Math. Program., Ser. A 100: 267–294 (2004)

Digital Object Identifier (DOI) 10.1007/s10107-003-0469-4

Samir Elhedhli · Jean-Louis Goffin

The integration of an interior-point cutting plane method within a branch-and-price algorithm

Received: March 19, 2001 / Accepted: July 1, 2003
Published online: September 30, 2003 –

Abstract. This paper presents a novel integration of interior point cutting plane methods within branch-and-price algorithms. Unlike the classical method, columns are generated at a "central" dual solution by applying the analytic centre cutting plane method (ACCPM) on the dual of the full master problem. First, we introduce some modifications to ACCPM. We propose a new procedure to recover primal feasibility after adding cuts and use, for the first time, a dual Newton's method to calculate the new analytic centre after branching. Second, we discuss the integration of ACCPM within the branch-and-price algorithm. We detail the use of ACCPM as the search goes deep in the branch and bound tree, making full utilization of past information as a warm start. We exploit dual information from ACCPM to generate incumbent feasible solutions and to guide branching. Finally, the overall approach is implemented and tested for the bin-packing problem and the capacitated facility location problem with single sourcing. We compare against Cplex-MIP 7.5 as well as a classical branch-and-price algorithm.

Key words. branch-and-price – Column generation – Lagrangean relaxation – interior-point methods – ACCPM

1. Introduction

The technique of using column generation within a branch-and-bound framework is commonly called branch-and-price [1] or integer programming (IP) column generation [38]. The approach was initiated by the pioneering work of Gilmore and Gomory on the cutting stock problem [12], [13]; and prospered in the context of routing and scheduling by Desrochers et al. [3], [4], [5]. Recently, there has been considerable interest in this solution technique. Barnhart et. al. [1] give an overview of the approach describing the different models and branching rules. Vanderbeck and Wolsey [38] develop a new branching rule that generalizes existing ones and that is easily handled in the branch-and-price framework.

By duality, branch-and-price is analogous to a Lagrangean-based branch-and-bound where the Lagrangean dual problem is solved using a cutting plane method. Formulating the Lagrangean dual problem as a linear program yields the dual of the full master problem that is solved at each node of the branch-and-price algorithm. Column generation solves the primal full master problem, starting with a restricted version and adding

S. Elhedhli: Department of Management Sciences, University of Waterloo, 200 University Ave. W. Waterloo, ON, Ca N2L 3G1. e-mail: elhedhli@uwaterloo.ca

J.-L. Goffin: GERAD/Faculty of Management, McGill university, 1001 Sherbrooke W. Montreal, Qc, Ca H3A 1G5. e-mail: jean-louis.goffin@mcgill.ca

Mathematics Subject Classification (1991): 20E28, 20G40, 20C20

columns as needed; while cutting plane methods solve the dual full master problem, starting with a relaxed version and appending constraints as necessary. In the sequel we will not make a distinction between the primal and the dual full master problems, but we will refer to the restricted master problem in the context of column generation and relaxed master problem in the context of cutting plane methods. In this paper, we expose the material from a Lagrangean relaxation perspective.

In a Lagrangean based branch-and-bound, the predominant task is the solution of the Lagrangean dual problem, which is nondifferentiable. Most of the literature uses subgradient optimization. Although simple to implement, subgradient methods are slow to converge and have no clear stopping criteria [1]. Alternatively, the Lagrangean dual can be formulated as a linear program with a large number of constraints and solved using a cutting plane method. The approach starts with a subset of the constraints and appends new ones as needed. The added constraints are chosen based on a query point from the relaxed master problem. The choice of the query point distinguishes different variants of cutting plane methods, equivalently, different variants of column generation schemes. Classical branch-and-price methods use a dual extreme point of the restricted master problem as a query point. By duality, this corresponds to Kelley's cutting plane method [21] where cuts are generated at an extreme point of the relaxed master problem. It is known that Kelley's method suffers from tailing effects and that generating cuts at a centre of the relaxed master problem's feasible region is superior [30]. The main difficulty with central point strategies resides in the calculation of centres of convex sets. Calculating the centre of gravity, for example, is more difficult than optimizing the original problem. The Analytic Centre Cutting Plane Method (ACCPM) [10] is designed to overcome this difficulty. Cuts are generated based on the "analytic centre" concept from the interior-point literature. The calculation of the analytic centre is comparable to solving a linear program using an interior point method.

The first full implementation of ACCPM is provided in [6]. Since then, ACCPM has shown promising results in practice on a variety of problems. See [11] for a survey and [14] and [31] for the convergence analysis.

In this paper, we discuss the use of ACCPM within a branch-and-price framework and describe in detail the different parts of this novel integration.

1.1. Contributions

Recent papers discussing branch-and-price methodologies have focused on the design of a branching rule that is compatible with the branch-and-bound scheme [1], [37], [38], [40]. This paper explores another venue in the efficient design of a branch-and-price algorithm: the generation of columns. In classical branch-and-price methods, columns are generated using a dual extreme point of the restricted master problem. We propose to generate columns based on a dual central point. More precisely, the dual full master problem is solved using ACCPM where the subproblems are called at the analytic centre of a bounded subset of the dual feasible region.

From another perspective, the paper is a serious step in the efficient use of interior point methods within branch-and-bound approaches for integer programming. Previous attempts, mainly those by Mitchell [26] [27], have focused on the solution of the linear

programs using an interior-point method. Our approach is fundamentally different in two ways. On the one hand, we use a Lagrangean bound rather than the linear-programming (LP) bound. On the other hand, the interior point method is used in a cutting plane context rather than as a direct solution method.

It is stated in [27] that the success of the simplex method in branch-and-bound settings is mainly due to the warm start strategy in solving the linear programs. At a child node, the dual simplex method is started using the final tableau at the parent node. Exploiting this same warm start strategy when using an interior-point method is not possible. A recent paper by Gondzio [17] focuses on the use of a warm start strategy when using an infeasible primal-dual method to solve the relaxed master problems. Before reaching optimality of one problem, he saves a point that will be used as a warm start for a subsequent problem. In this paper, we show how information generated at parent nodes is used as a warm start in child nodes both within the branch-and-price framework and within ACCPM. The computational experience clearly indicate the effectiveness of this warm start strategy.

The contributions of this paper concern both ACCPM and the interior point branch-and-price algorithm. First, we improve on ACCPM in three ways. We use a weighted log barrier potential function instead of the Karmarkar potential function [20] in the calculation of the analytic centre. In addition, we provide a new procedure for recovering primal feasibility after adding cuts. When cuts are added, we extend the central-cut procedure of [15] to the deep-cut case in order to generate a primal feasible point.

Second, we detail the different components of the interior point branch-and-price method. We show how to use ACCPM as the search goes deep in the branch-and-bound tree making use of past information as a warm start. We use, for the first time, a dual Newton's method within ACCPM to calculate the first analytic centre after branching and exploit dual information from ACCPM to generate incumbent solutions and to guide branching.

1.2. Outline

The remainder of the paper is organized as follows. In section 2, we use the capacitated facility location problem with single sourcing (CFLSS) and the bin packing problem (BP) to detail the branch-and-price algorithm. In Section 3, we discuss the solution of the Lagrangean dual problems using ACCPM. In particular, the calculation of analytic centres and the recovery of primal feasibility after adding cuts. In section 4, we describe the branching rule and the calculation of the analytic centre after branching. Issues related to fathoming and generating incumbent feasible solutions are addressed in section 5. In section 6, we present the computational results. Finally, we conclude and provide venues for future research in section 7.

2. Branch-and-price algorithm

In this section we describe the branch-and-price algorithm. We use two related problems for demonstration: the capacitated facility location problem with single sourcing and

the bin-packing problem. The problems lead to two different formulations of the master problems, with non-identical and identical subproblems respectively. We begin by discussing each problem, applying Lagrangean relaxation and formulating the Lagrangean dual problems as linear programs with large number of constraints (master problems).

The branch-and-price algorithm proceeds like a branch-and-bound algorithm that uses the solution of the master problems as a lower bound. The algorithm is sketched in Figure 1 and its major components are detailed in the following sections.

