{1.10}$$

in the last expression, recall that $x(u)$ denotes the answer of the oracle, i.e., an argmin in (1.4). The dual problem associated with (1.2) is then

$$\max \{\theta(u) : u \in \mathbb{R}^m_+\}, \tag{1.11}$$

which is the linear program

$$\max_{(u,s) \in \mathbb{R}^m_+ \times \mathbb{R}} \left\{ s : s \leqslant ub + (c - uA)x^i, \quad i \in I \right\}. \tag{1.12}$$

The left part of Fig. 1 illustates this: the thick line represents the dual function θ, which is the lower envelope of all the possible $L(x, \cdot)$'s, for x describing X—or $\mathrm{conv}\,(X)$, which does not change the picture.

Performing the same operations on the restricted master, we introduce likewise the *restricted dual function* (right part of Fig. 1, where $k = 3$))

$$\mathbb{R}^m \ni u \mapsto \theta^k(u) := \min_{x \in \mathrm{conv}\,(X^k)} L(x, u) = ub + \min_{i=1,\ldots,k} (c - uA)x^i, \tag{1.13}$$

which overestimates θ: $\theta^k(u) \geqslant \theta(u)$ for all u. The dual of (1.5) is then

$$\max \left\{ \theta^k(u) : u \in \mathbb{R}^m_+ \right\}, \tag{1.14}$$

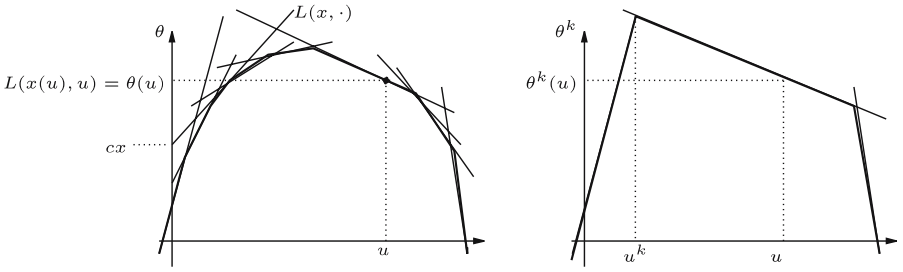

Fig. 1 The dual function θ (*left*) and its restricted version θ^k for $k=3$ (*right*)

i.e.,

$$\max_{(u,s)\in\mathbb{R}^m_+\times\mathbb{R}} \{s : s \leqslant ub + (c-uA)x^i, \quad i=1,\dots,k\}. \tag{1.15}$$

Setting $s = ub - r$, we recognize in this latter program the dual (1.8) of (1.6): s (1.8) [resp. r] stands for $\theta^k(u)$ [resp. $ub-\theta^k(u)$]; r can be understood as a fixed cost in the primal oracle (1.4). Observe here and now that u solves (1.14) when $(u,\theta^k(u))$ solves (1.15), or equivalently when $(u, ub-\theta^k(u))$ solves (1.8).

Column generation for (1.6) is row generation for its dual (1.8). Thus, the algorithm of Sect. 1.1 seen in the dual is a cutting-plane procedure to maximize θ: at each iteration, we maximize θ^k instead. Then we can write

$$\theta(u) \leqslant \theta^k(u) \leqslant \theta^k(u^k) \quad \text{for all } \ u\in\mathbb{R}^m_+. \tag{1.16}$$

To check whether the iterate u^k thus obtained does maximize θ, we compute the true value $\theta(u^k)$: we call the oracle (1.4) or (1.10) to obtain $x(u^k)$. Said otherwise, we check whether u^k is feasible for the unrestricted dual master (1.12), calling on (1.4) or (1.10) to identify the most violated inequality. The same two cases (i), (ii) of Sect. 1.1 can arise, namely:

(i) Either $x^{k+1} := x(u^k)$ defines a violated inequality:

$$\theta^k(u^k) = s^k > u^k b + (c-u^kA)x^{k+1} = \theta(u^k);$$

then, the new cut associated with x^{k+1} is appended to the bundle and the process is repeated.

(ii) Or all constraints in I are satisfied, i.e., $\theta(u^k)=\theta^k(u^k)$; then (1.16) shows that u^k maximizes θ, i.e. (1.12) is solved.

Remark 1.3 From the oracle (1.4), the number

$$s^k - u^k b - (c-u^kA)x^{k+1} = \theta^k(u^k) - \theta(u^k)$$

introduced in (i) is the largest violation of the constraints in (1.12). It is also the (opposite of the) smallest reduced cost of Remark 1.2. Besides, the primal-dual problems (1.5), (1.6), (1.8), (1.14), (1.15) have all the same optimal value:

$s^k = \theta^k(u^k) = c\hat{x}$, where we have used (1.7). Thus, this number is also the duality gap $c\hat{x} - \theta(u^k)$.

Just as in Remark 1.2, the algorithm can be stopped when the above number is "sufficiently small", i.e. when $c\hat{x} - \theta(u^k) \leqslant \varepsilon$. This means that $\hat{x}$ and u^k are ε-optimal, i.e. $c\hat{x}$ and $\theta(u^k)$ are within ε of the common optimal value in the primal-dual pairs of problems (1.2) or (1.3), (1.11) or (1.12). The tolerance ε is measured in the same units as the objective function of (1.2) and can be conveniently controlled.

However, because the sequence $\theta(u^k)$ is not monotone, a more efficient stopping test will use the best dual iterate

$$\hat{u} := \operatorname{argmax}\{\theta(u^i) : i = 1, \ldots, k\} \tag{1.17}$$

(assumed unique just for simplicity). In fact, if

$$c\hat{x} - \theta(\hat{u}) \leqslant \varepsilon, \tag{1.18}$$

then $c\hat{x}$ and $\theta(\hat{u})$ are ε-optimal. The best dual iterate $\hat{u}$ will play an important role in the sequel. □

Primal-dual relations coming from standard LP theory can be derived in our convex-analysis language. The set of optimal solutions in (1.13) (for given u), namely

$$\hat{X}^k(u) := \left\{x \in \operatorname{conv}(X^k) : L(x,u) = \theta^k(u)\right\} \tag{1.19}$$

is important for this. If u solves (1.14), $\hat{X}^k(u)$ gathers all columns associated to active constraints in (1.8) or (1.15), or equivalently with zero reduced cost in (1.6).

Theorem 1.4

(i) *An optimal solution u^k of the restricted master* (1.14) *is characterized by the existence of some $\hat{x} \in \hat{X}^k(u^k)$ satisfying complementarity slackness:*

$$A\hat{x} - b \geqslant 0, \quad u^k \geqslant 0, \quad u^k(A\hat{x} - b) = 0$$

(*abbreviated as* $0 \leqslant A\hat{x} - b \perp u^k \geqslant 0$).

(ii) *The $\hat{x}$'s described in* (i) *coincide with the optimal solutions of* (1.5).

(iii) *With $\hat{u}$ defined in* (1.17)*, these $\hat{x}$'s are optimal in* (1.2) *if $c\hat{x} = \theta(\hat{u})$.*

Proof Elementary convex analysis says that θ^k is concave and its subdifferential (the set of $g \in \mathbb{R}^m$ such that $\theta^k(v) \leqslant \theta^k(u) + (v-u)g$ for all $v \in \mathbb{R}^m$) is

$$\partial\theta^k(u) = \cup\left\{g = b - Ax : x \in \hat{X}^k(u)\right\};$$

see for example [23, Sect. D4.3-4] or [22, Sect. VI.4.3-4]. Then u^k maximizes θ^k if and only if some subgradient lies in the normal cone to $\mathbb{R}^m_+$ at u^k; this is (i).

For (ii), invoke for example [51, Theorem 28.1] or [22, Theorem VII.4.4.3]: the pairs $(\hat{x}, u^k)$ solving the pair of dual problems (1.5), (1.14) are the saddle-points of the Lagrangian (1.9), i.e., those feasible pairs satisfying

$$\forall (x,u) \in \text{conv}\,(X^k) \times \mathbb{R}^m_+,\ L(x,u^k) \geqslant L(\hat{x},u^k) \geqslant L(\hat{x},u).$$

It is not difficult to check that this repeats (i).

As for (iii), set $\varepsilon = 0$ in (1.18). □

Needless to say, an $\hat{x}$ described in (i) can be expressed via (1.7)—with $\hat{\lambda}$ optimal in (1.6)—in terms of a number of columns satisfying $cx^i - u^k Ax^i - s = 0$, i.e. $L(x^i, u^k) = \theta^k(u^k)$; see (1.19)!

Let us conclude this introduction to standard column generation:

- It consists in merely restricting X to X^k in (1.2), leaving everything else unchanged. In the dual space, θ is correspondingly "impoverished" to θ^k. In nonlinear optimization, this is known as the cutting-plane method of Kelley [24] or Cheney-Goldstein [7].
- It needs a number of columns to start.
- It can be desperately slow: an example constructed by A.S. Nemirovskii ([46, Sect. 4.3.6], reproduced in [22, Sect. XV.1.1]) shows that as many as $(1/\varepsilon)^{m/2}$ calls to the oracle may be necessary to reach ε-optimality.
- On the other hand, it directly provides a feasible primal point $\hat{x}$ which satisfies complementarity slackness with u^k and is a distinguished candidate to solving (1.2).
- As a result, it can be safely stopped as soon as the window $c\hat{x} - \theta(u^k)$ or $c\hat{x} - \theta(\hat{u})$ is small.

To finish this section, note that the method of subgradient [57,12,49] is also a valid candidate to maximize θ, as well as the ellipsoid method [54,45], which reaches accuracy ε after $(\log 1/\varepsilon)^m$ calls. Both methods behave poorly in practice, though. Incidentally, we mention a little-known fact: the subgradient method does provide a substitute for $\hat{x}$, as well as a stopping criterion resembling (1.18); see [1,34,55]. Also related is the volume algorithm of [3], considered in Sect. 3 below.

1.3 Stabilized column generation

Seen in the dual space, where the problem is to maximize θ, the rationale for (1.8) is the hope that the restriction to a subset of k columns gives a good approximation (1.13) of the true dual function (1.10): $\theta^k \simeq \theta$, so that maximizing θ^k should do good in terms of maximizing θ. Figure 2 reproduces the right part of Fig. 1, in which the horizontal line is the level $\theta(\hat{u})$ of the best answer of the oracle—see (1.17). Because $\theta^k \geqslant \theta$, the optimal $(u^*, \theta(u^*))$ clearly lies somewhere in the "safeguard polyhedron" P of Fig. 2. Standard column generation chooses u^k as the highest point in P.

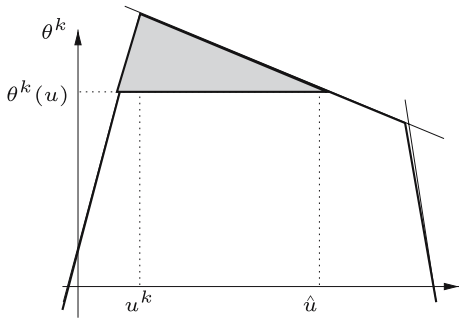

Fig. 2 The safeguard polyhedron P for $k = 3$

Nemirovskii's example warns us that choosing this highest point may be far too optimistic and we may well have $\theta(u^k) \ll \theta^k(u^k)$. It is advisable to adopt more conservative strategies, known as *stabilized column generation*, in which u^k is replaced by some less high but more central point in P. Referring for example to [6] for a wide overview, we restrict here our attention to the following stabilization scheme. Instead of θ^k, some other function is maximized, the dual restricted master (1.14) being replaced by

$$\max_{u \in \mathbb{R}^m_+} \tilde{\theta}^k(u) \quad \text{where } \tilde{\theta}^k(u) := \theta^k(u) - S(u - \hat{u}). \tag{1.20}$$

Here the stability center $\hat{u}$ is defined in (1.17); the stabilizing function S should have 0 as a minimum point, in order to pull u^k toward $\hat{u}$. Up to some minor variations, these methods work as follows.

Algorithm 1.5 (Schematic stabilized column generation)

Step 0 Choose an initial stability center $\hat{u}$. Select an initial set of k columns.
Step 1 Compute an optimal solution u^k of the *stabilized dual restricted master* problem (1.20).
Step 2 Call the oracle (1.4) at u^k to obtain the new column x^{k+1} and the dual value $\theta(u^k) = L(x^{k+1}, u^k)$. Perform the stopping test.
Step 3 If $\theta(u^k) > \theta(\hat{u})$ set $\hat{u} = u^k$.
Step 4 Increase k by 1 and go to Step 1. □

Step 3 is motivated by (1.17). The initial $\hat{u}$ may be set to any heuristic estimate of a dual optimal solution to (1.3); as a default, $\hat{u}$ may be initialized to 0. Most such algorithms may start at $k = 0$ with no initial column, u being initialized to $\hat{u}$ in Step 1. Standard column generation uses $S \equiv 0$ and does need a nonempty set of initial columns.

Beware that, in the stabilized version, u^k no longer maximizes θ^k, so Theorem 1.4 does not apply. The role of that theorem was to allow the construction of the primal candidate $\hat{x}$ from the optimality conditions in the dual restricted master (1.8), or as a solution of (1.6), viewed as the dual of (1.8). Stabilized duals of the form (1.20) will provide an alternative candidate—denoted by $\tilde{x}$ hereafter. To obtain this candidate by solving an optimization problem,

we need to define a dual of (1.20) resembling (1.5). This dualization is indeed possible via conjugate calculus (see [22, Chap. XI] or [23, Chap. E]). Specifically, introduce the *conjugate function*

$$S^*(g) := \max_{v\in\mathbb{R}^m} [vg - S(v)]; \tag{1.21}$$

g shall be interpreted as a slack for constraints $Ax \geqslant b$, while v stands for the deviation between u and the stability center $\hat{u}$. As will be seen in Theorem 1.7 below, a convenient dual to (1.20), called the *Fenchel dual*, is then

$$\min cx + \hat{u}g + S^*(g), \qquad Ax \geqslant b - g, \quad (x,g) \in \operatorname{conv}(X^k) \times \mathbb{R}^m. \tag{1.22}$$

This is admittedly a rather abstract problem, it will be made more precise when S is specified.

Remark 1.6 A way to obtain (1.22) is to formulate (1.20) as

$$\max \left[\theta^k(u) - S(v)\right], \quad v = u - \hat{u}, \ \ (u,v) \in \mathbb{R}^m_+ \times \mathbb{R}^m$$

and to apply Lagrangian relaxation: associating a multiplier $g \in \mathbb{R}^m$ with the constraint $v = u - \hat{u}$, the Lagrangian $\theta^k(u) - S(v) + (v - u + \hat{u})g$ produces the dual

$$\min_{g\in\mathbb{R}^m} \left[\max_{u\geqslant 0} \left(\theta^k(u) - ug\right) + S^*(g) + \hat{u}g\right].$$

Tedious calculations show that this coincides with (1.22).

Also, we mention two situations where (1.22) simplifies:

– An equality-constrained master problem (1.5) produces $Ax = b - g$ in (1.22), which becomes

$$\min cx + \hat{u}(b - Ax) + S^*(b - Ax), \quad x \in \operatorname{conv}(X^k).$$

The connection with (1.5) is more transparent; in particular, the above minimand discloses the "augmented Lagrangian" $L(x,\hat{u}) + S^*(b - Ax)$, with an augmenting function S^*. Remembering that S is minimal at 0, it can be shown that S^* is also minimal at 0, and therefore pulls $b - Ax$ toward 0.