It is worth noting that the solution of the master problem in step 2 can be done using any cutting plane method, including ACCPM, Kelley's classical method [21] and Bundle methods [23].

2.1. The master problems

2.1.1. CFLSS: Non-identical subproblems. The capacitated facility location problem with single-sourcing is a special case of the capacitated facility location problem where each customer is serviced by a single facility. The problem has applications in telecommunication and distribution networks design. It was treated in [22], [29], [32] and recently in [18]. The formulation is given by

$$[CFLSS]: \quad \min \quad \sum_{k=1}^{K} \sum_{l=1}^{L} c_{kl} y_{kl} + \sum_{k=1}^{K} f_k z_k \tag{1}$$

$$s.t. \quad \sum_{k=1}^{K} y_{kl} = 1 \qquad l = 1, .., L \tag{2}$$

Initilaization: Intial upper bound: $UB = \infty$. Initial set of nodes to explore: $S = \{1\}$. Initial lower bound for node 1 : $LB_1^0 = -\infty$. Initial matrix for node 1: A_1^0 is empty.
Iteration: While there are nodes to explore (S is not empty)

1. Pick a node, say node n
2. Get a lower bound LB_n from the full master problem.
   - Use the matrix A_n^0 as a starting matrix.
   - Use the lower bound LB_n^0 as an intial lower bound
   - Get a lower bound LB_n by applying a cutting plane/column generation method (ACCPM, Kelley's method [21] or Bundle methods [23]).
3. If possible, generate a feasible solution. This gives an upper bound UB_n (section 2.3).
4. Update upper bound: $UB = \min(UB_n, UB)$
5. If $LB_n \geq UB$
   - Fathom node n: $S = S \backslash \{n\}$.
6. Else
   - Branch: Create two new nodes n_1 and n_2 : $S = S \cup \{n_1, n_2\}$ (section 2.2).
7. Save warm starting information:
   - Use the branching rule to split the columns of A_n into two matrices $A_{n_1}^0$ and $A_{n_2}^0$. Use them as initial matrices for the child nodes n_1 and n_2.
   - The lower bound LB_n is valid for the child nodes. Use it to initialize the lower bounds at child nodes n_1 and n_2. $LB_{n_1}^0 = LB_n$ and $LB_{n_2}^0 = LB_n$.

End while

Fig. 1. The main steps of the branch-and-price algorithm.

$$\sum_{l=1}^{L} D_l y_{kl} \leq V_k z_k \qquad k = 1, ..., K \tag{3}$$

$$y_{kl}, z_k \in \{0, 1\} \qquad k = 1, ..., K; \quad l = 1, ..., L \tag{4}$$

where the set of potential facilities and the set of customers are indexed by k and l respectively. The facilities have capacity V_k and fixed cost f_k. There is a variable cost c_{kl} for assigning customer demand D_l to facility k. The binary variables z_k take value 1 when facility k is opened, while y_{kl} take value 1 when customer l is assigned to facility k. Constraints (2) are the single-sourcing constraints, while constraints (3) are capacity constraints that force facilities to be opened before servicing any customer.

There are a number of papers that discuss the exact solution of $[CFLSS]$. An LP-based branch-and-bound is used in [29]. Lagrangean-based heuristics and branch-and-bound methods are proposed in [32] and [18]. Most of the Lagrangean methods relax the single-sourcing constraints, which provides the sharpest Lagrangean bound. The exact methods considered in [32] and [18] do not qualify as branch-and-price methods since the master problems are not solved using a column generation approach. In [32], subgradient optimization is used once at the root node and the multipliers determined there are used without reoptimization throughout the branch-and-bound tree. In [18] , subgradient optimization is applied at every node, with the best dual multipliers at the parent node being used to initialize the method at child nodes.

Relaxing constraints (3) in a Lagrangean fashion leads to a trivial problem that has the integrality property. Thus it leads to the same bound as the LP-bound [9]. We choose to relax constraint (2) using dual multipliers $\lambda_l, l = 1, ..., L$. This leads to the subproblem

$$[KP_\lambda]: \min_{y,z} \sum_{k=1}^{K} \sum_{l=1}^{L} (c_{kl} - \lambda_l) y_{kl} + \sum_{k=1}^{K} f_k z_k \tag{5}$$

$$\sum_{l=1}^{L} D_l y_{kl} \leq V_k z_k \qquad k = 1, ..., K \tag{6}$$

$$y_{kl}, z_k \in \{0, 1\} \qquad k = 1, ..., K;\ l = 1, ..., L. \tag{7}$$

Subproblem $[KP_\lambda]$ decomposes into K independent problems that are easily solvable as 0-1 knapsack problems. In the sequel, we use $v(\bullet)$ to denote the optimal objective of problem $(\bullet)$. Therefore, $\sum_{l=1}^{L} \lambda_l + v(KP_\lambda) \leq v(CFLSS)$ for all λ. The best lower bound is given by the solution of the Lagrangean dual problem

$$\max_{\lambda} \left\{ \sum_{l=1}^{L} \lambda_l + v(KP_\lambda) \right\}. \tag{8}$$

Problem (8) is a nondifferentiable optimization problem that can be re-formulated as a linear program with a large number of constraints

$$\begin{aligned} &\max\ \theta + \sum_{l=1}^{L} \lambda_l \\ &s.t.\ \sum_{k=1}^{K} \sum_{l=1}^{L} (c_{kl} - \lambda_l) y_{kl}^h + \sum_{k=1}^{K} f_k z_k^h \geq \theta; \quad h = 1, ..., H \end{aligned} \tag{9}$$

where $(y^h, z^h), h = 1, ..., H$, denotes an enumeration of the integer solutions in the bounded set

$$\left\{(y, z) : \sum_{l=1}^{L} D_l y_{kl} \leq V_k z_k, \ \forall k\right\}$$

where $y = (y_{kl})_{k=1,...,K;\, l=1,...,L}$ and $z = (z_{kl})_{k=1,...,K}$.

Problem (9) is commonly called the *full master problem*. This formulation refers to the aggregated case where the possibility of decomposing $[K P_\lambda]$ into K subproblems is not exploited. In this paper, we use the disaggregated formulation where $[K P_\lambda]$ is split into a set of K independent subproblems. Disaggregation is preferred to aggregation since it allows the faster accumulation of cuts and accelerates the solution of the master problem. The disaggregated full master problem is

$$\begin{aligned}[FMP_{CFLSS}]: \quad & \max \sum_{k=1}^{K} \theta_k + \sum_{l=1}^{L} \lambda_l \\ & s.t. \sum_{l=1}^{L} (c_{kl} - \lambda_l) y_{kl}^h + f_k z_k^h \geq \theta_k; \quad h = 1, ..., H_k, \ k = 1, ..., K \end{aligned} \tag{10}$$

where, for each k, $((y_{kl}^h)_{l=1,...,L}, z_k^h), h = 1, ..., H_k$, denotes an enumeration of the integer solutions to the k-th bounded region:

$$\left\{(y_{kl}, z_k) : \sum_{l=1}^{L} D_l y_{kl} \leq V_k z_k\right\}.$$

If we take the dual of $[FMP_{CFLSS}]$, we get the Dantzig-Wolfe master problem

$$\begin{aligned} \min & \sum_{k=1}^{K} \sum_{h=1}^{H_k} \left(\sum_{l=1}^{L} c_{kl} y_{kl}^h + f_k z_k^h \right) \delta_{kh} \\ s.t. & \sum_{h=1}^{H_k} \delta_{kh} = 1 && k = 1, ..., K \\ & \sum_{k=1}^{K} \sum_{h=1}^{H_k} y_{kl}^h \delta_{kh} = 1 && l = 1, .., L \\ & \delta_{kh} \geq 0 && h = 1, ..., H_k, \ k = 1, ..., K. \end{aligned} \tag{11}$$

This establishes the primal-dual relationship between the Lagrangean dual and the Dantzig-Wolfe reformulation, subsequently, between branch-and-price and Lagrangean-based branch-and-bound where the master problems are solved using a cutting plane method. In the following section, we apply the same analysis to the bin-packing problem.

2.1.2. BP: Identical subproblems. In the bin-packing problem (BP) we seek to find the minimum number of bins of size V that can handle a set of L items, of varying sizes D_l, $l = 1, ..., L$. Problem BP is a special case of CFLSS where $c_{kl} = 0$ for $k = 1, ..., K$ and $l = 1, ..., L$, $f_k = 1$ and $V_k = V$ for all $k = 1, ...K$. The complete formulation is

$$[BP]: \quad \min \sum_{k=1}^{K} z_k \tag{12}$$

$$s.t. \quad \sum_{k=1}^{K} y_{kl} = 1 \qquad l = 1, .., L \tag{13}$$

$$\sum_{l=1}^{L} D_l y_{kl} \leq V z_k \qquad k = 1, ..., K \tag{14}$$

$$y_{kl}, z_k \in \{0, 1\} \qquad k = 1, ..., K; \quad l = 1, ..., L \tag{15}$$

where the set of bins and the set of items are indexed by k and l respectively. The binary variable z_k takes value 1 when bin k is used while y_{kl} takes value 1 when item l is assigned to bin k. Constraints (13) assign each item to exactly one bin. Constraints (14) are capacity constraints that force bins to be used before containing any item. The objective (12) minimizes the total number of bins used.

The bin-packing problem is a classical NP-hard problem for which different exact solution methods were proposed. Martello and Toth [24] summarize previous work and provide a branch-and-bound algorithm based on a combinatorial lower bound. Scholl et. al. [34] use a hybrid method that combines tabu search and branch-and-bound. Vance et al. [37] use a branch-and-price algorithm based on the Ryan-and-Foster branching rule [33]. Valerio de Carvalho [36] applies a branch-and-price algorithm on an arc-flow formulation of the bin-packing problem. Finally Vanderbeck [39] proposes a branch-and-price algorithm based on the branching scheme in [38] and enhances the algorithm using valid cuts, variable fixing and a rounding heuristic.

Since $[BP]$ is a special case of $[CFLSS]$, its Lagrangean relaxation leads to aggregated and disaggregated full master problems as in (9) and (10), respectively. Exploiting the fact that the subproblems are identical, the full disaggregated master problem becomes

$$[FMP_{BP}] \quad \max K\theta + \sum_{l=1}^{L} \lambda_l$$

$$s.t. \sum_{l=1}^{L} (-\lambda_l) y_l^h + z^h \geq \theta, \quad h = 1, ..., H$$

where H is the index set of the integer solutions to the K identical feasible regions

$$\left\{ (y_l, z) : \sum_{l=1}^{L} D_l y_l \leq V z; \;\; y_l, z = 0, 1; \qquad l = 1, .., L \right\}.$$

The K subproblems are identical 0-1 knapsack problems of the form

$$\begin{aligned} \min_{y,z} \quad & \sum_{l=1}^{L} (-\lambda_l) y_l + z \\ & \sum_{l=1}^{L} D_l y_l \leq V z \\ & y_l, z \in \{0,1\} \qquad\qquad l = 1, ..., L. \end{aligned}$$

The dual of $[FMP_{BP}]$ is the Dantzig-Wolfe master problem

$$\begin{aligned} \min \ & \sum_{h=1}^{H} \left(z^h\right) \delta_h \\ s.t. \ & \sum_{h=1}^{H} \delta_h = K \\ & \sum_{h=1}^{H} y_l^h \delta_h = 1 \qquad l = 1, ..., L \\ & \delta_h \geq 0 \qquad\qquad h = 1, ..., H. \end{aligned} \tag{16}$$

Note that in (11) there are K convexity constraints whereas in (16) there is a single convexity constraint. In addition, K subproblems are solved and K cuts are added for CFLSS, while a single subproblem is solved and a single cut is added for BP.

At each node of the branch-and-price algorithm, the Lagrangean dual problem is solved using ACCPM. A node is fathomed if the solution of the Lagrangean problem provides a feasible solution to the original problem, the node is infeasible or the lower bound is greater than the incumbent. The solution of the Lagrangean dual problem is feasible to the original problem if it satisfies the relaxed constraints (2). If the node is not fathomed, branching is performed.

2.2. Branching

The branching rule should be designed to integrate easily into the column generation scheme. The branching constraints are more efficient if they are appended to the subproblems so that infeasible columns are not generated in subsequent nodes. This implies that the subproblem changes from one node to an other. The challenge is to design a branching rule where the branching constraints are appended to the subproblems without distorting its structure. Barnhart et al. [1] give a branching rule for set-partitioning type of problems. The rule was originally proposed by Ryan-and-Foster [33] and is based on the idea of putting variables into a single subset versus putting them in different subsets. The rule is generalized in Vanderbeck [40], discussed in Barnhart et al. [1] and is successfully used in Vance et al. [37], du Merle et al. [7] and Vanderbeck [39].

In this paper, we also use a Ryan-and-Foster branching rule. Investigating the coefficient matrix in (11) or (16), we identify two rows ρ_1 and ρ_2 and two columns κ_1 and κ_2 having this pattern

	κ_1	$\cdots$	κ_2
ρ_1	1		1
$\vdots$			
ρ_2	1		0

For the bin-packing problem, there are always two rows ρ_1 and ρ_2 that correspond to two items l_1 and l_2. The branching constraints are

$$y_{kl_1} = y_{kl_2}, \qquad k = 1, ..., K \tag{17}$$

$$y_{kl_1} + y_{kl_2} \leq 1, \qquad k = 1, ..., K. \tag{18}$$

Note that (17) and (18) are not exclusive (they do not have to). If the inequality in (18) is changed to an equality then it would imply that every bin should contain either of items l_1 or l_2, violating the fact that every item is assigned to only one bin ($\sum_{k=1}^{K} y_{kl} = 1$). In the implementation, the columns are ordered according to the value of the dual multipliers and those with the highest multipliers are checked first. This is motivated by the fact that these columns are likely to be in an optimal solution. For the CFLSS, we look for two customer indices l_1 and l_2 that correspond to rows ρ_1 and ρ_2 and use the branching rule in (17-18). As there are K convexity constraints in (11), the first row ρ_1 may correspond to a convexity constraint $\bar{k}$, while ρ_2 corresponds to a customer index $\bar{l}$. In this case, we use the classical branching rule

$$y_{\bar{k}\,\bar{l}} = 0; \qquad y_{\bar{k}\,\bar{l}} = 1. \tag{19}$$

See [1] for more details. In the algorithm presented in this paper, priority is given to the Ryan-and-Foster branching rule (17-18). In case that is not possible, the classical rule (19) is used.

For BP (correspondingly CFLSS), the left branching constraint (17) forces l_1 and l_2 to be assigned to the same bin (served from the same facility), while the right branching constraint (18) forces l_1 and l_2 to be assigned to different bins (served from different facilities). Appending (17) is easily done by aggregating l_1 and l_2 into a single item (customer zone) $\bar{l}$ with demand $D_{\bar{l}} = D_{l_1} + D_{l_2}$ and cost $\frac{c_{k_1 l_1} D_{l_1} + c_{k_1 l_2} D_{l_2}}{D_{l_1} + D_{l_2}}$. Branching constraint (18) is more difficult to deal with. Adding it to the subproblems will distort its structure. According to Vance et al. [37], this difficulty at the subproblems pays back at the master problem, as set-partitioning problems with disjoint constraints are more likely to be integral. Furthermore, according to their computational experience, exploring the right branches was hardly done. In du Merle et al., there was no difficulty incorporating (18) into their quadratic 0-1 subproblem, as (18) is equivalent to $y_{k_1 l_1} . y_{k_1 l_2} = 0$, which is reinforced by setting the corresponding cost coefficient to a sufficiently large number. When using the classical branching rule, constraints (19) are easily handled when solving the knapsack problems. Item $\bar{l}$ is deleted from the list of items when $y_{\bar{k}\,\bar{l}} = 0$ while it is deleted and the capacity of facility $\bar{k}$ is changed to $V_{\bar{k}} - D_{\bar{k}\,\bar{l}}$ when $y_{\bar{k}\,\bar{l}} = 1$.