– Standard column generation uses $S \equiv 0$, whose conjugate is clearly

$$S^*(g) = \begin{cases} 0 & \text{if } g = 0\,, \\ +\infty & \text{otherwise} \end{cases} =: \mathrm{i}_{\{0\}}(g),$$

the so-called *indicator function* of $\{0\}$: the term $S^*(g)$ forces $g = 0$ in (1.22), which becomes exactly (1.5). □

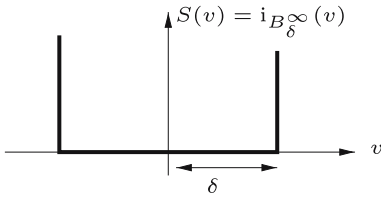

Fig. 3 The indicator function of B_δ^∞

We can now state an adapted version of Theorem 1.4:

Theorem 1.7 *Assume that S is a convex function over $\mathbb{R}^m$. Consider an optimal solution u^k of (1.20) such that $S(v) < +\infty$ for all v close to $u^k - \hat{u}$. Then:*

(i) *There exist $\tilde{x} \in \hat{X}^k(u^k)$ of (1.19) and a subgradient $\tilde{g} \in \partial S(u^k - \hat{u})$ such that $0 \leqslant (A\tilde{x} - b + \tilde{g}) \perp u^k \geqslant 0$.*
(ii) *The $(\tilde{x}, \tilde{g})$'s described in* (i) *coincide with the optimal solutions of (1.22).*
(ii′) *If $\tilde{g} = 0$ in* (i)*, then the corresponding $\tilde{x}$ is optimal in (1.5).*
(iii) *If, in addition, $c\tilde{x} = \theta(\hat{u})$, then $\tilde{x}$ is optimal in (1.2).*

Proof The local finiteness of S is just a technical "qualification" assumption, to guarantee in (1.20) that $\partial\tilde{\theta}(u^k) = \partial\theta^k(u^k) - \partial S(u^k - \hat{u})$ (see [22, Sect. XI.3.5]; other assumptions are possible, for example that $\tilde{\theta}$ is polyhedral). Then (i) goes as in Theorem 1.4(i); (ii) is the so-called Fenchel duality theorem and the rest is clear. □

Thus, $\hat{x}$ of Sect. 1.1 is replaced by $\tilde{x}$, which can be obtained by proving optimality of u^k in (1.20), or by directly solving its associated "bidual" (1.22). As before, $\tilde{x}$ can be expressed via (1.7) in terms of a number of columns satisfying $L(x^i, u^k) = \theta^k(u^k)$. What is missing is *feasibility*—and a fortiori complementarity: $A\tilde{x} \not\geqslant b$, unless $\tilde{g} \leqslant 0$. As a result, two things must be checked before a stabilized algorithm can be stopped: $c\tilde{x} - \theta(\hat{u})$ must be small as before, but the positive part of $\tilde{g}$ must also be small (so that $\tilde{x}$ is approximately feasible). See [14] for a convergence study of general stabilized methods.

Several choices have been proposed for S, which we briefly review now.

1.3.1 Boxstep

A first stabilizing device forces u in an ℓ_∞-box around $\hat{u}$. The stabilizing problem is

$$\max\{\theta^k(u) : u \geqslant 0,\ \|u - \hat{u}\|_\infty \leqslant \delta\}, \quad \text{i.e. } \max_{u \in \mathbb{R}^m_+} \theta^k(u) - \mathrm{i}_{B_\delta^\infty}(u - \hat{u}); \tag{1.23}$$

S is here the indicator function of the ℓ_∞-ball B_δ^∞ of radius δ and centered at the origin, see Fig. 3.

This is the boxstep method of [42]. Its original aim was not much to reduce the number of calls to the oracle, but rather to ease the resolution of (1.6),

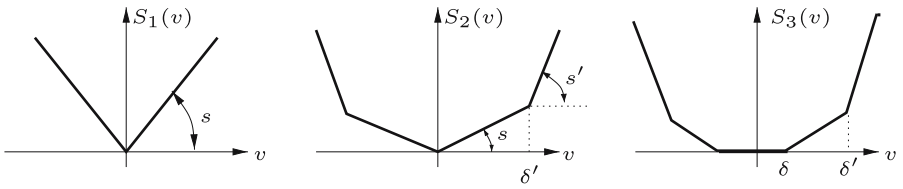

Fig. 4 Stabilization by ℓ_1 penalty

replacing it by

$$\max \theta(u), \qquad \|u - \hat{u}\|_\infty \leqslant \delta, \quad u \in \mathbb{R}^m_+.$$

The stability center $\hat{u}$ was chosen a priori; it was not moved as in (1.17) but only after the above problem was fully solved (unless $\|u^k - \hat{u}\|_\infty < \delta$, indicating overall optimality and termination of the column generation process).

The conjugate (1.21) is easy to compute when S is an indicator function: this gives $\mathrm{i}^*_{B^\infty_\delta}(g) = \delta\|g\|_1$. According to Remark 1.6 and Theorem 1.7, Boxstep therefore produces an $\tilde{x}$ solving

$$\min cx + \hat{u}g + \delta\|g\|_1, \qquad Ax \geqslant b - g, \quad (x,g) \in \mathrm{conv}\,(X^k) \times \mathbb{R}^m. \tag{1.24}$$

Both problems (1.23) and (1.24) can be formulated as linear programs; this will be done in Sect. 1.3.3 below.

1.3.2 Polyhedral penalties

A series of stabilizing terms S have been subsequently proposed, in which the stabilized dual restricted master (1.20) is a more and more sophisticated LP. Some of the corresponding stabilizing terms are depicted in Fig. 4.

In [25], S is V-shaped (left part of Fig. 4): (1.20) has the form

$$\max_{u\in\mathbb{R}^m_+} \theta^k(u) - s\sum_{j=1}^m |u_j - \hat{u}_j|, \quad \text{i.e.} \max_{u\in\mathbb{R}^m_+} \theta^k(u) - s\|u - \hat{u}\|_1, \tag{1.25}$$

with possibly $2m$ penalty coefficients s_j^+ and s_j^- instead of the single s. Interestingly enough, this stabilization is the dual counterpart of Boxstep: in fact, either LP calculations or standard convex analysis show that $(s\|\cdot\|_1)^* = \mathrm{i}_{B^\infty_s}$ and the Fenchel bidual (1.22) here takes the form

$$\min cx + \hat{u}g, \qquad Ax \geqslant b - g, \quad \|g\|_\infty \leqslant s, \ \ (x,g) \in \mathrm{conv}\,(X^k) \times \mathbb{R}^m, \tag{1.26}$$

which can again be formulated as a linear program.

The penalty used in [44] has a pair of breakpoints (central part of Fig. 4), again with possibly adaptable slopes s, s' and breakpoints δ'. The stabilizer of

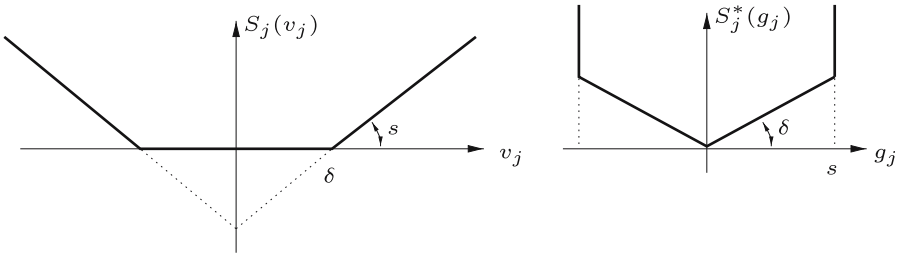

Fig. 5 An illustrating stabilization (and its conjugate)

the right part of Fig. 4 is proposed in [5], where the management of $\hat{u}$ is inspired by boxstep, rather than (1.17).

A common feature of these stabilizations is the necessity to manage carefully their parameters s, δ, etc. For example:

- In the boxstep stabilization of Fig. 3, a large δ stabilizes nothing, while a small δ slows down the iterates unduly.
- The form S_1 of Fig. 4 requires $s \to 0$, to force a small $\tilde{g}$ in Theorem 1.7—see (1.26).

1.3.3 Polyhedral stabilization in the primal

The main merit of the stabilizations reviewed so far is to allow LP formulations, both in the primal and dual spaces—as was alluded to for (1.24) and (1.26). Let us make explicit such corresponding LP programs in the case $S(v) = s\sum_{j=1}^{m} \max\{0, |v_j| - \delta\}$: the left part of Fig. 5 depicts the one-dimensional component j of S, which can be represented by

$$S_j(v_j) = s \min\{\rho_j : \rho_j \geqslant 0,\ \rho_j \geqslant v_j - \delta,\ \rho_j \geqslant -v_j - \delta\}. \tag{1.27}$$

The LP formulation of (1.20) is here

$$\begin{cases} \max ub - r - s\sum_{j=1}^{m} \rho_j, & \\ uAx^i - r \leqslant cx^i & i = 1,\dots,k, \ \text{multipliers } \lambda_i \\ u_j - \rho_j \leqslant \hat{u}_j + \delta & j = 1,\dots,m, \ \text{multipliers } g_j^+ \\ -u_j - \rho_j \leqslant -\hat{u}_j + \delta & j = 1,\dots,m, \ \text{multipliers } g_j^- \\ u, \rho \geqslant 0, & \end{cases}$$

whose LP dual is (A_j being the j^{th} row of A)

$$\begin{cases} \min \sum_{i=1}^{k} (cx^i)\lambda_i + \sum_{j=1}^{m} (\delta + \hat{u}_j) g_j^+ + \sum_{j=1}^{m} (\delta - \hat{u}_j) g_j^-, & \\ \sum_{i=1}^{k} \lambda_i = 1, & \\ \sum_{i=1}^{k} (A_j x^i)\lambda_i + g_j^+ - g_j^- \geqslant b_j & j = 1,\dots,m, \\ g_j^+ + g_j^- \leqslant s & j = 1,\dots,m, \\ \lambda, g^+, g^- \geqslant 0. & \end{cases} \tag{1.28}$$

This stabilization amounts to relaxing the constraints $Ax \geqslant b$ by introducing slack variables g_j^+ and g_j^-, whose sum is bounded by s, and with marginal costs $\delta + \hat{u}_j$ and $\delta - \hat{u}_j$ respectively.

LP formulations are possible because the penalty function S and its conjugate S^* can be written via linear relations. An illustrative exercise is then to derive the "Fenchel bidual" (1.22) on the above example, the fundamental operation being to compute the conjugate S^*. This can be done by rather elegant conjugacy calculus, or by tedious calculations starting from (1.27). In either case, our Fenchel bidual (1.22) boils down to

$$\min cx + \hat{u}g + \delta\|g\|_1, \qquad Ax \geqslant b - g, \quad \|g\|_\infty \leqslant s, \quad x \in \operatorname{conv}(X^k).$$

Setting $s = +\infty$ [resp. $\delta = 0$] reproduces (1.24) [resp. (1.26)]. Then, additional calculations result in (1.28) (a key observation is that the constraints $g_j^+ + g_j^- \leqslant s$ can be replaced by $g_j^+ \leqslant s, g_j^- \leqslant s$ since both g_j^+ and g_j^- cannot be simultaneously positive).

1.3.4 Euclidean penalty: bundle methods

Contemplating the sequence of stabilizers of Fig. 4 suggests that they tend to mimic a quadratic function—which takes δ and s "infinitely small", and "infinitely many" breakpoints. In fact, stabilization is a rather fundamental paradigm in nonlinear programming (not limited to Kelley's method) and there are good reasons to use quadratic stabilizers; see some explanations in [22, Sect. II.2.2(c)].

Bundle methods, aimed at maximizing a concave function such as the present θ, go back to [35]. In their latest form, initiated in [36,27] and fully described in [22, Chap. XV], they are indeed a stabilized Kelley's method, with a Euclidean norm for S in (1.20): u^k solves

$$\max_{u \in \mathbb{R}^m_+} \theta^k(u) - \frac{1}{2t}\|u - \hat{u}\|^2 \tag{1.29}$$

(at this point, we mention a technical detail motivated by non-polyhedral θ's: $\hat{u}$ is not exactly the best point of (1.17): actually, Step 3 of Alg. 1.5 moves $\hat{u}$ to the new iterate u^k only if this results in a "definite" increase for θ).

Clearly, $S^*(g) = \frac{t}{2}\|g\|^2$, so the Fenchel bidual (1.22) writes

$$\min cx + \hat{u}g + \frac{t}{2}\|g\|^2, \qquad Ax \geqslant b - g, \quad (x, g) \in \operatorname{conv}(X^k) \times \mathbb{R}^m. \tag{1.30}$$

Solving this problem with respect to g for fixed x gives the explicit value $g = \max\{b - Ax, -\hat{u}/t\}$; plugging it back into the objective function discloses the augmented Lagrangian of [52]. A rather compact writing of the resulting

problem is

$$\min_{x\in\mathrm{conv}\,(X^k)} cx + \frac{1}{2t}\left\|(t(b-Ax)+\hat{u})_+\right\|^2 - \frac{1}{2t}\|\hat{u}\|^2.$$

Remark 1.8 Assume an equality-constrained master problem (1.5), in which case u is unconstrained in (1.29). As mentioned in Remark 1.6, the Fenchel bidual simplifies to

$$\min cx + \hat{u}(b-Ax) + \frac{t}{2}\|b-Ax\|^2, \quad x \in \mathrm{conv}\,(X^k). \tag{1.31}$$

The augmented Lagrangian is now more transparent: instead of imposing the constraint $Ax = b$ as in (1.5), we

- dualize it via $\hat{u}(b-Ax)$ (observe the careful choice of the dual variable, which is set to the stability center $\hat{u}$),
- and penalize it via $\frac{t}{2}\|b-Ax\|^2$, which limits constraint violations by the candidate $\tilde{x}$. □

The relevant primal-dual relations now rely on a crucial output of the quadratic restricted master (1.29):

$$\tilde{g} := \frac{u^k - \hat{u}}{t}. \tag{1.32}$$

Theorem 1.9 *Use the notation* (1.32).

(i) *The unique optimal solution u^k of* (1.29) *is characterized by the existence of some $\tilde{x} \in \hat{X}^k(u^k)$ such that $0 \leqslant (A\tilde{x} - b + \tilde{g}) \perp u^k \geqslant 0$.*

(ii) *Together with $\tilde{g}$, the $\tilde{x}$'s described in* (i) *coincide with the optimal solutions of* (1.30).

(ii′) *If $u^k = \hat{u}$ then these $\tilde{x}$'s coincide with the optimal solutions of* (1.5).

(iii) *In case* (ii′), *they are also optimal in* (1.2).

Proof That (1.29) has a unique optimal solution is clear. Then apply Theorem 1.7. Here (1.22) is (1.30) and $\partial S(u-\hat{u})$ is clearly the singleton $(u-\hat{u})/t$, providing the explicit value for $\tilde{g} = (u^k - \hat{u})/t$.