2.3. Generating feasible solutions

ACCPM has the advantage of providing a dual feasible solution at every iteration of the cutting plane method. These dual solutions are feasible to the Dantzig-Wolfe formulation in (11) (respectively (16)). By rounding them to 0 or to 1, a feasible solution can be constructed. The dual multipliers δ that are nearer to 1 are rounded up, those that are nearer to 0 are rounded down and the ones in between are rounded up or down in a greedy manner so as to satisfy the constraints of (11) (respectively (16)).

For the CFLSS, we additionally use a rounding heuristic on the original variables $y_{kl} = \sum_h y_{kl}^h \alpha_{kh}$. Generally, the resulting y does not satisfy both the single sourcing constraints (2) and the capacity constraints (3). If the rounding threshold is significantly high, say 0.9, it is likely that the capacity constraints are satisfied while the single sourcing constraints are satisfied as inequalities. The heuristic tries to satisfy the demand for the unassigned customers ($y_{kl} = 0$) without exceeding the warehouse capacities. First the opened facilities are investigated and if need arises those with the lowest fixed cost are opened first.

For the BP, we look at the generated columns and try to select a subset of them that satisfies (13). We put the columns in a sorted list and select them one by one until (13) is satisfied or the heuristic fails. We use two sorting strategies: increasing order of the dual multipliers and decreasing order of the bin waste, i.e. the slack in (14).

3. Solving the master problem: ACCPM

For ease of exposition, let us write the full master problem, whether $[FMP_{CFLSS}]$ or $[FMP_{BP}]$, as

$$[FMP] : \max\{b^T u : A^T u \leq c\}.$$

Cutting plane methods use a subset of constraints to form a relaxed master problem

$$[RMP] : \max\{b^T u : A_q^T u \leq c_q\}$$

where A_q is the matrix containing a subset of the rows of A and c_q is the vector containing the corresponding elements of c. The index q stands for a particular iteration of the cutting plane method.

At each iteration, the solution of the relaxed master problem $[RMP]$ is checked for optimality for the full master problem $[FMP]$. If it is not optimal, an additional set of constraints is appended and a new relaxed master problem is formed. The relaxation becomes tighter as more constraints are added. The generation of constraints is done by calling the oracle, i.e., solving the subproblems $[KP_\lambda]$ at a query point $\overline{u}^q$. In addition, the oracle provides a lower bound z_{lower}, that is updated as

$$z_{\text{lower}} = \max\left\{z_{\text{lower}}, \sum_{l=1}^{L} \lambda_l + v(KP_\lambda)\right\}.$$

Based on that, we form the localization set

$$\mathcal{F}_D^q(z_{\text{lower}}) = \left\{ u : b^T u \geq z_{\text{lower}}; \quad A_q^T u \leq c_q \right\}$$

which is a bounded subset of the feasible region that contains any optimal solution to $[FMP]$. To ensure its boundedness, the box constraints $\ell \leq u \leq \mu$ are added.

A cutting plane algorithm constructs a sequence of query points $\{\overline{u}^q\}$ that are used to generate cuts $B_q u \leq r_q$ and to update the constraint matrix as follows

$$A_{q+1} = \left[A_q, \ B_q \right]; \quad c_{q+1} = \begin{bmatrix} c_q \\ r_q \end{bmatrix}.$$

At each iteration, any dual feasible point of the relaxed master problem $\max\{b^T u : A_q^T u \leq c_q\}$, say $\underline{x}^q$, is dual feasible for the full master problem $[FMP]$, and so it provides an upper bound $z_{\text{upper}} = c_q^T \underline{x}^q$ to $v(FMP)$.

In Kelley's cutting plane method [21], cuts are generated at the optimal solution of $[RMP]$. Cuts generated at the centre of the localization set lead to faster methods [30]. Unfortunately, the calculation of the analytic centre of gravity is very hard. With this in mind, ACCPM generates cuts at a central point that is fairly easy to compute. In particular cuts are generated at the analytic centre of $\mathcal{F}_D^q(z_{\text{lower}})$. The analytic centre is the point that maximizes the product of the distances from the boundaries of the localization set:

$$u^a = \arg\max(b^T u - z_{\text{lower}}) \prod_j (c_q^j - a_q^{jT} u)$$

where c_q^j is the j^{th}component of c_q and a_q^j is the j^{th} column of A_q. If we take the Logarithm, u^a is the point that maximizes the dual potential function .

$$\varphi_D^q(u) = \log(b^T u - z_{\text{lower}}) + \sum_j \log(c_q^j - a_q^{jT} u).$$

To create a balance between the objective constraint $b^T u \geq z_{\text{ lower}}$ and the rest of the constraints, we use a weighted analytic centre that replicates the objective constraint m times, i.e. it is given a weight of m :

$$u^a = \arg\max(b^T u - z_{\text{lower}})^m \prod_j (c_q^j - a_q^{jT} u).$$

If we take the Logarithm, we get the weighted dual potential function

$$\varphi_D^q(u) = m \log(b^T u - z_{\text{lower}}) + \sum_j \log(c_q^j - a_q^{jT} u). \tag{20}$$

In this paper, we set m to the total number of cuts in the restricted master problem. The analytic centre $\overline{u}^q$ is used by the subproblems to generate new cuts and update the lower bound z_{lower}. The computation of analytic centers and the generation of cuts continues until $z_{\text{upper}} - z_{\text{lower}}$ falls below a desired tolerance level ε. The main steps of ACCPM are described in Figure 2.

The main step in ACCPM is the computation of the analytic centre, which is discussed in the following section

Initialization: Let $\mathcal{F}_D^0 = \{u : b^T u \geq z_{lower} : \ell \leq u \leq \mu\} = \{u : b^T u \geq z_{lower} : A_0^T u \leq c_0\}$ where $A_0 = [I, -I]$ and $c_0 = \begin{pmatrix} \mu \\ -\ell \end{pmatrix}$. z_{lower}^0 and z_{upper}^0 are initial lower and upper bounds. The stopping parameter is ε

Iteration: while $\left| z_{upper} - z_{lower} \right| > \varepsilon$

1. Compute the analytic center $\overline{u}_q$ of $\mathcal{F}_D^q(z_{lower})$.
2. The oracle returns the cuts $B_q^T u \leq r_q$ and a lower bound z_{lower}^q
3. Update the coefficient matrix $A_{q+1} = [A_q, B_q]$, $c_{q+1} = \begin{bmatrix} c_q \\ r_q \end{bmatrix}$
4. Update the lower bound $z_{lower} = \max(z_{lower}^q, z_{lower})$
5. Update the localization set $\mathcal{F}_D^{q+1}(z_{lower}) = \{b^T u \leq z_{lower}\ , A_{q+1}^T u \leq c_{q+1}\}$ and find an upper bound z_{upper}^q.
6. Update the upper bound $z_{upper} = \min(z_{upper}, z_{upper}^q)$

End while

Fig. 2. The main steps of ACCPM.

3.1. Calculating the analytic centre

The analytic centre concept was introduced by Sonnevend [35] and is well studied in the interior-point literature. Three different approaches are proposed to calculate it: the primal, the dual and the primal-dual. For clarity of exposition, let us drop the iteration index q.

The weighted analytic center of $\mathcal{F}_D(z_{lower})$ is the unique point maximizing the weighted dual potential function (20). More specifically, it is the point $\overline{u}$ that solves

$$\begin{aligned} \max \quad & \varphi_D(s) = m \log s_0 + \sum_{i=1}^{m} \log s_i \\ s.t. \quad & A^T u + s = c \\ & b^T u - s_0 = z_{\text{lower}} \\ & s_{0,} s > 0. \end{aligned} \tag{21}$$

The necessary and sufficient first order optimality conditions of (21) are

$$\begin{aligned} & Sx = e \\ & s_0 x_0 = m \\ & x_0 b - Ax = 0, \qquad && x, x_0 > 0 \\ & A^T u + s = c, && s > 0 \\ & b^T u - z_{\text{lower}} = s_{0,} && s_0 > 0, \end{aligned} \tag{22}$$

where e is the vector with appropriate dimension whose components are all ones. Conditions (22) are also the first order optimality conditions when maximizing the weighted primal log potential function

$$\begin{aligned} \max \quad & \varphi_P(x) = -c^T x + z_{lower} x_0 + m \log x_0 + \sum_{i=1}^{m} \log x_i \\ s.t. \quad & Ax = x_0 b, \quad x, x_0 > 0, \end{aligned} \tag{23}$$

and the primal-dual potential function

$$\begin{array}{lll} \max & \varphi_{PD}(x,s) = \varphi_P(x) + \varphi_D(s) & \\ s.t. & Ax = x_0 b, & x, x_0 > 0 \\ & A^T u + s = c, & s > 0 \\ & b^T u - s_0 = z_{\text{lower}}, & s_0 > 0. \end{array} \tag{24}$$

Defining N as the diagonal matrix of $(m, 1, ..., 1)$, $\tilde{x} = \begin{bmatrix} x_0 \\ x \end{bmatrix}$, $\tilde{c} = \begin{bmatrix} -z_{lower} \\ c \end{bmatrix}$, $\tilde{s} = \begin{bmatrix} s_0 \\ s \end{bmatrix}$ and $\tilde{A} = [-b, A]$, the conditions in (22) are written as

$$\tilde{S}\tilde{x} = Ne \equiv \nu \tag{25}$$

$$\tilde{A}\tilde{x} = 0, \quad \tilde{x} > 0 \tag{26}$$

$$\tilde{A}^T u + \tilde{s} = \tilde{c}, \quad \tilde{s} > 0 \tag{27}$$

In practice, an approximate analytic center is used. The quadratic centering condition in (25) is replaced by the relaxed condition

$$\left\| \tilde{S}\tilde{x} - Ne \right\| \leq \xi, \quad \xi < 1, \tag{28}$$

which is usually called a proximity measure [41]. Any point $(\tilde{x}, \tilde{s})$ satisfying conditions (28),(26),(27) is called a ξ-analytic center.

If only a primal feasible solution $\tilde{x} > 0$ is available, then a primal Newton's method can be used to compute a ξ-center. If only a dual feasible solution $\tilde{s} > 0$ is available, then a dual Newton's method can be used to compute a ξ-center. If both a primal and a dual feasible point $(\tilde{x} > 0, \tilde{s} > 0)$ is available then a primal-dual Newton's method can be used.

3.2. Computing the next analytic center after adding cuts

In ACCPM, the primal Newton's method is favored over the dual or the primal-dual ones. The reason is that the addition of cuts is likely to eliminate a big portion of the dual feasible region leading to the infeasibility of the current analytic centre. Although, it is difficult to recover dual feasibility, primal feasibility can be recovered easily. This recovery is discussed next.

3.2.1. Recovering a primal feasible point. At iteration q of ACCPM, the relaxed master problem is $\max\{b^T u : b^T u \geq \overline{z}_{\text{lower}}, A_q^T u \leq c_q\}$, for which the corresponding analytic centre is $\overline{u}^q$, its dual slack is $(\overline{s}_0^q, \overline{s}^q)$ and the corresponding primal analytic centre is $(\overline{x}_0^q, \overline{x}^q)$. Adding the cuts $B_q^T u \leq r_q$ and updating the lower bound to z_{lower}, the new relaxed master problem becomes $\max\{b^T u : b^T u \geq \overline{z}_{\text{lower}}, A_q^T u \leq c_q, B_q^T u \leq r_q\}$. Not only a new set of constraints $B_q^T u \leq r_q$ are added but also the objective cut $b^T u \geq z_{\text{lower}}$ is shifted whenever the lower bound z_{lower} is updated. Note that the old analytic centre $\overline{u}^q$ and its corresponding slack $\overline{s}^q$ are not necessarily feasible with respect to the new

cuts $B_q^T u \leq r_q$. For clarity of exposition, let us drop the iteration index q and use a compact notation where $A \leftarrow [-b, A]$, $c \leftarrow (-z_{\text{lower}}, c)$, $\bar{c} \leftarrow (-\bar{z}_{\text{lower}}, c)$, $\bar{x} \leftarrow (\bar{x}_0, \bar{x})$, $\bar{s} \leftarrow (\bar{s}_0, \bar{s})$ as in (25-27).

Based on the first order optimality conditions at the old analytic centre $\bar{u}$

$$\begin{aligned} \bar{X}\bar{s} &= \bar{\nu}, \\ A^T\bar{u} + \bar{s} &= \bar{c}, \quad \bar{s} > 0 \\ A\bar{x} &= 0, \quad \bar{x} > 0, \end{aligned} \tag{29}$$

one must efficiently compute the new analytic centre u, that satisfies

$$\begin{aligned} \begin{pmatrix} Xs \\ \Omega\sigma \end{pmatrix} &= \begin{bmatrix} \nu \\ e \end{bmatrix} \\ A^T u + s &= c, \quad s > 0 \\ B^T u + \sigma &= r, \quad \sigma > 0 \\ Ax + B\alpha &= 0, \quad x > 0, \alpha > 0, \end{aligned} \tag{30}$$

where α and σ are the new primal variables and dual slacks respectively, $\Omega = \text{diag}(\alpha)$, and $\bar{\nu}$ and ν are the old and new weights of the old cuts. As the objective cut $b^T u \geq z_{lower}$ is the first constraint in $A^T u \leq c$, ν and $\bar{\nu}$ differ only in their first entry, where $\nu - \bar{\nu}$ equals the number of rows of B.

Adding a cut to the localization set eliminates a portion of the dual space, which leads in most cases to the infeasibility of the current analytic centre $\bar{u}$. This corresponds to introducing a new variable in the primal space. Dual feasibility is difficult to restore, especially when the new cut is deep. Primal feasibility can always be recovered. The idea is to find a search direction d_x^{recov} and a step length ε that gets $\bar{x} + \varepsilon d_x^{recov}$ into the interior of the dual feasible region of the new master problem, i.e.,

$$x = \bar{x} + \varepsilon d_x^{recov}, \qquad Ax + B\alpha = 0, \quad x > 0, \quad \alpha > 0.$$

We can generalize the direction proposed by Mitchell [25] to get

$$\begin{aligned} d_x^{recov} &= -N^{-1}\bar{X}^2 A^T (AN^{-1}\overline{X}^2 A^T)^{-1} Be \\ \alpha &= \varepsilon.e \end{aligned}$$

Hence, $Ad_x^{recov} = -AN^{-1}\overline{X}^2 A^T (AN^{-1}\overline{X}^2 A^T)^{-1} Be = -Be$ and $Ax + B\alpha = A\bar{x} = 0$. An appropriate step size ε should be taken to ensure that $x > 0$, i.e., $\bar{x}_j + \varepsilon d_{x_j}^{recov} > 0$ for all j, the maximum value that ε can take is given by the ratio test

$$\varepsilon_{\max} = \min_j \{-\frac{\bar{x}_j}{d_{xj}^{recov}} : d_{xj} < 0\}. \tag{31}$$

We choose the step size $\varepsilon^* \in (0, \varepsilon_{\max})$ as the minimizer of the potential function $\varphi_p(x, \alpha)$ in the $[d_x^{recov}, e]$ direction.

A recent paper by Goffin and Vial [15] proposes a more rigorous way to recover primal feasibility. In their paper they consider the non-weighted case with central cuts.

Here we extend their analysis to the weighted case with all types of cuts, especially deep cuts.