In case (ii′), we have $0 \in \partial\theta^k(u^k) - 0$, hence u^k maximizes θ^k. Besides, $\tilde{g} = 0$ in (i) shows that $\tilde{x}$ is feasible in (1.5) and (1.2). Finally, observe in Algorithm 1.5 that $\theta^k(\hat{u}) = \theta(\hat{u})$. Thus, $\hat{u}$ maximizes θ and (iii) is proved. □

The proof of (iii) deserves a comment. First observe that a key is the existence of the gradient $\tilde{g} = \nabla S(u^k - \hat{u})$. Indeed (iii) holds in Theorem 1.7 as well, whenever S in (1.20) is a differentiable function. Also, (iii) seems paradoxical: it implies that checking $c\tilde{x} = \theta(\hat{u})$ is unnecessary to stop the process. The trickery here is that the oracle has actually been already called at $\hat{u}$, during some previous iteration. This guarantees the last equality in the chain

$$\forall u \in \mathbb{R}^m, \quad \theta(u) \leqslant \theta^k(u) \leqslant \theta^k(u^k) = \theta^k(\hat{u}) = \theta(\hat{u}).$$

The crucial $\tilde{g}$ of (1.32) measures the "lack of complementarity" between $\tilde{x}$ and $\hat{u}$ (or the "lack of optimality" of the latter); and its convergence to 0 conditions the convergence of a bundle method. This latter property does not depend much on t: it holds for example with $t \equiv 1$. However practical efficiency does require a $t = t^k$ adapted at each iteration.

1.3.5 ACCPM

The analytic-center cutting-plane method of [17] replaces θ^k by a barrier function as follows. With respect to the variable $(u,s) \in \mathbb{R}^m_+ \times \mathbb{R}$, the safeguard polyhedron P of Fig. 2 is defined by the k constraints of the dual restricted master (1.15), and a level constraint, namely

$$\ell_i(u,s) := L(x^i,u) - s \geqslant 0,\ i = 1,\ldots,k, \quad \text{and} \quad \ell_0(u,s) := s - \theta(\hat{u}) \geqslant 0.$$

The so-called analytic center of P is then the unique point maximizing the barrier function $B(u,s) := \sum_{i=0}^{k} \log \ell_i(u,s)$. Note that replacing θ^k by B is a stabilization in itself: the analytic center certainly satisfies the constraints strictly (see Fig. 2 again). The constraints ℓ_i are actually suitably scaled, and a complexity analysis [18] shows that a few Newton iterations maximizing B suffice to guarantee good overall convergence. In a variant of the method [47,48], an extra quadratic stabilizer $\frac{1}{2t}\|u - \hat{u}\|^2$ is subtracted from B and convergence is improved; see also [19].

2 Numerical comparisons

We have compared numerically the bundle stabilization of Sect. 1.3.4 with the standard Kelley algorithm of Sect. 1.1. Ideally, a comparison with the most advanced LP stabilizations would be interesting. However, the necessity of application-specific tuning of various parameters makes it impractical. The applications used for these comparative tests are the cutting-stock, vextex-coloring, capacitated vehicle-routing, multi-item lot-sizing, and traveling-salesman problems. They illustrate the diversity of models that can arise when X has a more complex structure, as alluded in Remark 1.1. Each application is introduced by presenting the specific form taken by the original integer program, the associated Lagrangian dual and its Dantzig-Wolfe formulation. First we give some implementation details.

2.1 Implementations

The algorithms described in Sects. 1.1 and 1.3.4 are implemented using the environment of *BaPCod*, a generic Branch-And-Price Code [59]. Xpress^{MP}[9] is used for solving master linear programs while customized solvers are used for the oracles. We emphasize that, in all of our tests below, the oracle minimizes

the Lagrangian exactly (returns the most negative reduced cost), even though this may not be the usual practice in column generation.

Kelley Two versions are considered: Kbasic and Krich, which differ by their initial columns. These "columns", only characterized by their cost and constraint value, are purely artificial: they do not correspond to any element of X, or even of conv(X). As a result, they must be eliminated from the solution $\hat{x}$ of the master before a reliable feasible cost $c\hat{x}$ can be used for the stopping test.

- In the basic version Kbasic, the algorithm of Sect. 1.1 is initialized with a single artificial column – denote it by x^1 —with constraint value $Ax^1 \geqslant b$, and whose cost cx^1 is set to an estimate of the optimal value of (1.1).
- Krich is initialized with m artificial columns. Their constraint values form the canonical basis of $\mathbb{R}^m$: A_j being the jth row of A, $A_j x^i = 1$ if $i = j$, 0 otherwise; and $cx^i = \alpha \hat{u}_i$ where $\hat{u}$ is an application-specific heuristic estimate of the optimal dual solution of (1.2) and α is a factor, set to 1.2 unless otherwise specified.[1]

For both versions, the algorithm is stopped when (1.18) holds with ε set at 10^{-6}. However, if artificial columns are in the primal solution $\hat{\lambda}$ at that stage, their cost is increased by the factor α and the column generation procedure is continued.

Remark 2.1 Let $j = 1, \ldots, m$ index the artificial columns in Krich and call g_j their multipliers: the kth primal master is

$$\begin{cases} \min \sum_{i=1}^{k} (cx^i)\lambda_i + \alpha \sum_{j=1}^{m} \hat{u}_j g_j, \\ \sum_{i=1}^{k} \lambda_i = 1, \\ \sum_{i=1}^{k} A_j x^i \lambda^i + g_j \geqslant b_j \qquad\qquad j = 1, \ldots, m, \\ \lambda, g \geqslant 0. \end{cases}$$

This resembles (1.28), with g^- fixed to 0, $\delta_j = (\alpha - 1)\hat{u}_j$; also s is set to $+\infty$. The dual of the above program is

$$\begin{cases} \max ub - r, \\ uAx^i - r \leqslant cx^i \quad i = 1, \ldots, k, \\ u_j \leqslant \alpha \hat{u}_j \qquad j = 1, \ldots, m, \\ u \geqslant 0. \end{cases}$$

Thus, Krich resembles a sort of "static half-boxstep", in which $\hat{u}$ remains fixed and u is bounded from above only (0 being a natural lower bound). The size of the box is increased when the artificial cost is increased (to force the artificial variable out of the optimal LP solution). □

Bundle Our implementation follows rather accurately [41], in particular for the management of the stabilizing parameter $t = t^k$. The restricted master

[1] The artificial x^i's are actually *rays*: it is only large enough multiples of them that contribute feasibility.

problem (1.29) is solved by the QP solver of Kiwiel [26,29]. Unless otherwise specified, we start with an empty initial set of columns. Most of the time, two versions are tested: BUNDLE, where the initial iterate $\hat{u}$ is problem-dependent, and BUNDLE0 initialized at $\hat{u} = 0$. The algorithm needs an initial value for t, and this is obtained from an estimate of the optimal value of (1.2). The algorithm stops when (1.18) holds, together with $\|\tilde{g}\| \leqslant \varepsilon\sqrt{m}$ (meaning that $\tilde{x}$ satisfies the dualized constraints within ε on the average). The same value $\varepsilon = 10^{-6}$ is used throughout, for bundle and for Kelley.

Feasible primal points Once the artificial columns are properly eliminated, Kelley's method provides a feasible candidate $\hat{x}$ at each iteration; the bulk of the work is then to improve this feasible solution. By contrast, primal feasibility is only asymptotic for a bundle method, which stops as soon as a feasible enough primal point is obtained—in a way, this is predicted by Theorem 1.9(iii). Reaching primal feasibility is thus the whole business of a bundle method. We have therefore considered BUNDLE+K, a variant aimed at combining the two approaches: bundle generates the sequence of dual iterates u^k, while Kelley is used to obtain a primal solution $\hat{x}'$ based on which we perform an additional test at each iteration, in the hope to stop the algorithm earlier. Thus, BUNDLE+K is as follows (compare with Algorithm 1.5).

Algorithm 2.2 (Combination of bundle and Kelley)

Step 0	Choose an initial stability center $\hat{u}$. Select an initial set of k columns.
Step 1	Compute the optimal solution u^k of the quadratic restricted master problem (1.29).
Step 1′	Solve the linear restricted master problem (1.6) to obtain $\hat{x}'$. Stop if $c\hat{x}' - \theta(\hat{u}) \leqslant \varepsilon$.
Step 2	Call the oracle (1.4) at u^k to obtain the new column x^{k+1} and the dual value $\theta(u^k) = L(x^{k+1}, u^k)$. Perform the stopping test.
Step 3	If $\theta(u^k) > \theta(\hat{u})$ set $\hat{u} = u^k$.
Step 4	Increase k by 1 and go to Step 1. □

Our numerical experiments have been performed on a PC pentium 3, 1 Ghz. Most of the tables below record the number of iterations and CPU times.

- One iteration means one execution of the oracle and of the master. Besides the number of iterations, the tables also record the total number of columns generated (including the artificial one for KBASIC, but not for KRICH and including the last subproblem solution with zero reduced cost when stopping does not arise before that). Note that, for a stabilized algorithm such as bundle, the oracle may answer the same column to several calls, even though these calls are made with different u's.
- The total CPU time is spent respectively in the oracle and in the master, plus an overhead which may not be negligible. Note that when the oracle is exponential (as is the case for all applications but the multi-item lotsizing and the TSP), its computing time may vary in fairly large proportions.

2.2 The cutting stock problem (CSP)

The problem is to minimize the number of stock pieces of width W, used to meet demands $d_1, \ldots, d_m$, for items $j = 1, \ldots, m$, to be cut at their width $w_1, \ldots, w_m$. We assume that every w_j is smaller than W and that there are enough stock pieces, say N, available for a feasible cutting: for example $N = \sum_j d_j$ is certainly enough.

Let us put the resulting problem in the framework of the present paper. Call $y^\ell \in \{0,1\}$ an indicator of whether stock piece ℓ is used, and $z_j^\ell \in \mathbb{N}$ the quantity of item j cut in stock piece ℓ. Then, a possible formulation for (1.1) is

$$\begin{cases} \min \sum_{\ell=1}^N y^\ell, & \\ \sum_{\ell=1}^N z_j^\ell \geqslant d_j & \text{for } j = 1, \ldots, m, \\ \sum_{j=1}^m z_j^\ell w_j \leqslant W y^\ell & \text{for } \ell = 1, \ldots, N, \\ y^\ell \in \{0,1\},\ z_j^\ell \in \mathbb{N} & \text{for } j = 1, \ldots, m,\ \ell = 1, \ldots, N. \end{cases} \tag{2.1}$$

Introducing the variable vector $x = (y, z)$, call $Ax \geqslant b$ the demand-covering constraints $\sum_\ell z^\ell \geqslant d$ and put the other constraints in X; the Lagrangian (1.9) is then

$$L(y, z, u) = \sum_{\ell=1}^N \big(y^\ell - u z^\ell\big) + ud.$$

It is decomposable with respect to ℓ and its minimization produces the dual function (1.10) which takes the form

$$\theta(u) = \mathrm{d}u + \sum_{\ell=1}^N \min\big\{y^\ell - u z^\ell :\ w z^\ell \leqslant W y^\ell, \quad y^\ell \in \{0,1\},\ \ z^\ell \in \mathbb{N}^m\big\}.$$

This is the juxtaposition of N identical optimization problems, which can be solved by inspection on y^ℓ: one solves the knapsack problem[2]

$$\max uz, \qquad wz \leqslant W, \quad z \in \mathbb{N}^m \tag{2.2}$$

to obtain an optimal solution $z(u)$—we drop the useless index ℓ. The minimum value in the above curly bracket is then $\min\{0, 1 - uz(u)\}$, so that the dual function has the value

$$\theta(u) = du + \begin{cases} 0 & \text{if } uz(u) \leqslant 1, \\ N(1 - uz(u)) & \text{otherwise.} \end{cases} \tag{2.3}$$

[2] The knapsack solver proceeds by transforming the problem into a multiple-class binary knapsack problem and solves it using an extension of the standard branch-and-bound algorithm of Horowith and Sahni, as presented in [58].

An alternative presentation follows the Dantzig–Wolfe model (1.3). Let I enumerate solutions $x = (y, z)$, where z is feasible for (2.2) and y is the associated indicator variable. The problem can be formulated as

$$\min \sum_{i\in I} y^i\lambda_i, \qquad \sum_{i\in I} z^i\lambda_i \geqslant d \in \mathbb{R}^m, \quad \sum_{i\in I} \lambda_i \leqslant N, \quad \lambda_i \geqslant 0, \ \ i \in I.$$

Here $\sum_i \lambda_i \leqslant N$ is inherited from the original convexity constraints $\sum_i \lambda_i^\ell = 1$ associated with each stock piece: as stock pieces are identical, convexity constraints can be aggregated; the resulting constraint can be relaxed to a $\leqslant$-constraint since $(y, z) = 0$ is a solution of I. They express that the total number of available stock pieces is limited. Our N being an overestimate of the actual number of stock pieces used, this constraint is not binding and can be dropped: the formulation becomes

$$\min \sum_{i\in I} \lambda_i, \qquad \sum_{i\in I} z^i\lambda_i \geqslant d \in \mathbb{R}^m, \quad \lambda_i \geqslant 0, \ \ i \in I, \tag{2.4}$$

whose dual is

$$\max \mathrm{d}u, \qquad uz^i \leqslant 1, \quad i \in I, \ \ u \in \mathbb{R}^m_+. \tag{2.5}$$

Remark 2.3 It is interesting to observe that the formulation (2.5) of the dual is actually a *nonlinearly constrained* problem, namely

$$\max \mathrm{d}u, \qquad h(u) \leqslant 1, \quad u \geqslant 0,$$

where

$$h(u) := \max_{wz\leqslant W} uz = uz(u).$$

In other words, the oracle delivers the value of a dual *constraint* function h rather than a dual *objective* function θ. From this point of view, maximizing (2.3) appears as a so-called *exact penalty* approach to solve (2.5): the term $N(1 - uz(u))$ imposes a price to pay for infeasible u's. □

For this application, we compare in Tables 1 and 2 the two versions of Kelley, the standard bundle method, and the version BUNDLE+K of Algorithm 2.2. For KBASIC, the cost of the unique artificial column is set to $\lceil \frac{\sum_i w_i d_i}{W} \rceil$, a lower bound on the required number stock pieces. For KRICH, the artificial column costs are set to $\alpha\hat{u}$ with $\alpha = 1.2$ and $\hat{u} := w/W$. This value of $\hat{u}$ solves the dual of the LP relaxation of (2.1); it turns out to be a very good estimation of the optimal solution to the Lagrangian dual. As for bundle, we tested two initializations:

- BUNDLE$\frac{1}{m}$: $\hat{u} = 1/m$ to test a poor initialization;
- BUNDLE: $\hat{u} = w/W$, an accurate initialization.

Table 1 CSP: averages on 9 industrial instances with 7–33 items

Master	Counters		Timers (in s)		
	Iter	Cols	Oracle	Master	Total
KBASIC	42	43	0.38	0.03	0.52
KRICH	29	29	20.82	0.01	20.91
BUNDLE $\frac{1}{m}$	90	52	60.86	0.07	70.04
BUNDLE	43	33	42.57	0.03	42.68
BUNDLE+K	38	32	38.20	0.05	38.37
Kelley-Inc	24	49	0.12	0.03	0.25
BUNDLE-Inc	41	56	40.47	0.40	40.63
BUNDLE+K-Inc	25	49	6.33	0.13	6.55

For BUNDLE+K, $\hat{u} = w/W$, and the master used for Kelley includes the same articial column as in KBASIC. We also report on tests where the algorithm is initialized with the columns of an initial incumbent primal solution, obtained via a first-fit decreasing heuristic. These versions are denoted by Kelley-Inc, BUNDLE-Inc, and BUNDLE+K-Inc. Kelley-Inc is initialized with the columns of the heuristic solution only (no artificial columns are used). The counter "cols" in the tables includes these heuristic columns.

Also, the stopping test of Remark 1.3 is strengthened: knowing that the optimal value to (2.1) is integer, the dual bound is rounded-up to the next integer; earlier stop may occur when a primal bound is available (note: such is the case with Kelley but not with bundle).