Let us define $V = A\bar{X}\bar{S}^{-1}A^T$ as Dikin's matrix at the point $(\bar{x}, \bar{s})$ and $H = B^T V^{-1} B$ as the variance-covariance matrix of the new cuts in the metric defined by Dikin's matrix, where $\bar{X} = \text{diag}(\bar{x})$ and $\bar{S} = \text{diag}(\bar{s})$. Taking the new iterates as $u = \overline{u} + d_u^{recov}$, $x = \bar{x} + d_x^{recov}$ and $s = \bar{s} + d_s^{recov}$, equations (30) reduce to

$$\begin{aligned} \overline{S} d_x^{recov} + \overline{X} d_s^{recov} &= \nu - \bar{\nu} \equiv \hat{\nu} \\ A^T d_u^{recov} + d_s^{recov} &= c - \bar{c} \equiv \widehat{c} \\ A d_x^{recov} + B\alpha &= 0 \\ B^T d_u^{recov} + \sigma &= r - B^T \overline{u} \\ \Omega\sigma &= e. \end{aligned}$$

Solving the previous system leads to

$$\begin{aligned} d_u^{recov} &= -V^{-1}(B\alpha + A\overline{S}^{-1}(\hat{\nu} + \overline{X}\widehat{c})); & d_s^{recov} &= -A^T d_y^{recov} + \widehat{c} \\ d_x^{recov} &= -\bar{S}^{-1}\bar{X} d_s^{recov} + \overline{S}^{-1}\hat{\nu}; & \sigma &= \Omega^{-1} e \end{aligned} \tag{32}$$

where α is the solution to

$$-H\alpha + \alpha^{-1} + g = 0, \quad \alpha > 0 \tag{33}$$

where H and V are as defined previously, $\alpha^{-1} = \Omega^{-1} e$ and $g = B^T \bar{y} - r - B^T V A S^{-1}[\hat{\nu} + \overline{X}\widehat{c}]$.

Equation (33) is the first order optimality condition when maximizing

$$F(\alpha) = \sum_j \ln \alpha_j - \frac{1}{2}\alpha^T H\alpha + g^T\alpha. \tag{34}$$

$F(\alpha)$ is a concave, self-concordant barrier function corresponding to the quadratic programming problem [2]

$$\max_{\alpha > 0} -\frac{1}{2}\alpha^T H\alpha + g^T\alpha.$$

Furthermore, $F(\alpha)$ is bounded from above by the concave function $e^T\alpha - \frac{1}{2}\alpha^T H\alpha + g^T\alpha$, so it admits a finite solution. It can be maximized using Newton's method where the search direction d_α is given by

$$d_\alpha = -[F''(\alpha)]^{-1} F'(\alpha) = (\alpha^{-2} + H)^{-1}(\alpha^{-1} - H\alpha + g).$$

The complete procedure is summarized in Figure 3.

The new variables are then updated using the search direction (32), resulting in

$$\begin{aligned} u &= \overline{u} + d_u^{recov}; & s &= \bar{s} + d_s^{recov} \\ x &= \bar{x} + d_x^{recov}; & \sigma &= \alpha^{-1}. \end{aligned}$$

Initialization: $\alpha^0 > 0$. ε accuracy.
Iteration: while $||d_\alpha|| > \varepsilon$
1. $d_\alpha = (\alpha^{-2} + H)^{-1}(\alpha^{-1} - H\alpha + g)$
2. If $||d_\alpha|| > 1$ then
   - $\beta = \max\{F(\alpha + \beta d_\alpha) : \alpha + \beta d_\alpha > 0\}$
3. else
   - $\beta = 1$
4. $\alpha \leftarrow \alpha + \beta d_\alpha$

End while

Fig. 3. Maximizing $F(\alpha)$ using Newton's method.

In our implementation, an initial α^0 is calculated by approximating H by $diag(H_{jj})$. Therefore, (33) reduces to a series of simple quadratic equations

$$H_{jj}(\alpha_j^0)^2 - g_j\alpha_j^0 - 1 = 0, \ \forall j \tag{35}$$

whose positive solution is given by

$$\alpha_j^0 = \frac{g_j + \sqrt{g_j^2 + 4H_{jj}}}{2H_{jj}}, \ \forall j.$$

The Newton's procedure then proceeds as described in Figure 3. This procedure does not guarantee that $x > 0$ and $s > 0$. A primal point can be constructed by taking an appropriate step ε so that $x + \varepsilon d_x^{recov} > 0$ as done in (31). Unfortunately, there is no guarantee to recover a dual feasible solution. This fact favors the primal Newton's method to calculate the next analytic centre when adding cuts.

3.2.2. The primal Newton's method. In previous work [10], [11], the primal Newton's method in ACCPM was based on the maximization of Karmarkar's potential function [20]

$$\left\{ \max_{x,x_0>0} \varphi_{KP}(x) = (2m)\log(c^T x - z_{\text{lower}}x_0) - \sum_{i=1}^{m} \log x_i : Ax = x_0 b \right\}.$$

In this paper we use the weighted primal log potential function $\varphi_p(x)$ defined in (23). The primal Newton's method starts from a strictly primal feasible point x and calculates the primal direction d_x as

$$\begin{aligned} d_x &= N^{-\frac{1}{2}} X \left[\nu^{\frac{1}{2}} - N^{-\frac{1}{2}} X s(x) \right] \\ s(x) &= c - A^T u(x) \\ u(x) &= (AN^{-1}X^2A^T)^{-1} AN^{-1}X^2 c. \end{aligned}$$

The method proceeds iteratively, by updating the primal iterates as

$$x^+ = x + \alpha_p d_x.$$

The step size α_p is chosen so that $x^+ > 0$. In practice, a line search along d_x is performed on $\varphi_p(x+\alpha_p d_x)$. Let us define $q(x)$ as $\nu^{\frac{1}{2}} - N^{-\frac{1}{2}} Xs(x)$. The quantity $\eta_p(x) = ||q(x)||^2$ measures the proximity of a point x to the analytic centre. When far from the analytic centre ($\eta_p(x) > 1$), the primal Newton's procedure decreases the primal potential with a constant amount. In the vicinity of the analytic centre ($\eta_p(x) \leq 1$), a full Newton's step ($\alpha_p = 1$) guarantees strict primal feasibility and quadratic convergence [41]. The full procedure is summarized in Figure 4.

3.3. Calculating the new analytic centre after branching

After branching, the constraint matrix is partitioned according to the branching constraints and the analytic centre is recalculated for each child node. In the general ACCPM context, this corresponds to the case of calculating the analytic centre after dropping cuts. To calculate the next analytic centre, we propose to use a dual Newton's method.

Suppose that the master problem at a parent node is

$$\max \left\{ b^T u : b^T u \geq z_{\text{lower}},\, A^T u \leq c,\, B^T u \leq r \right\}$$

where $B^T u \leq r$ do not satisfy the branching constraints while $A^T u \leq c$ do. In other words, the matrix $[A; B]$ corresponds to the coefficient matrix of the parent node, which is partitioned to lead to the coefficient matrix A at one of the child nodes. Therefore, the initial relaxed master problem at the child node is $\max \left\{ b^T u : b^T u \geq z_{\text{lower}},\, A^T u \leq c \right\}$. Note that the lower bound z_{lower} at the parent node is used as an initial lower bound at the child node. Let us again use the compact notation as in paragraph 3.2.1. Therefore, we have the analytic centre $(\bar{x}, \bar{s}, \bar{u})$ of the following system

$$\begin{aligned} &\begin{pmatrix} \bar{X}\bar{s} \\ \bar{\Omega}\bar{\sigma} \end{pmatrix} = \begin{bmatrix} \bar{\nu} \\ e \end{bmatrix}, \\ &A^T \bar{u} + \bar{s} = c, \quad \bar{s} > 0, \\ &B^T \bar{u} + \bar{\sigma} = r, \quad \bar{\sigma} > 0, \\ &A\bar{x} + B\bar{\alpha} = 0, \quad \bar{x} > 0, \quad \bar{\alpha} > 0. \end{aligned} \tag{36}$$

Initialization: $x^0 > 0$, $Ax^0 = 0$. The centering parameter is $0 < \varepsilon < 1$.
Iteration: while $||q(x)|| > \varepsilon$

1. $u(x) = (AN^{-1}X^2A^T)^{-1}AN^{-1}X^2c$
2. $s(x) = c - A^T u(x)$
3. $q(x) = \nu^{\frac{1}{2}} - N^{-\frac{1}{2}} Xs(x)$
4. $d_x = N^{-\frac{1}{2}} Xq(x)$
5. If $||q(x)|| > 1$ then
   - $\alpha_p = \max\{\varphi_p(x + \alpha d_x) : x + \alpha d_x > 0\}$
6. else
   - $\alpha_p = 1$
7. $x \leftarrow x + \alpha_p d_x$

End while

Fig. 4. The Primal Newton's method to calculate the analytic centre.

and we want to calculate the new analytic centre (x, s, u) to

$$\begin{aligned} Xs &= \nu, \\ A^T u + s &= c, \quad s > 0, \\ Ax &= 0, \quad x > 0, \end{aligned} \tag{37}$$

As the current analytic centre $(\overline{u}, \overline{s})$ is readily dual feasible, we use a dual Newton's method to compute the next analytic centre.

Starting from $(\overline{u}, \overline{s})$, the Newton's method calculates search directions

$$\begin{aligned} d_u &= -(AN\overline{S}^{-2}A^T)^{-1}AN\overline{S}^{-1}e \\ d_s &= -A^T d_u = A^T(AN\overline{S}^{-2}A^T)^{-1}AN\overline{S}^{-1}e \end{aligned}$$

and selects a step size α_d so that the new iterates $(\overline{u} + \alpha_d d_u, \overline{s} + \alpha_d d_s)$ remain strictly feasible, i.e. $A^T d_u + d_s = 0$ and $\overline{s} + \alpha_d d_s > 0$. In practice, α_d is found by a line search along $\varphi_D(\overline{s} + \alpha_d d_s)$.

Let us define $x(s)$ as $N\overline{S}^{-1}(e - \overline{S}^{-1}d_s)$ and $q(s)$ as $\nu^{\frac{1}{2}} - N^{-\frac{1}{2}}\overline{S}x(s)$. The progress of the method is measured by $||q(s)||$ which will approach 0 as the iterates approach the analytic centre. When $||q(s)|| < 1$, $x(s)$ is primal feasible and the Newton's procedure will generate a sequence of iterates hat converges quadratically to the analytic centre. When very far from the analytic centre, the step size α_d is chosen so that the weighted dual potential function is increased by a guaranteed constant amount. The complete dual Newton's algorithm is detailed in Figure 5.

4. Implementation and testing

The AC-BP algorithm is coded under the MATLAB 5.3 environment. The subproblems are solved using CPLEX-MIP 7.5. Matlab is chosen for its sparsity handling capability and easy programming environment that enables the testing of new ideas fairly quickly. The testing is done on a Sun Ultra-10/440 workstation.

Initialization: $s > 0$, $A^T u + s = c$. The centering parameter is $0 < \varepsilon < 1$.
Iteration: while $||q(s)|| > \varepsilon$

1. $q(s) = N^{\frac{1}{2}}S^{-1}A^T(ANS^{-2}A^T)^{-1}ANS^{-1}e$
2. $x(s) = N^{\frac{1}{2}}S^{-1}(\nu - q(s))$
3. $d_u = -(ANS^{-2}A^T)^{-1}ANS^{-1}e = -(ANS^{-2}A^T)^{-1}Aq(s)$
4. $d_s = N^{-\frac{1}{2}}Sq(s)$
5. If $||q(s)|| > 1$ then
   - $\alpha = \max\{\varphi_D(s + \alpha d_s) : s + \alpha d_s > 0\}$
6. else
   - $\alpha = 1$
7. $s = s + \alpha d_s$
8. $u = u + \alpha d_u$

End while

Fig. 5. The dual Newton's method to calculate the analytic centre.

4.1. Implementation issues

For the AC-BP algorithm, we use a depth first search strategy until a first feasible solution is identified. The algorithm exploits readily generated information both at the search tree and ACCPM.

At the parent node, the columns that satisfy the branching rule and the best lower bound z_{lower} are used to initialize the problem at the child nodes. It usually takes few calls to the subproblems to get to the new lower bound at the child node. At the level of ACCPM, it takes few iterations to find the next analytic centre both when cuts are added and when cuts are deleted. In addition, the incumbent is used to stop the iterations at the level of ACCPM without the need to solve the full master problem to optimality. As soon as the lower bound exceeds the incumbent, ACCPM is stopped and the node is fathomed.

The predominant computational effort at every Newton's iteration, is the solution of the least squares problems. They are solved using Matlab's standard least-squares solver which is based on a Cholesky factorization.

To ensure the boundedness of the localization set, initial lower and upper bounds ($\ell \leq u \leq \mu$) are added. The choice of these box constraints is crucial since larger bounds are expected to slow down the method. The choice of these bounds should be based on the problem parameters. For instance, consider the relaxed Lagrangean problem $[KP_\lambda]$

$$\begin{aligned} \min \ & \sum_{l=1}^{L} (c_{kl} - \lambda_l) y_{kl} + f_k z_k \\ s.t. \ & \sum_{l=1}^{L} D_l y_{kl} \leq V_k z_k \\ & y_{kl}, z_k = 0, 1 \qquad\qquad l = 1, ..., L. \end{aligned}$$

This problem has a feasible solution of ($y_{kl} = 0, \quad l = 1, ..., L; \quad z_k = 0$), which implies that the optimal objective value is nonpositive. Therefore the upper bound should be set high enough to allow for the objective to be nonpositive. An upper bound of $\max_{k,l} c_{kl} + \max_k f_k$ achieves that. Let us suppose that the solution is given by $\widetilde{y}_{kl} \neq 0$ and obviously $\widetilde{z}_k = 1$. Then, the following inequalities hold

$$\begin{aligned} \sum_{l=1}^{L} (c_{kl} - \lambda_l)\widetilde{y}_{kl} + f_k \widetilde{z}_k &= \sum_{l=1}^{L} (c_{kl} - \left\{ \max_{k,l}(c_{kl}) + \max_k f_k \right\})\widetilde{y}_{kl} + f_k \widetilde{z}_k \\ &\leq \sum_{l=1}^{L} (c_{kl} - \max_{k,l} c_{kl})\widetilde{y}_{kl} - \max_k f_k \sum_{l=1}^{L} \widetilde{y}_{kl} + f_k \leq 0. \end{aligned}$$

The lower bound is set to $\min_k(c_{kl})$, since for all $\lambda_l < \min_k(c_{kl})$, the cost coefficients $(c_{kl} - \lambda_l)$ are positive and so the optimal objective is zero.

At the level of ACCPM, we use a stopping criterion of

$$\frac{z_{\text{upper}} - z_{\text{lower}}}{z_{\text{lower}}} \leq 10^{-4}$$

The criterion for the recovery of primal feasibility is 10^{-7}, and for the approximate analytic centre is 0.1 both for the primal and dual Newton's methods.

4.2. The test problems

To test the ACCPM-based branch-and-price approach (AC-BP), we use randomly generated instances of the capacitated facility location problem with single sourcing and the bin-packing problem. The CFLSS instances are generated following the procedure in [18]. The coordinates of the facilities and customer zones are generated from Uniform(10, 200). The costs are determined as $c_{kl} = \rho e_{kl}$; where e_{kl} is the Euclidean distance between facility k and customer l and ρ is a positive scalar. The demand, capacities and fixed costs are generated from Uniform(10,50), Uniform(100,500) and Uniform(300,700) respectively. The instances range from 5 potential facility locations and 15 customers to 10 potential facility locations and 60 customers. They are denoted by CflssK_L where K is the number of potential facilities and L is the number of customer zones. The bin-packing test problems are similar to the triplet instances in [8] where exactly 3 items are assigned to each bin. These instances are denoted by BinT_L where L is the number of items. We also generate random instances of the bin-packing problem as done in [39]. The bin capacity is 100 and the item demands are generated from Uniform(1,100). These instances are denoted by BinG_L where L is the number of items.

4.3. The performance of ACCPM

In this section we evaluate the efficiency and stability of ACCPM. We assess the empirical rate of convergence of the method and its speed in generating the analytic centres.

Figure 6 depicts the progress of the lower and upper bounds as iterations proceed for an instance of CFLSS with 10 potential facility locations and 60 customers. As the Figure reveals, the method does not suffer from tailing effects and the lower bound converges rapidly when enough cuts are available. Two phases are identified. During the first phase, the bound does not improve significantly as the method is still gathering information about the problem by generating the necessary cutting planes. Then, the bound improves sharply and approaches the optimum value. This aspect is very important in a cutting plane scheme as once the bound improves the algorithm could be stopped without sacrificing the quality of the lower bound.