Table 1 concerns the results on nine industrial instances, whose number m of items ranges from 7 to 33; N is specified in the data. Table 2 gives the same statistics on 20 randomly generated instances, having 50 items each; the average demand is 100 and the widths are drawn uniformly in [500, 5,000], while the wideroll width is 10,000 and $N = 2,500$. The fastest methods seem to be KBASIC and Kelley-Inc, while the fewer iterations are with Kelley-Inc and BUNDLE+K-Inc. An important observation is that the number of calls to the oracle and the total time spent in the oracle are somehow anti-correlated: more time is spent in the oracle when it is called by a stabilized method. In fact, the more expensive oracles are those called by the end of the algorithms, when u and w tend to be collinear vectors: this behaviour is illustrated by Fig. 6, obtained with one of the random instances. A final observation is that the times spent in the master are (small and) comparable for all variants, whether this master is the LP (1.6) or the QP (1.30).

In another series of runs, we consider a variant of CSP: the number of items j to cut must take a value within a prescribed interval $[\underline{d}_j, \overline{d}_j]$. The objective is then to minimize the waste (this would be equivalent to minimizing the number of stock pieces if the demands were fixed). The resulting master takes the form

$$\min \sum_{i\in I}(W - wz^i)\lambda_i, \qquad \underline{d} \leqslant \sum_{i\in I} z_j\lambda_i \leqslant \overline{d}, \quad \lambda_i \geqslant 0, \ \ i \in I.$$

Table 2 CSP: averages on 20 random instances with 50 items

Master	Counters		Timers (in s)		
	Iter	Cols	Oracle	Master	Total
KBASIC	235	236	3.45	0.50	4.51
KRICH	142	142	3.15	0.21	3.72
BUNDLE $\frac{1}{m}$	246	185	12.30	0.54	13.37
BUNDLE	163	111	26.51	0.27	27.17
BUNDLE+K	155	109	25.16	0.39	26.08
Kelley-Inc	136	211	3.36	0.20	4.61
BUNDLE-Inc	164	186	26.94	0.40	28.70
BUNDLE+K-Inc	131	170	20.18	0.46	22.00

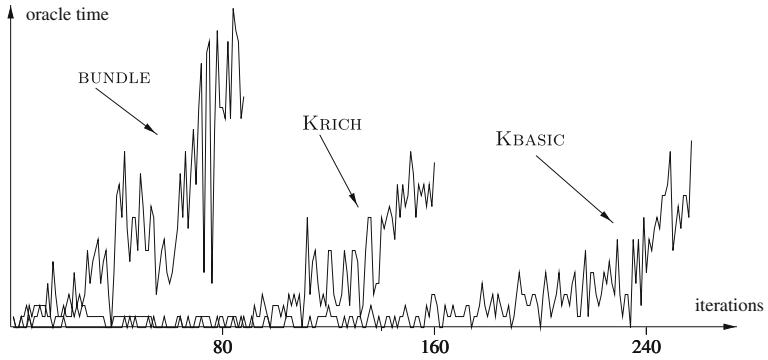

Fig. 6 CSP: oracle computing time along iterations

In this variant, convergence is no longer truncated by rounding dual bounds (the objective value is no longer an integer). The test problems are the same, but with a 5% tolerance on production: $\underline{d}_j = \lfloor 0.95\, d_j \rfloor$ and $\overline{d}_j = \lceil 1.05\, d_j \rceil$.

A possible heuristic to define $\hat{u}$ is as follows. Let R be an upper bound on the optimal waste (obtained by solving the first variant). Then:

- set $u_j = \frac{Rw_j}{\sum_k w_k d_k}$ for the covering constraints $\sum_i z^i \lambda_i \geqslant \underline{d}$;
- set $u_j = \frac{R}{w_j \sum_k \frac{d_k}{w_k}}$ for the packing constraints $\sum_i z^i \lambda_i \leqslant \overline{d}$.

Intuitively, the portion of the waste induced by a covering constraint is proportional to the width of the item, while the waste induced by the production upper bounds is inversely proportional to the usefulness of the item in filling the holes (the more useful items being the small ones).

The results for this CSP variant are given in Tables 3 and 4. In these particular experiments, the QP (1.30) was cold-started at each iteration, even though its $k+1$st execution could take advantage of the kth one; a comparison of the master timers with the previous tables illustrates the benefit of warm starts. For KRICH, we define artificial column cost as $\alpha\hat{u}$ with $\alpha = 1.1$. The initialization BUNDLEW&0 refers to using $\hat{u}$ as defined above for covering constraints

Table 3 CSP, minimization of the waste: averages on nine industrial instances

Master	Counters		Timers (in s)		
	Iter	Cols	Oracle	Master	Total
KBASIC	43	43	2.39	0.03	2.53
KRICH	31	26	8.42	0.03	8.53
BUNDLE	60	35	15.35	0.25	15.71
BUNDLEW&0	48	34	54.19	0.23	54.51
BUNDLE+K	59	36	15.22	0.30	15.66
Kelley-Inc	44	69	4.78	0.05	4.97
bundle-Inc	55	59	19.28	0.30	19.74
BUNDLE+K-Inc	54	59	19.35	0.37	19.87

Table 4 CSP, minimization of the waste: averages on 20 random instances

Master	Counters		Timers (in s)		
	Iter	Cols	Oracle	Master	Total
KBASIC	144	143	8.25	0.29	9.03
KRICH	142	120	32.80	0.29	33.56
BUNDLE	200	86	58.92	5.78	65.15
BUNDLEW&0	203	177	56.83	7.12	65.52
BUNDLE+K	198	87	58.20	5.88	64.65
Kelley-Inc	171	247	7.48	0.44	9.46
bundle-Inc	196	162	58.21	9.11	68.85
BUNDLE+K-Inc	195	162	57.80	9.24	68.65

but using 0 for packing constraints. For BUNDLE and BUNDLE+K, we use $\hat{u}$ as defined above. To obtain an initial incumbent, the first-fit decreasing heuristic is used to cover demands $\underline{d}_j$; then production surplus are used to fill holes in existing cutting patterns. Now the fastest method is KBASIC. Again, supbroblems take longer to solve with stabilized schemes. Nevertheless, the fewer number of iterations is achieved by KRICH. Whatever the initialization, bundle takes more iterates than Kelley.

We now come to the bin-packing problem, which is a special case of the cutting-stock problem where item demands take value 1. However, when an instance involves identical items (items with the same width) their demands are aggregated. Hence, bin-packing instances can be viewed as cutting-stock problems with low average demand (and typically more items). Our test problems are extracted from the OR library [4]. We use 20 random instances with 120 and 250 items, whose sizes are integer and vary between 20 and 100, while the size of the bin is 150. We use also 20 so-called triplets instances with 120 and 249 items, for which the bin capacity is 100, and the items sizes are real and range from 25 to 49.9. The triplets instances are generated from an optimum solution where there are exactly three items in each bin, which fill the bin capacity exactly [hence $\hat{u} = w/W$ is an optimal dual solution to (2.4)]. They are known to be hard to solve for standard column generation algorithms. The results are

Table 5 Bin packing: results are averages on 20 instances

	Master	Counters		Timers (in s)		
		Iter	Cols	Oracle	Master	Total
BP-120	KBASIC	222	223	0.79	0.31	1.79
	KRICH	101	101	0.68	0.19	1.15
	BUNDLE	107	90	0.79	0.23	1.36
BP-250	KBASIC	260	260	1.66	0.50	2.90
	KRICH	134	134	1.35	0.30	2.03
	BUNDLE	112	105	0.82	0.30	1.60
BP-t120	KBASIC	339	340	5.82	0.74	7.83
	KRICH	120	120	3.38	0.34	4.14
	BUNDLE	80	79	2.17	0.20	2.84
BP-t249	KBASIC	553	554	56.56	2.37	61.36
	KRICH	189	189	14.79	0.81	16.49
	BUNDLE	139	138	7.20	0.69	9.09

given in Table 5. The comparison between KBASIC and KRICH shows the impact of the good a priori knowledge of the dual solution. Bundle initialized with $\hat{u} = w/W$ typically takes fewer iterations. It is also the fastest method on the hardest instances.

2.3 The vextex coloring problem (VCP)

Given a graph $G(V, E)$, the coloring problem is to use the minimum number of colors in assigning a color to each vertex so that no adjacent vertices receive the same color. This minimum number of colors $\chi(G)$ is called the *chromatic number*. Let N be an upper bound on $\chi(G)$. To formulate the coloring problem as (1.1), let us introduce variables $y^\ell \in \{0,1\}$ to indicate whether color ℓ is used and $z_j^\ell \in \{0,1\}$ to indicate whether vertex j is assigned color ℓ. Then $\chi(G)$ is the optimal value of

$$\min_{y,z\in\{0,1\}} \sum_{\ell=1}^{N} y^\ell, \quad \sum_{\ell=1}^{N} z_j^\ell \geqslant 1, \quad j \in V, z_i^\ell + z_j^\ell - y^\ell \leqslant 0, \ \ (ij) \in E, \ \ \ell = 1, \ldots, N.$$

We dualize the covering constraints $\sum_{\ell=1}^{N} z_j^\ell \geqslant 1$ and keep the other constraints in X. The Lagrangian (1.9) is then

$$L(y, z, u) = \sum_{\ell=1}^{N} \left(y^\ell - \sum_{j\in V} u_j z_j^\ell \right) + \sum_{j\in V} u_j,$$

resulting in the dual function

$$\theta(u) = \sum_j u_j + \min_{y,z\in\{0,1\}} \sum_\ell \left\{ y^\ell - \sum_j u_j z_j^\ell \ : \ z_i^\ell + z_j^\ell - y^\ell \leqslant 0, \quad (ij) \in E \right\}.$$

The above minimization problem is decomposable with respect to colors ℓ; in fact, θ is obtained as the sum of N identical minimization problems. The value $y^\ell = 0$ produces $z^\ell = 0$, while for $y^\ell = 1, z = z^\ell$ solves a maximum independent set problem on G with weights u:

$$\max_{z\in\{0,1\}} \sum_j u_j z_j, \qquad z_i + z_j \leqslant 1, \quad (ij) \in E.$$

Denoting by $z(u)$ an optimal solution to this problem, the dual function is just as for CSP:

$$\theta(u) = 1u + N \min\{0, 1 - uz(u)\}.$$

Now, let us write the Dantzig-Wolfe master program (1.3): $\min \sum_{i\in I} y^i \lambda_i$, $\sum_{i\in I} z_j^i \lambda_i = 1, j \in V$, $\sum_{i\in I} \lambda_i \leq N$, $\lambda_i \geq 0, i \in I$, where I enumerates the solutions $(y, z) \in X^\ell$. As for CSP, $\sum_{i\in I} \lambda_i \leq N$ are inherited from original convexity constraints. Moreover, the constraint can be dropped given the objective. Thus the useful formulation is

$$\min \left\{ \sum_{i\in I} \lambda_i \ : \ \sum_{i\in I} z_j^i \lambda_i \geqslant 1, \quad j \in V, \quad \lambda_i \geq 0, \ \ i \in I \right\},$$

Then, I enumerates independent sets of the graph G and x^is are indicator vectors of these sets. The dual is:

$$\max\{1u \ : \ uz^i \leq 1, \quad i \in I, \ \ u \geq 0\}.$$

The above column-generation mechanism was given in [43]; it produces the so-called *fractional chromatic number*, denoted by $\chi_F(G)$, which is also an upper bound for the maximum clique number ω.

Numerical tests were performed on two academic families of graphs (the Mycielski and Queen graphs), as well as on real-life instances (register allocation). These graphs were used as benchmarks for the DIMACS challenge.[3] Three methods are compared: KBASIC, KRICH and BUNDLE. The initial $\hat{u}$ for BUNDLE is set to $\hat{u}_j = \deg(j)/(|V| - 1)$ where $\deg(j)$ is the number of neighbours of node j: the dual cost of assigning a color to a vertex connected to every other vertex is 1. KRICH is initialized with artificial variable cost set to $\alpha\hat{u}$ with $\alpha = 1.1$.

[3] http://mat.gsia.cmu.edu/COLOR/instances.html.

Table 6 Vertex coloring: results on Mycielski graphs

Problem	Master	Counters		Timers (in s)		
data		Iter	Cols	Oracle	Master	Total
Mycielski2	KBASIC	8	8	0.01	0.00	0.02
(5,5)	KRICH	6	5	0.00	0.01	0.03
2 – 3 – 2.5	BUNDLE	6	5	0.00	0.00	0.05
Mycielski3	KBASIC	29	29	0.01	0.01	0.09
(11,20)	KRICH	12	11	0.00	0.00	0.07
2 – 4 – 2.9	BUNDLE	19	11	0.04	0.00	0.09
Mycielski4	KBASIC	91	91	0.17	0.12	0.56
(23,71)	KRICH	47	46	0.13	0.03	0.41
2 – 5 – 3.24	BUNDLE	61	25	0.26	0.06	0.46
Mycielski5	KBASIC	264	264	2.79	0.74	4.96
(47,236)	KRICH	177	176	2.83	0.69	5.05
2 – 6 – 3.55	BUNDLE	177	55	4.59	0.53	5.76
Mycielski6	KBASIC	928	928	109.82	14.19	138.84
(95,755)	KRICH	690	690	76.42	8.34	95.78
2 – 7 – 3.83	BUNDLE	447	113	90.86	5.11	101.30
Mycielski6	KBASIC	3,724	3,724	8,875.99	997.05	10,109.26
(191,2360)	KRICH	3,204	3,203	6,704.52	684.32	7,587.72
2 – 8 – 4.10	BUNDLE	1,254	510	7,160.93	28.46	7,217.13

The (NP-Hard) oracle subproblems are solved using a recursive enumeration algorithm [43].

The Mycielski graphs form a sequence of increasingly large graphs with clique number 2 (triangle-free graphs) and increasing chromatic number (for instance, Mycielski 2 is a pentagon). Since the fractional chromatic number is sandwiched between those two numbers, a large gap means a graph hard to color. Table 6 gives the computational results for these instances. Its first column gives, for each problem:

- the name of the problem,
- (number of nodes , number of edges),
- the clique, chromatic and fractional chromatic numbers: $\omega - \chi - \chi_F$.

Compared to KBASIC, the rough initialization of KRICH helps to reduce the computational time and the number of iterations. The computing time for the bundle method is competitive compared to KRICH, with a gain in solving the master.

The Queen graphs arise from the puzzle of placing as many queens as possible on an $n \times n$ chessboard so that no two queens are on the same row, column or diagonal : graphs are constructed so that the answer to the puzzle correspond to a maximum independent set on the graph. The results on these highly symmetric instances are presented in Table 7, which reads as Table 6. Once again, stabilisation decreases stubstantially the number of iterations; but observe the corresponding increase in oracle's computing time, which can be dramatic.

In Table 8, we compare the three methods on 50-vertex subgraphs extracted from real instances arising in register allocation: assign variables of a program to high-speed access memory registers. On these graphs, computing times are

Table 7 Vertex coloring: results on Queen graphs

Problem	Master	Counters		Timers (in s)		
data		Iter	Cols	Oracle	Master	Total
Queen 5x5	KBASIC	60	61	0.14	0.03	0.25
(25,160)	KRICH	6	6	0.01	0.01	0.08
5 – 5 – 5	BUNDLE	11	7	0.05	0.01	0.13
Queen 6x6	KBASIC	84	84	0.25	0.12	0.59
(36,290)	KRICH	53	53	0.27	0.18	0.70
6 – 7 – 7	BUNDLE	44	41	0.62	0.07	0.85
Queen 7x7	KBASIC	141	142	1.15	0.32	1.86
(49,476)	KRICH	206	206	3.65	1.11	5.83
7 – 7 – 7	BUNDLE	14	11	1.08	0.05	1.25
Queen 8x8	KBASIC	197	198	10.23	0.67	11.64
(64,728)	KRICH	134	134	8.83	0.40	9.96
8 – 9 – 8.44	BUNDLE	74	70	30.98	0.28	31.82
Queen 9x9	KBASIC	351	351	117.26	2.13	121.26
(81,2112)	KRICH	293	292	142.40	1.80	145.88
9 – 9 – 9	BUNDLE	102	96	334.76	0.55	336.37
Queen 10x10	KBASIC	457	458	1,177.39	4.60	1,185.11
(100,2940)	KRICH	349	349	1,063.42	3.65	1,069.64
10 – 10 – 10	BUNDLE	312	156	6,759.60	2.26	6,764.44
Queen 11x11	KBASIC	510	511	5,912.88	8.01	5,925.31
(121,3960)	KRICH	374	374	5,457.54	6.07	5,467.19
11 – 11 – 11	BUNDLE	125	124	13,017.26	3.88	13,025.21

much shorter using KBASIC. For KRICH, there are typically more vertices with strictly positive reward to consider when solving the subproblem, implying even more computing time than for BUNDLE. Moreover, the initial $\hat{u}$ does not reflect the structure of the optimal dual solution, which typically involves three classes of vertices: those that receive weight 1 (as those in the largest clique), those with zero weight, and the "undecided". Thus, several phases are required to eliminate all artificial columns from the solution.

2.4 The capacitated vehicle routing problem (CVRP)

Given a set of m customers, indexed by $j = 1, \ldots, m$, demanding a delivery d_j, and N identical vehicles, indexed by $\ell = 1, \ldots, N$, available to make these deliveries, the problem is to determine a route for each vehicle satisfying the following constraints:

- the route starts and ends at the depot (denoted by index 0);
- the deliveries assigned to a vehicle cannot exceed its capacity C;
- each customer j must be delivered d_j by a single vehicle.

The distance matrix $\{c_{ij}\}_{0 \leqslant i,j \leqslant m}$ between customer sites is known (the distances are assumed to be integer and triangular inequalities hold). The goal is to minimize the sum of the lengths of the N routes.

Let $y_j^\ell = 1$ if customer j is visited by vehicle ℓ; let $z_{ij}^\ell = 1$ if arc (i,j) is in the route of vehicle ℓ and zero otherwise; call q_j^ℓ the cumulated load of vehicle ℓ on departure from customer j. Then a possible formulation for (1.1) is:

$$\begin{cases} \min \sum_{i,j,\ell} c_{ij} z_{ij}^\ell & \\ \sum_{\ell=1}^N y_j^\ell \geqslant 1 & \text{for } j > 0 \qquad (*) \\ \sum_i z_{ij}^\ell = \sum_i z_{ji}^\ell & \text{for all } j, \ell \\ \sum_i z_{ij}^\ell = y_j^\ell & \text{for all } j, \ell \\ q_i^\ell - q_j^\ell + C z_{ij}^\ell + d_j y_j^\ell \leqslant C & \text{for } i \neq j > 0 \text{ and all } \ell \\ d_j y_j^\ell \leqslant q_j^\ell \leqslant C y_j^\ell & \text{for } j > 0 \text{ and all } \ell \\ q_i^\ell \geqslant 0 & \text{for } j > 0 \text{ and all } \ell \\ y_j^\ell, z_{ij}^\ell \in \{0,1\} & \text{for } i \neq j > 0 \text{ and all } \ell \end{cases}$$

Introducing the variable vector $x = (q, y, z)$, call $Ax \geqslant b$ the cover[4] constraints $(*)$; the other constraints are put in X. This gives the Lagrangian dual function (1.10)

$$\theta(u) = \sum_{j=1}^m u_j + \min_{(q,y,z)\in X} \sum_{\ell=1}^N \left(\sum_{i,j} c_{ij} z_{ij}^\ell - \sum_j u_j y_j^\ell \right)$$

which is decomposable with respect to ℓ: X is the Cartesian product of N identical subsets X^ℓ. The optimization sub-problem, $\min_{(q,y,z)\in X^\ell} \sum_{i,j} c_{ij} z_{ij} - \sum_j u_j y_j$ (where the useless index ℓ is dropped), is an elementary shortest path problem with resource constraints; it can be solved by dynamic programming [13]. Denoting its optimal solution by $x(u) = (q(u), y(u), z(u))$, the dual function has value:

$$\theta(u) = 1u + N[c\, z(u) - u\, y(u)]$$

An alternative presentation of the Lagrangian dual problem is Dantzig-Wolfe formulation (1.3). Here the master is:

$$\min \left\{ \sum_{i\in I} \left(\sum_{kj} c_{kj} z_{kj}^i \right) \lambda_i : \quad \sum_{i\in I} y_j^i \lambda_i \geq 1 \ \ \forall j, \quad \sum_{i\in I} \lambda_i \leq N, \quad \lambda_i \geq 0, \ \ i \in I \right\}$$

where I enumerates solutions $x^i = (q^i, y^i, z^i)$ of X^ℓ, i.e. feasible vehicle routes. Its dual takes the form:

$$\max \left\{ 1\, u - Nr : \quad u y^i - r \leq c\, z^i, \quad i \in I, \ \ (u,r) \in \mathbb{R}_+^m \times \mathbb{R}_+ \right\}.$$

[4] Covering contraints may be used instead of partitioning because triangular inequalities guarantee that a customer will not be visited more than once in an optimal solution.

Table 8 Vertex coloring: results on register-allocation problems

Register	Counters		Timers (in s)		
instances	Iter	Cols	Oracle	Master	Total
fpsol2.i.1	88	89	0.22	0.14	0.63
fpsol2.i.2	76	77	0.16	0.09	0.48
fpsol2.i.3	76	77	0.21	0.06	0.51
mulsol.i.1	125	126	0.40	0.16	0.90
mulsol.i.2	93	93	0.23	0.14	0.67
mulsol.i.3	93	93	0.24	0.11	0.61
mulsol.i.4	93	93	0.24	0.07	0.60
mulsol.i.5	93	93	0.25	0.10	0.63
zeroin.i.1	75	75	0.19	0.09	0.55
zeroin.i.2	75	75	0.14	0.10	0.49
zeroin.i.3	75	75	0.16	0.08	0.51
KBASIC average	87.5	87.8	0.22	0.10	0.60
fpsol2.i.1	117	106	3.31	0.21	3.78
fpsol2.i.2	142	114	7.85	0.26	8.50
fpsol2.i.3	142	114	7.88	0.22	8.38
mulsol.i.1	138	128	1.33	0.26	1.84
mulsol.i.2	276	247	2.13	0.45	3.28
mulsol.i.3	276	247	2.27	0.49	3.45
mulsol.i.4	276	247	2.25	0.51	3.43
mulsol.i.5	276	247	2.15	0.59	3.41
zeroin.i.1	146	134	4.89	0.25	5.50
zeroin.i.2	146	134	4.90	0.27	5.49
zeroin.i.3	146	134	4.85	0.22	5.42
KRICH average	189.2	168.4	3.98	0.34	4.77
fpsol2.i.1	62	58	0.61	0.09	0.92
fpsol2.i.2	71	62	0.62	0.15	1.00
fpsol2.i.3	71	62	0.68	0.19	1.03
mulsol.i.1	58	52	0.53	0.13	0.84
mulsol.i.2	51	40	0.41	0.11	0.68
mulsol.i.3	51	40	0.50	0.06	0.70
mulsol.i.4	51	40	0.46	0.11	0.70
mulsol.i.5	51	40	0.43	0.08	0.70
zeroin.i.1	64	54	0.60	0.14	0.93
zeroin.i.2	64	54	0.61	0.10	0.90
zeroin.i.3	64	54	0.59	0.13	0.90
BUNDLE average	59.8	50.5	0.55	0.12	0.85

Kelley and bundle are compared on 12 test problems from the website [2]. The three P-instances and the E-instance are just those of [2]. The eight remaining ones are reduced B-instances of [2], obtained by selecting a subset of N routes from the optimal solution and restricting the problem to the associated customers. The sizes m and N are provided in the problem names.

To define artificial column costs, we take an estimate edge cost $\bar{c}_i$ for connecting client i, and then use

$$B := N\frac{\sum_j (c_{0j} + c_{j0})}{m} + \frac{m-N}{m}\sum_i \bar{c}_i,$$

which estimates the optimal cost of an integer solution. Indeed, a solution can be seen as a TSP tour visiting all customers and making N detours by the depot; for the inter customer distance, we take the average value $\frac{\sum_i \bar{c}_i}{m}$; for the cost of the detour to the depot, we take the average value $\frac{\sum_j (c_{0j}+c_{j0})}{m}$. One could take $\bar{c}_i$ to be the average edge cost on, say, the four closest neighbors. A finer estimate is to take into account that customer with high demand are harder to accomodate in a cluster and, as a result, can be paired up with customer that are further away. Hence, we define $\bar{c}_i = \frac{1}{\beta_i} \sum_{j=1}^{\beta_i} c_{ij}$ where the neighbors, j, are assumed to be sorted in non-decreasing order of the edge costs, $\beta_i = 2(1+\lceil \frac{d_i}{\bar{d}} \rceil)$, and $\bar{d}$ is the average demand.

Seven versions are tested:

- KBASIC, where the master is initialized with an artificial column of cost equal to the above estimation B.
- KRICH initialized with m artificial columns associated with each client i. Their cost represents the fraction of total cost estimate B that is attributed to client j, i.e.

$$\hat{u}_j = \frac{w_j}{\sum_{k=1}^{m} w_k} B$$

  where the weight factors are defined as $w_i = \bar{c}_i + \frac{1}{1+\frac{C-d_i}{\bar{d}}} c_{0i}$ to reflect a larger portion of the cost to customer with high connection cost, high demand (and thus lower number of neighbors with whom to share the cost of travelling from and to the depot) and that are far from the depot. When artificial columns remain in the solution we multiply them by $\alpha = 1.2$.
- BUNDLE0, initialized at $\hat{u} = 0$.
- BUNDLE, initialized at $\hat{u}$ defined as for KRICH.
- KRICHMC is KRICH with multi-column generation for each subproblem: we insert in the master all feasible routes (or up to 500) found by the dynamic programming solver for the subproblem (this is standard practice when solving VRP by column generation).
- Kelley-Inc is initialized with a primal heuristic solution obtained using Clarke and Wright's iterative route merging procedure (the artificial column of KBASIC is also included in case the heuristic solution is not feasible: i.e. if it uses too many vehicles).
- bundle-Inc, initialized at $\hat{u}$ defined as for KRICH with an initial bundle made of the columns of Clarke and Wright's solution.

We also tested BUNDLE+K but the results hardly differ from the use of bundle alone.

Table 9 gives the average results. Details are given in Table 10 (for the first 4 instances) and Table 11 (the 8 remaining ones). These latter two tables explain in particular BUNDLE0's large average computing time, which occurs in some instances only. Version KRICH is the fastest while KRICHMC needs fewer

Table 9 Average results for CVRP instances

Averages	Iter	Cols	Oracle	Master	Total
KBASIC	41	41	3.82	0.10	4.10
KRICH	17	14	0.74	0.08	0.91
KRICHMC	7	136	0.85	0.07	1.21
BUNDLE0	25	18	86.14	0.05	86.32
BUNDLE	24	13	1.58	0.06	1.73
Kelley-Inc	31	36	1.40	0.09	1.66
bundle-Inc	24	18	1.58	0.06	1.75

Table 10 Detailed comparison on CVRP

Problem	Method	Counters		Timers (s)		
		Iter	Cols	Oracle	Master	Total
	KBASIC	29	29	0.15	0.06	0.31
	KRICH	18	17	0.04	0.06	0.19
P-m16-N8	BUNDLE0	30	22	0.08	0.06	0.24
	BUNDLE	40	20	0.18	0.10	0.42
	KRICHMC	6	42	0.07	0.03	0.18
	KBASIC	57	57	0.54	0.18	0.90
	KRICH	37	31	0.38	0.17	0.70
P-m22-N8	BUNDLE0	46	34	0.72	0.16	10.10
	BUNDLE	38	26	0.65	0.11	0.93
	KRICHMC	17	125	0.61	0.12	10.06
	KBASIC	65	62	0.72	0.26	1.21
	KRICH	46	39	0.70	0.53	1.42
P-m23-N8	BUNDLE0	70	39	1.51	0.25	2.11
	BUNDLE	69	31	1.67	0.24	2.13
	KRICHMC	22	145	1.13	0.47	0.02
	KBASIC	83	83	15.41	0.32	16.14
	KRICH	37	35	3.37	0.11	3.71
E-m22-N4	BUNDLE0	35	30	745.39	0.05	745.65
	BUNDLE	44	23	5.58	0.14	5.85
	KRICHMC	10	231	1.90	0.10	2.78

iterations. However, these results are sensitive to the fine tuning of the cost estimate and resulting value of $\hat{u}$ (we previously tested rough estimates that yielded more iterations for KRICH). By contrast, bundle is more robust: the number of iterations does not suffer from a poor initialization (although the time spent in the oracle does).

Most of the computing time is spent in the oracles. The resource-constrained shortest path problem solver of [13] enumerates many partial solutions when the rewards u_j are high and many clients seem attractive to be included in the route. We noted experimentally that our initial $\hat{u}$ overestimates on average the optimal dual solution: for most customers the dual values decrease in the course of the column generation procedure. Hence, the most expensive oracle calls tend to be at the beginning of the column generation procedure. For KBASIC

Table 11 Detailed comparison on reduced CVRP

Problem	Method	Counters		Timers (s)		
		Iter	Cols	Oracle	Master	Total
	KBASIC	21	21	0.07	0.04	0.17
	KRICH	9	8	0.06	0.03	0.12
gen-B-m8-N2	BUNDLE0	16	10	0.13	0.04	0.22
	BUNDLE	12	7	0.10	0.01	0.16
	KRICHMC	6	73	0.27	0.04	0.44
	KBASIC	40	40	3.43	0.09	3.72
	KRICH	12	11	1.45	0.04	1.54
gen-B-m12-N2	BUNDLE0	23	20	748	0.03	763
	BUNDLE	25	12	3.55	0.03	3.68
	KRICHMC	7	282	2	0.01	2.68
	KBASIC	34	34	2.57	0.03	2.74
	KRICH	5	3	0.15	0.01	0.19
gen-B-m12-N2	BUNDLE0	15	12	3.78	0.02	3.86
	BUNDLE	10	4	0.43	0.01	0.49
	KRICHMC	4	81	0.31	0.02	0.41
	KBASIC	40	40	6.37	0.05	6.67
	KRICH	4	3	0.19	0	0.26
gen-B-m14-N2	BUNDLE0	9	6	215.47	0	215.51
	BUNDLE	6	3	0.45	0	0.52
	KRICHMC	2	71	0.23	0	0.30
	KBASIC	29	29	0.61	0.06	0.76
	KRICH	6	5	0.21	0.01	0.26
gen-B-m10-N2	BUNDLE0	9	6	75	0.02	80
	BUNDLE	14	7	1.52	0.03	1.60
	KRICHMC	4	125	0.69	0.02	0.83
	KBASIC	46	46	15.68	0.09	16.03
	KRICH	6	5	1.79	0.01	1.85
gen-B-m13-N2	BUNDLE0	17	14	57.29	0.02	57.42
	BUNDLE	5	2	3.52	0.01	3.57
	KRICHMC	4	224	2.02	0.01	2.33
	KBASIC	27	28	0.22	0.04	0.35
	KRICH	13	12	0.32	0.06	0.42
gen-B-m10-N2	BUNDLE0	21	17	0.69	0.03	0.80
	BUNDLE	23	13	0.95	0.03	1.03
	KRICHMC	6	138	0.45	0.02	0.72
	KBASIC	25	25	0.15	0.01	0.29
	KRICH	13	10	0.24	0.03	0.30
gen-B-m10-N2	BUNDLE0	19	14	0.42	0.03	0.53
	BUNDLE	13	8	0.40	0.01	0.44
	KRICHMC	7	106	0.55	0	0.80

and BUNDLE0 however, many rewards u_j are very small in the initial iterations (initial extreme dual solutions under KBASIC concentrate all the reward on a few clients) and the oracle preprocessor removes the unattractive nodes, making the problem easier. After a few timid iterations, BUNDLE0 increases the u's largely and uniformly, which yields expensive oracles. This behaviour is illustrated by Fig. 7.

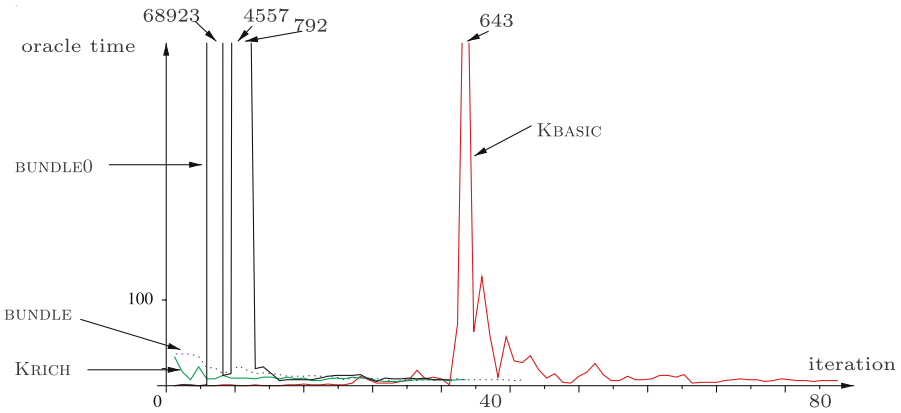

Fig. 7 CVRP: oracle computing time for instance E-m22-N4

2.5 The multi-item lot-sizing (MILS) problem

The problem is to optimize the production planning of items $\ell = 1, \ldots, N$ for time periods $j = 1, \ldots, m$. Before production can take place, the machine must be specifically setup for the item (at a cost f_j^ℓ and with a loss of production capacity b_j^ℓ). The other costs associated with item ℓ are unit production costs p_j^ℓ and holding costs h_j^ℓ. Item demands for each period j are denoted by d_j^ℓ and the machine production capacity in period j is C_j. Set the binary variable y_j^ℓ to 1 if the machine is setup for item ℓ in period j. Let z_j^ℓ be the production of item ℓ in period j and s_j^ℓ the stock of ℓ at the end of period j.

Then (1.1) takes the form:

$$\begin{cases} \min \sum_{j,\ell} (f_j^\ell y_j^\ell + p_j^\ell z_j^\ell + h_j^\ell s_j^\ell) & \\ \sum_{\ell=1}^N z_j^\ell + b_j^\ell y_j^\ell \leqslant C_j & \text{for all } j, \qquad (*) \\ s_{j-1}^\ell + z_j^\ell = d_j^\ell + s_j^\ell & \text{for all } j, \ell, \\ z_j^\ell \leqslant D_j^\ell y_j^\ell & \text{for all } j, \ell, \\ z_j^\ell, s_j^\ell \geqslant 0 & \text{for all } j, \ell, \\ y_j^\ell \in \{0,1\} & \text{for all } j, \ell. \end{cases}$$

Here $D_j^\ell = \sum_{\tau=j}^m d_\tau^\ell$; the corresponding constraint forces $z_j^\ell = 0$ unless $y_j^\ell = 1$.

Introducing the variable vector $x = (y, z, s)$, call $Ax \geqslant b$ the capacity constraints $(*)$ and put the other constraints in X; this gives the Lagrangian dual function (1.10)

$$\theta(u) = -\sum_j u_j C_j + \min_{(y,z,s)\in X} \sum_j^\ell \left[(f_j^\ell + b_j^\ell u_j) y_j^\ell + (p_j^\ell + u_j) z_j^\ell + h_j^\ell s_j^\ell \right]$$

which is decomposable with respect to ℓ: X is the Cartesian product of N non-identical subsets X^ℓ. Each optimization sub-problem

$$\theta^\ell(u) := \min_{(y,z,s)\in X^\ell} \sum_t \big[(f_j + b_j u_j) y_j + (p_j + u_j) z_j + h_j s_j\big] \tag{2.6}$$

(where the useless index ℓ is dropped) is a single-item lot-sizing problem with unbounded capacity; it can be solved by dynamic programming in polynomial time. The dual function is then

$$\theta(u) = \sum_{\ell=1}^{N} \theta^\ell(u) - \sum_{j=1}^{m} C_j u_j.$$

The master formulation (1.3) takes the form:

$$\min \left\{ \sum_{\ell, i\in I^\ell} \left(\sum_j (f_j^\ell y_j^i + p_j^\ell z_j^i + h_j^\ell s_j^i) \right) \lambda_i^\ell \right.$$

$$\sum_{\ell, i\in I^\ell} (z_j^i + b_j^\ell\, y_j^i)\, \lambda_i^\ell \leqslant C_j \quad \forall j, \tag{2.7}$$

$$\left. \sum_{i\in I^\ell} \lambda_i^\ell \geqslant 1 \quad \forall \ell, \quad \lambda_i^\ell \geqslant 0 \ \ \forall \ell, i \in I^\ell \right\}, \tag{2.8}$$

where I^ℓ enumerates production plans $x^i = (y^i, z^i, s^i)$ of X^ℓ, i.e. $X^\ell = \{x^i\}_{i\in I^\ell}$. Its dual takes the form:

$$\max \left\{ \sum_{\ell=1}^{N} r_\ell - \sum_j u_j C_j : r_\ell - \sum_j u_j (z_j^i + b_j^\ell y_j^i) \right.$$

$$\left. \leqslant \left(f_j^\ell y_j^i + p_j^\ell z_j^i + h_j^\ell s_j^i \right) \quad \forall \ell,\ i \in I^\ell,\ \ (u, r) \in \mathbb{R}_+^m \times \mathbb{R}_+^N \right\}.$$

This model has an important implementation feature: in view of (2.6), the dual function θ is a sum of N *different* concave functions θ^ℓ. The approximation θ^k of (1.13) at iteration k can therefore be *disaggregated* in the sum of N "restricted local" dual functions

$$\theta_k^\ell(u) := \min_{i=1,\ldots,k} L^\ell(x^i, u);$$

here $L^\ell(x^i, u)$ denotes the Lagrangian corresponding to product ℓ. Such a disaggregation results in a more accurate restricted dual function. In our experiments, Kelley uses disaggregation; but, even though this would be possible for bundle (as described and assessed for example in [40]), we have not implemented the

Table 12 MILS: average results on 10 random instances with 20 items and 60 periods

Master	Counters			Timers (s)		
	Iter	Sp	Cols	Oracle	Master	Total
KBASIC	9	186	118	0.72	0.01	2.63
KRICH	9	180	90	0.67	0.01	2.50
BUNDLE0	37	754	80	3.94	0.06	5.64
BUNDLE	53	1,062	303	5.59	0.07	9.95

corresponding software. It should be mentioned that the use of disaggregation in bundle would probably result in fewer iterations and more CPU time in the master.

In Table 12, we compare Kelley and bundle on randomly generated instances with $N = 20$ items and $m = 60$ periods. Each line is an average over 10 instances. For version KBASIC, the master is initialized with N artificial columns—one for each subproblem—with cost equal to a large constant: the artificial column associated with item ℓ has coefficient 1 in the associated item covering constraint (2.8) and zero elsewhere. For version KRICH, the master is initialized with $m+N$ artificial columns—one for each capacity constraint (2.7) and one for each item covering constraint (2.8)—with coefficient 1 in the associated constraint and zero elsewhere. Their cost is an estimation of the dual value associated with the constraint: for $\hat{r}_\ell$, we use the cost of the production planning solving the single-item lot-sizing problem with unbounded capacity for item ℓ; $\sum_\ell \hat{r}_\ell$ defines a lower bound on the multi-item lot-sizing problem; we then arbitrarily assume that the capacity constraints will yield a 20% cost increase over this bound and we define $\hat{u}_j = \frac{0.2 \sum_\ell \hat{r}_\ell}{mC_j}$; the artificial variable costs are set to $\alpha\hat{u}_j$ and $\alpha\hat{r}_\ell$ respectively, with $\alpha = 1.2$. The above $\hat{u}$ is used to initialize BUNDLE, while the initial $\hat{u}$ is 0 for BUNDLE0. Counter "Sp" indicates the total number of subproblems solved; the reported number "col" of different columns is likewise disaggregated (for KBASIC, cols includes $N = 20$ artificial columns).

2.6 The traveling salesman problem (TSP)

Given a complete undirected graph $G = (V, E)$ and length c_e for each edge $e \in E$, the problem is to find a tour of minimum length. For $S \subset V$, let $\delta(S)$ be the set of edges $\{i, j\}$ with $i \in S$ and $j \notin S$, and let $E(S)$ be the set of edges $\{i, j\}$ with $i \in S$ and $j \in S$. To make our notation consistent with the rest of the paper, we set $V = \{0, 1, \ldots, m\}$; a possible formulation for (1.1) is

$$
\begin{cases}
\min \sum_{e \in E} c_e x_e, & \\
\sum_{e \in \delta(\{j\})} x_e = 2 & \text{for } j = 0, 1, \ldots, m\,, \qquad (*) \\
\sum_{e \in E(S)} x_e \leqslant |S| - 1 & \text{for all } \emptyset \neq S \subseteq \{1, \ldots, m\}\,, \\
\sum_{e \notin \delta(\{0\})} x_e = n - 2, & \\
x_e \in \{0, 1\} & \text{for } e \in E.
\end{cases}
$$

Call $Ax \geqslant b$ those constraints $(*)$ with $j > 0$ (they can in fact be replaced by inequalities) and put the other constraints in X: the dual variables are $u_1, \ldots, u_m$. For notational convenience, we set $u_0 = 0$, and the Lagrangian is

$$L(x,u) = \sum_{e=\{i,j\}\in E} (c_e - u_i - u_j)x_e + 2\sum_{j=1}^{m} u_j. \tag{2.9}$$

This relaxation was introduced by M. Held and R. Karp: [20,21].

In the definition of X, no two variables x_e and $x_{e'}$ appear in the same constraint if $e \in \delta(\{0\})$ and $e' \notin \delta(\{0\})$. As a result, the solutions of the oracle are the minimum-weight 1-trees, i.e. the minimum-weight spanning trees on $\{1, \ldots, m\}$ completed by the two minimum-weight edges of $\delta(\{0\})$. The oracle is then an easy problem which can be solved by a Prim's algorithm [50].

Let I enumerate the 1-trees, i.e., the solutions $x^i \in X$. Then, the master program (1.3) of a Dantzig–Wolfe reformulation is

$$\begin{cases} \min \sum_{i\in I} \lambda_i \sum_{e\in E} c_e x_e^i, & \\ \sum_{i\in I} \lambda_i \sum_{e\in\delta(\{j\})} x_e^i = 2 & \text{for all } j \in V \setminus \{0\}\,, \\ \sum_{i\in I} \lambda_i \leqslant 1, & \\ \lambda_i \geqslant 0 & \text{for all } i \in I. \end{cases}$$

Its dual (1.12) takes the form

$$\begin{cases} \max\ 2u - r \quad (u,r) \in \mathbb{R}^m_+\,, & \\ \sum_{j\in V\setminus\{0\}} u_j \sum_{e\in\delta(\{j\})} x_e^i - r \leqslant \sum_{e\in E} c_e x_e^i & \text{for all } i \in I. \end{cases}$$

We have applied KBASIC, KRICH and BUNDLE0 on three instances from tsplib[5] having 29, 76 and 442 cities, respectively. For KRICH, we compute

$$\hat{u}_j := \min \left\{ \sum_{e\in\delta(\{j\})} c_e x_e \ :\ \sum_{e\in\delta(\{j\})} x_e = 2,\ x_e \in \{0,1\} \quad \text{for } e \in \delta(\{j\}) \right\};$$

then we define m artificial columns with cost $\alpha\hat{u}$ for $\alpha = 1.2$. The results are reported in Figs. 8, 9, 10. On the third example (pcb442), no convergence can be obtained from Kelley, even with a rich initialization. The reported primal-dual gap is the value $c\hat{x} - \theta(u)$, but with a $\hat{x}$ that may still involve artificial columns (then $c\hat{x}$ is not a true primal bound).

Finally, Fig. 11 displays the 1-trees computed by the oracle during the first 39 iterations of both versions of Kelley, and the 31 iterations of BUNDLE0. Many of the 1-trees computed by KBASIC are star-shaped. In fact, the dual variables are extreme points of the epigraph of the restricted dual function θ^k. During the first iterations, θ^k is a "simple" function and most of these extreme points

[5] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

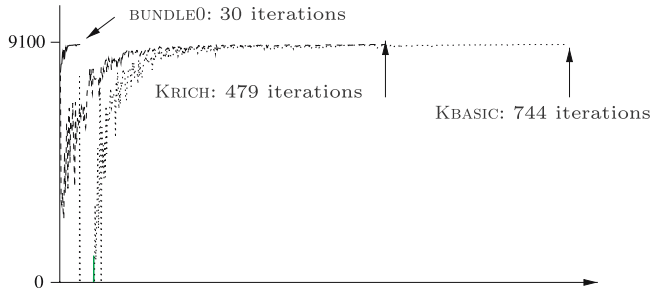

Fig. 8 TSP: Kelley versus bundle on bays29

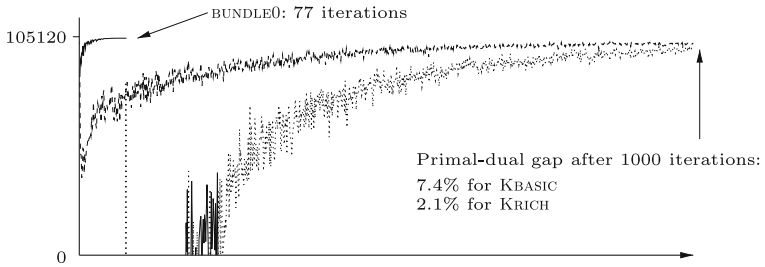

Fig. 9 TSP: Kelley versus bundle on pr76

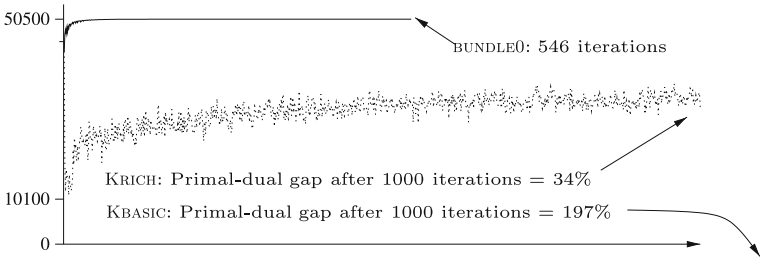

Fig. 10 TSP: Kelley versus bundle on pcb442

are those introduced by the artificial column x^1: they have only one nonzero coordinate, probably rather large. As seen in (2.9), the edges issued from the corresponding node have a rather low cost and are therefore chosen by priority.

3 Additional comparisons: the volume algorithm

To complete our study, we present a rudimentary comparison with the volume algorithm [3], which computes the dual iterates as

$$u^k = \hat{u} + s^k(b - Ax^k) + d^k; \tag{3.1}$$

here s^k is a stepsize and d^k is a suitable correction, updated by simple recurrence formulae. Being based on the subgradient paradigm, this method has

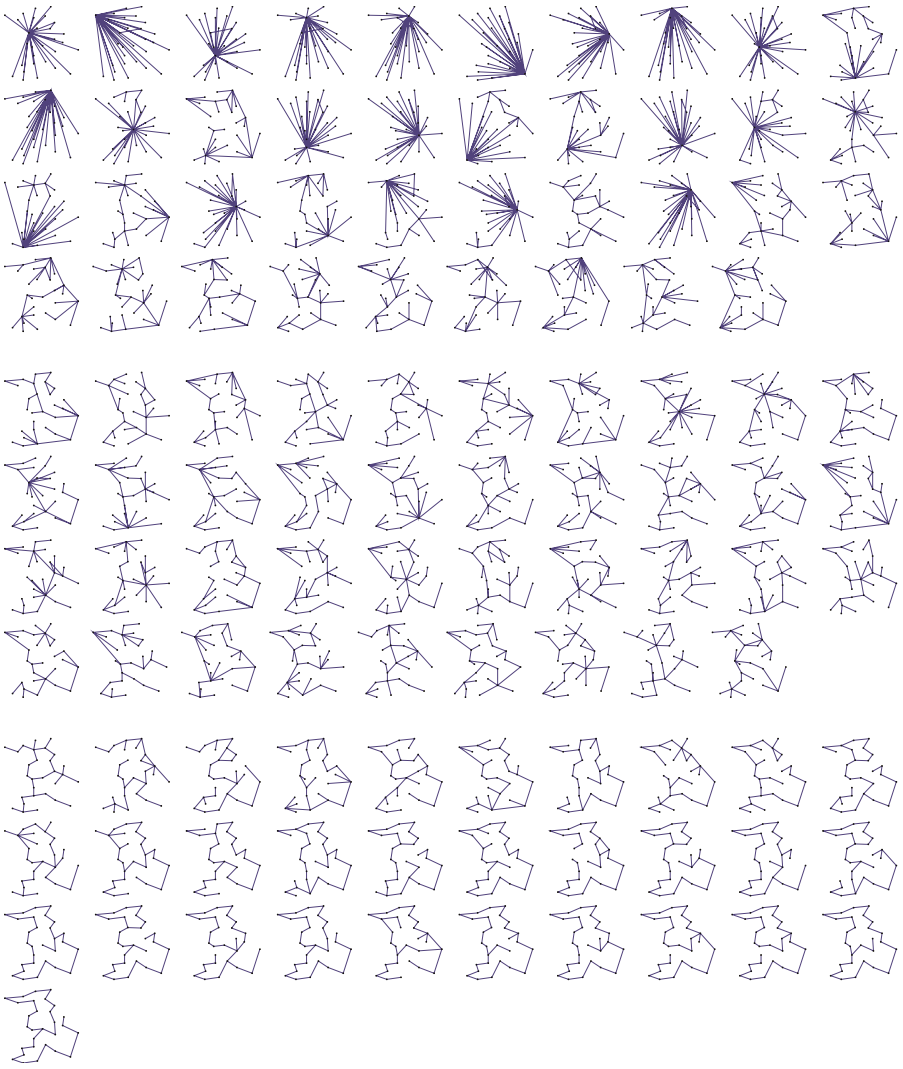

Fig. 11 TSP: primal behaviour on Bays29 for KBASIC (*top*), KRICH (*middle*) and BUNDLE0 (*bottom*)

apparently little to do with stabilized column generation; however there are some analogies:

– Bundle can indeed be viewed as an improved subgradient method: economic versions can be defined, in which each X^k in (1.30) has only two vertices: $X^k = [\tilde{x}, x^k]$; then the iterates are computed analogously to (3.1) (except that the corrections d^k are not described explicitly: we still have to solve a quadratic program—with two variables, though). This aspect of the method is totally overlooked in the present paper but see [37, Sect. 5.3].

– As shown in [33], the volume algorithm can somehow be viewed as a variant resembling the above bundle form.

We use the volume implementation available in the COIN-OR library [8], downloaded from https://projects.coin-or.org/Vol; it will be referred to as VOLUME below. We compare it to BUNDLE0 because it apparently uses the initialization $u^1 = 0$. On the other hand, the volume algorithm has no stopping test; and we could not find a reliable way to stop VOLUME; we therefore just compare the evolution of the dual function $\theta(u^k)$ along the iterations. This is a rather primitive way of comparing methods that are so different; but a detailed comparison of computing times would be hazardous. Taking into account their structural differences (VOLUME is written in C++ and we have not tried to optimize transfers with the oracle, which is in Fortran; BUNDLE0 is totally in Fortran) we can only say that the time of a whole run is roughly of the same order of magnitude for both methods—and see Remark 3.1 below for the oracle computing time.

Our comparison uses two families of problems: CSP and TSP, with larger instances than in Sect. 2. For each family, we have scanned a number of values for the optional parameters of VOLUME; differences may be substantial, and our results below report on the best sets only.

3.1 CSP with volume

A first series of tests picks Beasley's bin-packing problems from Table 5. Actually, these are made genuine CSP, items of identical width being merged into one item, whose demand is appropriately multiplied. The optional parameters input to VOLUME are `greentestinvl` = 2, `yellowtestinvl` = 1, `redtestinvl` = 6 and `alphaint` = 200. Both algorithms behave consistently on all problems tested: for the BP-t501 series [resp. BP-t249, resp. BP-1000],

– in some 500 iterations [resp. 350, resp. 150], BUNDLE0 converges to the optimal solution and proves its optimality,
– in some 2,000 iterations [resp. 1,500, resp. 1,500], VOLUME reaches optimality within less than 1%—enough to get the correct integer lower bound.

We also used cutting-stock examples from [30]: with $m = 50$ items of Gau-Waescher [15], and medium-size instances with $m = 100$ items of Degraeve-Peeters [10]. Here, best parameters are `greentestinvl` = 3, `yellowtestinvl` = 1, `redtestinvl` = 9, `alphaint` = 200.

Table 13 gives an idea of what happens, on a few particular instances extracted from the whole sets. For example, the first line of this table concerns instance 14 of BP-t501 (the largest instance in Beasley's set, with $m = 203$ items) and says the following: VOLUME attains the value 166.3 at iteration 1991; BUNDLE0 attains this same value at iteration 522, and then proves at iteration 547 that the optimal value is 167 (remember that the initial value $\theta(u^1)$ is $\theta(0) = 0$; VOLUME eventually stops at iteration 2,001). Figure 12 illustrates the evolution of $\theta(u^k)$ for each method on this instance.

The results are less homogeneous on cutting stock than bin packing: the iterations needed by VOLUME spread on a substantially broader range, from

Table 13 Bin packing and cutting stock: volume versus bundle

Problem	Best value	Obtained at iteration		Optimum	
	by volume	VOLUME	BUNDLE0	Value	Iter
BP-t501.14	166.3	1,991	522	167	547
BP-t249.01	82.9	1,366	349	83	355
CS-WG50-75-10.03	240.8	2,809	94	241	98
CS-WG50-75-50.11	871.3	1,488	126	916.4	196
CS-DPM100-15-25.05	991.4	1,906	175	993	247
CS-DPM100-05-00.00	3,045	2,615	204	3,055	216

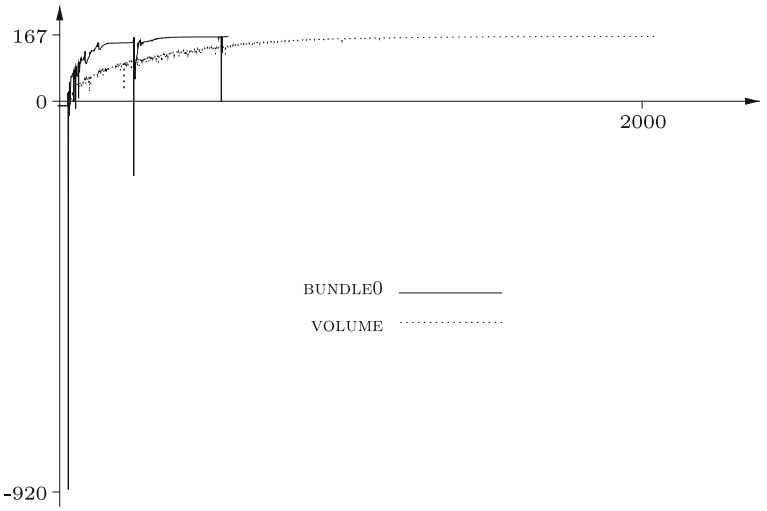

Fig. 12 Bin packing: volume versus bundle on BP-t501-14

1,500 to 3,000; the dual maximal value is often approximated rather closely, but sometimes more grossly: 5% on instance 11 of WG50-75-50; the corresponding evolution of both algorithms is illustrated by Fig. 13, which we find fairly aesthetic. Table 13 reports on instance 00 of DPM100-05-00 because it is rather special:

- the dual maximum appears to be very sharp, (see below point (iii) in our conclusion section and the right part of Fig. 16), an unfavourable case for the volume algorithm;
- the available upper bound N on the number of stock pieces is actually the optimal primal value;
- there is no duality gap.

Remark 3.1 The oracle is Martello–Toth's knapsack solver; but in order to mitigate the erratic behaviour of an NP oracle, we have inserted a relaxed stopping criterion, as in [30]: a knapsack is declared optimal and returned to the master as soon as its optimality within 10^{-8} is proved.

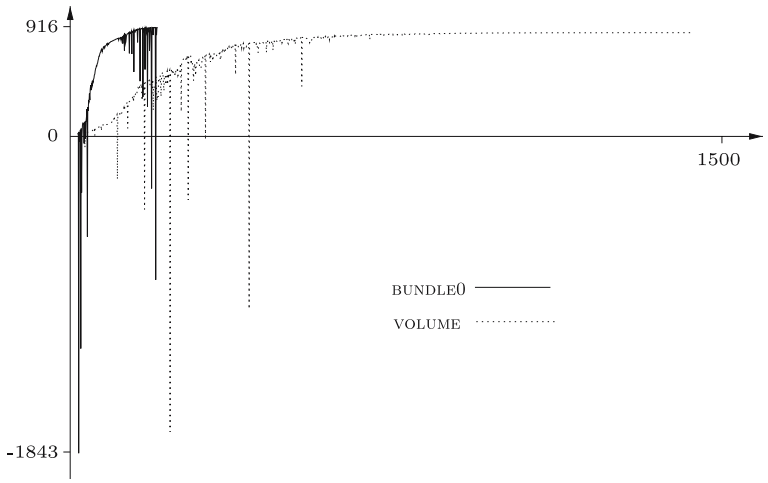

Fig. 13 CSP: volume versus bundle on WG50-75-50.11

Table 14 TSP: volume versus bundle

Problem	Best value	Obtained at iteration		Optimum (bundle)	
	by volume	VOLUME	BUNDLE0	Value	Iter
bays29	9,014.3	66	19	9,015	52
pr76	105,113	135	103	105,120	112
pcb442	50,495	502	191	50,499	357
pcb1173	56,348	456	209	56,351	527
pcb3038	136,557	502	316	136,587	4,957

The result can be spectacular. On the industrial instance with 30 items from Table 1, the CPU time is divided by 100, while the iteration counter goes from 98 to 100. On a 50-item instance of Gau-Wäscher, the algorithm collapses after 110 iterations, simply because the oracle runs forever.

Note that the results in the present section can by no means be compared to those of Sect. 2.2, the oracle and the computing environment being so different. □

3.2 TSP with volume

Our experiments in Sect. 2.6 were limited by the poor performances of Kelley. However volume can deal efficiently with bigger instances: in addition to those of Sect. 2.6, we present results for pcb1173 and pcb3038. The default values of the optimal parameters are used in VOLUME—we could not find a better set; it runs systematically during about 500 iterations in all the test-problems. Results are reported on Table 14, which reads just as Table 13.

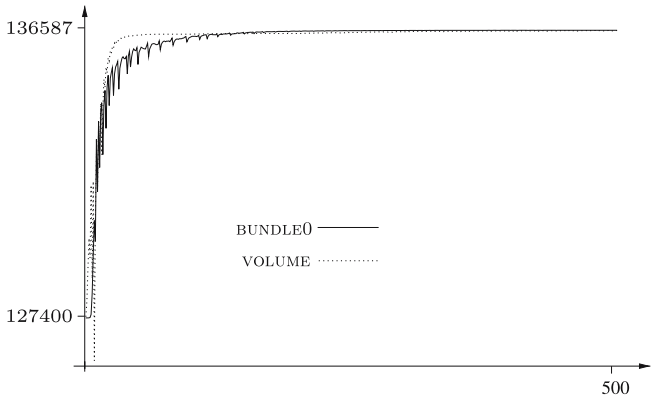

Fig. 14 TSP: volume versus bundle on pcb3038

If poor accuracy is accepted, both methods behave very similarly, as illustrated by Figs. 14 and 15, which show their respective evolutions along iterations. Note a rather special behaviour on Fig. 15: VOLUME reaches its best value faster than bundle; but then, the iterates deteriorate substantially and consistently until they stop (at iteration 502).

3.3 Comments

We have conducted the above experiments without any involvement of the volume team. The results obtained with the volume algorithm may therefore not be the best possible and should be taken with care. For example, the behaviour illustrated by Fig. 15 might perhaps be overcome by an appropriate action on VOLUME. Here we will just propose the following cautious comments:

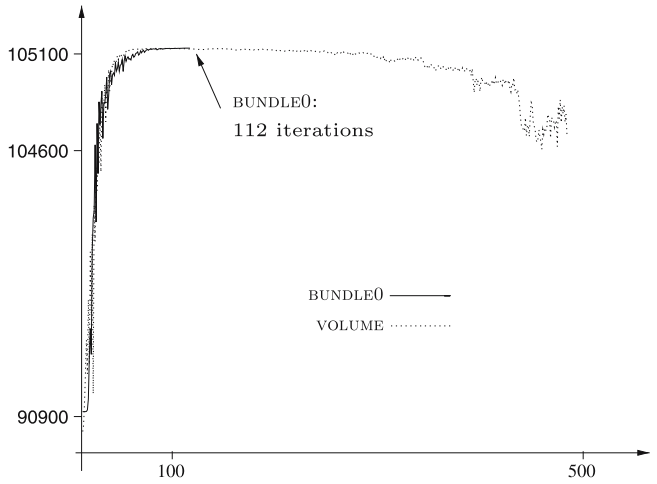

Fig. 15 TSP: volume versus bundle on pr76

– As a subgradient-type method, volume behaves fairly well.
– However it can hardly reach high accuracy—just as any other subgradient-type method.
– "Seen from far", bundle behaves similarly (at least on TSP),
– but combines the advantages of Kelley; in particular, it enjoys reliable stopping criteria,
– which however may be fairly expensive to reach: see pcb3038 in Table 14.

Conclusions

The first part of this paper has presented in a unified way various stabilizing schemes of column generation, including the bundle and ACCPM approaches. Stabilization is conveniently introduced in the dual space; deriving its primal counterpart is easy when an LP formulation exists. However, we have mentioned a possible dualization scheme allowing the same primal derivation, even when the stabilized master is a genuine nonlinear program.

This uniform and concise overview of the theory and algorithms, which uses well-known tools from convex analysis, is not new (it is also treated in [14] for instance). The innovative part of the paper is really Sect. 2. The numerical behaviour of the bundle method is compared with standard column generation on a few combinatorial problems, while previous comparative reports tend to concentrate on a concrete application with specific tuning (see for instance [11,53]). Studying a larger spectrum of applications emphasizes that no method is consistently better. With no big surprise, our study gives further evidence of what might have been the general perception of the comparative behavior of these algorithms amongst the experts. Hopefully this study shall make the non-expert more aware of these facts and of the alternatives to standard LP-based approaches.

Let us summarize the conclusions that can be drawn from our experiments (the comments below concern the comparison Kelley vs. bundle, see Sect. 3.3 for a few comments about volume).

(i) Sometimes Kelley and bundle are grossly comparable in terms of number of calls to the oracle. This is generally observed for problems of Sects. 2.2–2.5.
Remark 2.3 is worth recalling here: in the case of CSP and VCP, we use our bundle code to solve a constrained problem by exact penalty, which is known to be rather inefficient. Using a constrained bundle method such as [28,39]—or, better: [32]—would probably improve the results.
Another useful observation is the lack of robustness of numerical experiments in nonsmooth optimization. Because the $x = x(u)$ answered by the oracle is discontinuous, small changes (in the data, the initialization, the accuracy of the master, …) may result in substantial changes in the behaviour of the column generation process. To solve a given problem by a given method, it is common that *the same* software takes substantially more iterations (say 20%) when compiled on a different computer.

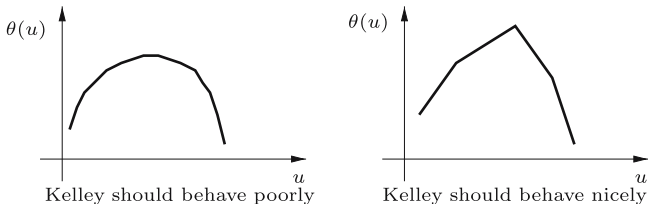

Fig. 16 The graph of the dual function

Keeping this in mind, the differences reported in Sects. 2.2–2.5 should not be overemphasized.

(ii) Sometimes, typically when m grows, Kelley simply collapses: see Sect. 2.6. This phenomenon was known before, the interesting point is that bundle still works for much larger instances: we refer to [37], in which instances with 10^3 vertices are solved to (dual) optimality; ACCPM is reported to behave similarly.

(iii) An interesting question related with (ii) is: when should Kelley behave poorly compared to bundle? According to Nemirovskii's example (see the end of Sect. 1.1), this should be more likely for large m. Common sense suggests that it should generally happen when θ^k of (1.13) approximates poorly the true dual function θ of (1.10); such is the case when the graph of θ presents many small facets. Figure 16 gives a rudimentary illustration of this point: both functions θ on the left and on the right are piecewise linear; nevertheless, approximating the one on the left by a piecewise linear model θ^k requires many sampling points; a quadratic model should probably be more efficient—and this is precisely what bundle proposes.
Anyway, it seems daring to propose a reliable answer to this question (assuming that a reliable answer exists at all!)

(iv) A phenomenon, already known and confirmed by our experiments, is that stabilization may affect the computing time spent in the oracle. The choice of the oracle solver (a branch-and-bound versus a dynamic program, for instance), its implementation and even the initialization of the master program, are determinant factors when analyzing oracle computing time over the course of the column generation algorithm. Nevertheless, even though the reported times should be taken cautiously, our NP oracles may be perceivably more difficult when called by a stabilized method. Paroxystic entries such as Queen 11x11 in Table 7 or E-m22-N4 in Table 10 can hardly be tolerated. We have not fully explained this behaviour, which may occur at any time during the column generation process: either by the end (CSP) or at the beginning (CVRP).
On the other hand, the dual iterates u^k of a stabilized algorithm are supposedly close together. It might therefore be advantageous to use "warm-started" oracles *à la* [56].

(v) Being specifically designed to maximize a concave function such as θ, the bundle method needs an *exact* oracle, to compute exact values of θ.

Keeping in mind (iv) above, this can become a killing disadvantage; by contrast, Kelley simply needs separating hyperplanes (which do not have to touch the graph of θ). One of the consequences is that, for problems with exponential oracle – especially VCP and CVRP—our experiments were limited to relatively small instances.
The cure lies in "robust" stabilized algorithms, accepting *inaccurate* oracles which do not fully minimize the Lagrangian. Concerning the bundle method, a promising step forward is made along these lines in [31]: preliminary experiments reported in [30] show that the benefit can be drastic.

(vi) Perhaps one of the most important messages delivered by these experiments is that, in terms of CPU time, the price to pay for solving the quadratic master (1.30) is quite reasonable with respect to the LP master (1.6). Here this price is negligible but the dual problems to solve are fairly small, and the oracles are expensive. Nevertheless, QP is also cheap for TSP, even large ones—see [37].
Besides, it should be observed that little R&D work has been devoted so far to QP, compared to the 50 years of intense activity in LP technology. The relative price of QP can therefore be expected to decrease in the future.

(vii) It can also be mentioned that an appropriate initialization of Krich can require fairly delicate heuristic considerations. The paper provides such dual heuristics. They do yield significant reduction in the number of iterations, compared with poorer initialization on which we did not report. On the other hand, they can be sensitive to the characteristics of the datasets; VCP is an illustration. By contrast, bundle appears as a push-button method, in which even the initial $\hat{u}$ has little importance.

(viii) As mentioned at the outset of Sect. 2, we should ideally have compared bundle with the most advanced LP stabilizations of Kelley. However a fair comparison between methods should imply the involvement of the methods' developers (judging from the difficulties we had to merge [59] with [41], this is hard work). Besides, a fair amount of application-specific tuning would be needed. We believe that our comparison between Kbasic and Krich already gives a flavour of what LP stabilizations of Kelley can bring. Similar comments can be made concerning volume.

Acknowledgment We are indebted to anonymous referees for their careful reading and for their suggestion to insert comparisons with the volume algorithm. For this, K.C. Kiwiel gracefully made available to us his oracle and test-problems.

References

1. Anstreicher, K., Wolsey, L.A.: On Dual Solutions in Subgradient Optimization. Unpublished manuscript, CORE, Louvain-la-Neuve (1993)
2. Augerat, P.: VRP problem instances (1995) http://www.branchandcut.org/VRP/data/
3. Barahona, F., Anbil, R.: The volume algorithm: Producing primal solutions with a subgradient method. Math. Program. **87**(3), 385–399 (2000)

4. Beasley, J.E.: Or-library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990) http://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html
5. Ben-Amor, H., Desrosiers, J.: A proximal-like algorithm for column generation stabilization. Technical report G-2003-43, Les Cahiers du Gerad, Montréal (2003)
6. Benameur, W., Neto, J.: Acceleration of cutting plane and column generation algorithms: application to network design (to appear, 2006)
7. Cheney, E., Goldstein, A.: Newton's method for convex programming and Tchebycheff approximations. Numer. Math. **1**, 253–268 (1959)
8. COmputational INfrastructure for Operations Research: http://www.coin-or.org
9. Dash Optimization: Xpress-MP: User guide and reference manual, release 12 (2001) http://www.dashoptimization.com
10. Degraeve, Z., Peeters, M.: Optimal integer solutions to industrial cutting stock problems: part 2: benchmark results. INFORMS J. Comput. **15**(1–3), 58–81 (2003)
11. du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P.: Stabilized column generation. Disc. Math. **194**(1–3), 229–237 (1999)
12. Ermol'ev, Y.M.: Methods of solution of nonlinear extremal problems. Kibernetica **2**(4), 1–17 (1966)
13. Feillet, D., Dejax, P., Gendreau, M., Gueguen, C.: An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks **44**(3), 216–229 (2004)
14. Frangioni, A.: Generalized bundle methods. SIAM J. Optim. **13**(1), 117–156 (2003)
15. Gau, T., Wäscher, G.: CUTGEN1: a problem generator for the standard one-dimensional cutting stock problem. Eur. J. Oper. Res. **84**, 572–579 (1995)
16. Geoffrion, A.M.: Lagrangean relaxation for integer programming. Math. Program. Study **2**, 82–114 (1974)
17. Goffin, J.-L., Haurie, A., Vial, J.-Ph.: Decomposition and nondifferentiable optimization with the projective algorithm. Manage. Sci. **38**(2), 284–302 (1992)
18. Goffin, J.-L., Luo, Z.-Q., Ye, Y.: Complexity analysis of an interior cutting plane for convex feasibility problems. SIAM J. Optim. **6**(3), 638–652 (1996)
19. Goffin, J.-L., Vial, J.-Ph.: Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method. Optim. Methods Softw. **17**(5), 805–867 (2002)
20. Held, M., Karp, R.: The traveling salesman problem and minimum spanning trees. Oper. Res. **18**, 1138–1162 (1970)
21. Held, M., Karp, R.: The traveling salesman problem and minimum spanning trees: part II. Math. Program. **1**(1), 6–25 (1971)
22. Hiriart-Urruty, J.-B., Lemaréchal, C.: Convex Analysis and Minimization Algorithms. Springer, Berlin Heidelberg New York (1993, Two volumes)
23. Hiriart-Urruty, J.-B., Lemaréchal, C.: Fundamentals of Convex Analysis. Springer, Berlin Heidelberg New York (2001)
24. Kelley, J.E.: The cutting plane method for solving convex programs. J. Soc. Indust. Appl. Math. **8**, 703–712 (1960)
25. Kim, S., Chang, K.N., Lee, J.Y.: A descent method with linear programming subproblems for nondifferentiable convex optimization. Math. Program. **71**(1), 17–28 (1995)
26. Kiwiel, K.C.: A dual method for certain positive semidefinite quadratic programming problems. SIAM J. Sci. Stat. Comput. **10**(1), 175–186 (1989)
27. Kiwiel, K.C.: An aggregate subgradient method for nonsmooth convex minimization. Math. Program. **27**, 320–341 (1983)
28. Kiwiel, K.C.: Methods of Descent for Nondifferentiable Optimization. Lecture Notes in Mathematics 1133. Springer, Berlin Heidelberg New York (1985)
29. Kiwiel, K.C.: A Cholesky dual method for proximal piecewise linear programming. Numer. Math. **68**, 325–340 (1994)
30. Kiwiel, K.C.: An inexact bundle approach to cutting stock problems. Technical report, Systems Research Institute, Warsaw. Submitted to INFORMS J. Comput. (2004)
31. Kiwiel, K.C.: A proximal bundle method with approximate subgradient linearizations. SIAM J. Optim. **16**(4), 1007–1023 (2006)
32. Kiwiel, K.C., Lemaréchal, C.: An inexact conic bundle variant suited to column generation (in preparation)

33. Sagastizábal, C., Bahiense, L., Maculan, N.: The volume algorithm revisited: relation with bundle methods. Math. Program. **94**(1), 41–69 (2002)
34. Larsson, T., Patriksson, M., Strömberg, A.B.: Ergodic, primal convergence in dual subgradient schemes for convex programming. Math. Program. **86**(2), 283–312 (1999)
35. Lemaréchal, C.: An algorithm for minimizing convex functions. In: Rosenfeld, J.L. (ed.) Information Processing '74, pp. 552–556. North Holland, Amsterdam (1974)
36. Lemaréchal, C.: Nonsmooth optimization and descent methods. Research Report 78-4, IIASA (1978)
37. Lemaréchal, C.: Lagrangian relaxation. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization, pp. 112–156. Springer, Berlin Heidelberg New York (2001)
38. Lemaréchal, C.: The omnipresence of Lagrange. 4OR **1**(1), 7,25 (2003)
39. Lemaréchal, C., Nemirovskii, A.S., Nesterov, Yu.E.: New variants of bundle methods. Math. Program. **69**, 111–148 (1995)
40. Lemaréchal, C., Pellegrino, F., Renaud, A., Sagastizábal, C.: Bundle methods applied to the unit-commitment problem. In: Dolež al, J., Fidler, J. (eds.) System Modelling and Optimization, pp. 395–402. Chapman and Hall, London (1996)
41. Lemaréchal, C., Sagastizábal, C.: Variable metric bundle methods: from conceptual to implementable forms. Math. Program. **76**(3), 393–410 (1997)
42. Marsten, R.E., Hogan, W.W., Blankenship, J.W.: The boxstep method for large-scale optimization. Oper. Res. **23**(3), 389–405 (1975)
43. Mehrotra, A., Trick, M.A.: A column generation approach to graph coloring. INFORMS J. Comput. **8**(4), 344–354 (1996)
44. du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P.: Stabilized column generation. Discrete Math. **194**, 229–237 (1999)
45. Nemirovskii, A.S., Yudin, D.: Informational complexity and efficient methods for the solution of convex extremal problems. Ékonomika i Mathematicheskie Metody **12**, 357–369 (1976) (in Russian. English translation: Matekon **13**, 3–25)
46. Nemirovskii, A.S., Yudin, D.: Problem Complexity and Method Efficiency in Optimization. Wiley-Interscience Series in Discrete Mathematics (1983). (Original Russian: Nauka, 1979)
47. Nesterov, Yu.E.: Complexity estimates of some cutting plane methods based on the analytic barrier. Math. Program. **69**(1), 149–176 (1995)
48. Nesterov, Yu.E., Vial, J.-Ph.: Homogeneous analytic center cutting plane methods for convex problems and variational inequalities. SIAM J. Optim. **9**(3), 707–728 (1999)
49. Polyak, B.T.: A general method for solving extremum problems. Soviet Math. Doklady **8**, 593–597 (1967)
50. Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Technol. J. **36**, 1389–1401 (1957)
51. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (1970)
52. Rockafellar, R.T.: A dual approach to solving nonlinear programming problems by constrained optimization. Math. Program. **5**, 354–373 (1973)
53. Rousseau, L.-M., Gendreau, M., Feillet, D.: Interior point stabilization for column generation. Working paper 39, Centre de Recherche sur les Transports, Univ. Montréal (2003)
54. Shor, N.: Utilization of the operation of space dilatation in the minimization of convex functions. Cybernetics **6**(1), 7–15 (1970)
55. Shor, N.Z.: Minimization methods for non-differentiable functions. Springer, Berlin Heidelberg New York (1985)
56. Thiongane, B., Nagih, A., Plateau, G.: Adapted step size in a 0-1 biknapsack lagrangean dual solving algoritm. Ann. Oper. Res. **139**(1), 353–373 (2004)
57. Uzawa, H.: Iterative methods for concave programming. In: Arrow, K., Hurwicz, L., Uzawa, H. (eds.) Studies in Linear and Nonlinear Programming, pp. 154–165. Stanford University Press, Stanford (1959)
58. Vanderbeck, F.: Extending Dantzig's bound to the bounded multi-class binary knapsack problem. Math. Program. **94**(1), 125–16 (2002)
59. Vanderbeck, F.: Dantzig-Wolfe re-formulation or how to exploit simultaneaously original formulation and column generation re-formulation. Working paper U-03.24, University Bordeaux 1, Talence (2003)