During the first phase, it takes from 1 to 6 iterations to find the new analytic centre, while during the second phase, it takes an average of 3 iterations. Similarly the Figure shows that during the first phase the recentering procedure takes a maximum of 2 iterations as compared to the second phase where it takes between 2 and 9 iterations.

Table 1 shows the average number of iterations to calculate the analytic centre (AC) at the root node (Node 0) and at the rest of the nodes (Rest), the average number of iterations to recover primal feasibility (recovering steps) at the root node (Node 0) and at the rest of the nodes (Rest) and the number of dual iterations to recalculate the analytic centre after branching (Dual Iters).

The number of Newton's iterations to find the analytic centre ranges from 3 to 4 for CFLSS and 1 to 2 for BP. Note that for CFLSS, the number of added cuts at every call to the oracle corresponds to the number of potential facility locations, while for BP, it is a single cut. Even though the number of added cuts for CFLSS ranges from 5 to 10, the

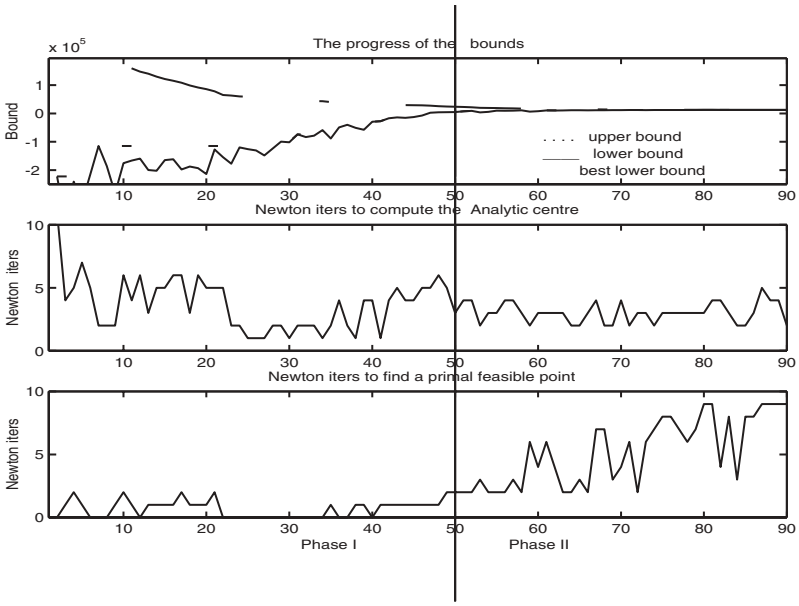

Fig. 6. The progress of the lower and upper bounds in ACCPM.

number of iterations needed to calculate the analytic centre is fairly constant (between 3 and 4) for all nodes.

The number of steps to recover primal feasibility is around 3 for the CFLSS. For BP it is 0 because the approximate solution of (35) for the single cut case gives the optimal value.

The dual Newton's method that is used to find the first analytic centre after branching requires a number of iterations which is around 10 for the CFLSS and around 6 for the BP. This relatively high number may be due to the fact that a large number of cuts from the parent node is eliminated by the branching rule when initializing the child node.

4.4. The performance of the branch-and-price algorithm

In this section we discuss the performance of the analytic centre branch-and-price (AC-BP) algorithm. We compare it against the commercial solver Cplex-MIP 7.5 as well as against a classical branch-and-price algorithm (K-BP). K-BP is coded in Matlab and is similar to AC-BP, except for the use of Kelley's cutting plane method [21] instead of ACCPM. We compare the three solution methods based on the number of nodes explored and the CPU time required. We impose a time limit of 60 minutes for the randomly generated bin-packing instances (BinG), 120 minutes for the triplet bin-packing instances (BinT). There is no time limit for the CFLSS instances.

Table 2 displays the quality of the Lagrangean bound (Lag Bnd) and the quality of the LP bound (LP Bnd) as a percentage of the optimal solution. As expected, the

Table 1. Summary statistics for ACCPM

	AC		Recentering Steps		Dual
	Node 0	Rest	Node 0	Rest	Iters
BinG_10	2	0.00	0	0.00	4.40
BinG_20	2	1.25	0	0.00	5.95
BinG_30	1	3.38	0	0.00	4.67
BinG_40	2	0.83	0	0.00	5.54
BinG_50	1	1.50	0	0.00	3.43
BinG_60	2	2.49	0	0.00	7.08
BinG_70	1	1.89	0	0.00	6.24
BinG_80	1	1.59	0	0.00	5.72
BinG_90	2	1.14	0	0.00	6.76
BinG_100	1	2.50	0	0.00	6.62
BinG_120	1	2.23	0	0.00	6.54
BinG_140	2	1.65	0	0.00	6.70
BinG_170	2	0.32	0	0.00	6.27
BinG_200	1	2.15	0	0.00	6.91
BinT_12	1	—	0	—	—
BinT_21	1	—	0	—	—
BinT_30	1	—	0	—	—
BinT_51	1	—	0	—	—
BinT_60	1	2.73	0	0.00	5.57
BinT_72	1	2.95	0	0.00	6.23
BinT_81	1	1.96	0	0.00	5.56
BinT_90	1	3.74	0	0.00	7.00
BinT_102	1	3.04	0	0.00	6.28
BinT_111	1	3.50	0	0.00	7.98
BinT_120	1	2.45	0	0.00	6.23
BinT_141	1	3.42	0	0.00	7.44
BinT_150	1	3.49	0	0.00	8.97
BinT_180	1	2.64	0	0.00	6.64
BinT_210	1	2.88	0	0.00	7.96
Cflss5_15	3	—	2	—	—
Cflss5_20	4	2.10	2	1.54	9.46
Cflss5_25	3	4.28	2	3.29	9.26
Cflss5_30	3	3.23	2	3.28	10.07
Cflss8_24	4	3.74	3	2.74	9.43
Cflss8_32	4	—	3	—	0.00
Cflss8_40	4	3.68	2	2.29	10.66
Cflss8_48	3	3.25	3	3.24	12.25
Cflss10_30	3	2.93	2	2.31	9.36
Cflss10_40	3	2.76	3	1.64	9.34
Cflss10_50	3	3.03	2	3.29	10.19
Cflss10_60	4	3.23	2	2.79	9.73

—: no branching was done.

Lagrangean bound is superior to the LP bound on all instances. Specifically, the Lagrangean bound is on average 99.12% , while the LP bound is on average 90.41% over all BinG and CFLSS instances (for BinT instances, the two bounds are 100%). The rest of the columns of Table 2 display the number of nodes of the search tree (Nodes) and the CPU time (CPU) for AC-BP, Cplex-MIP 7.5 and K-BP respectively as well as the optimality gap (Gap) for problems that Cplex-MIP 7.5 fails to solve to optimality.

For BinG instances, AC-BP and K-BP solve all the instances exploring a comparable number of nodes (32.27 for K-BP and 37.07 for AC-BP). AC-BP, however, requires on

Table 2. Comparsion between AC-BP, K-BP and Cplex-MIP 7.5

	LP	Lag.	AC- BP		Cplex-MIP 7.5			K-BP	
	bound	bound	Nodes	CPU^b	Nodes	Gap	CPU^b	Nodes	CPU^b
BinG_10	77	92.86	4	0.03	77	—	0.01	4	0.05
BinG_20	83	95.83	9	0.06	511	—	0.08	9	0.15
BinG_30	93.13	96.19	16	0.17	10082	7.14%(1)^a	60	16	0.41
BinG_40	96.84	100	18	0.3	505	—	0.29	20	0.76
BinG_50	93.7	98.15	24	0.54	5327	3.84%(1)	60	23	1.38
BinG_60	89.12	100	18	0.69	223	—	0.93	26	2.16
BinG_70	87.74	100	20	1.04	460	—	2.47	25	3.62
BinG_80	97.17	99.07	31	2.79	4057	2.43%(1)	60	37	10.05
BinG_90	91	99.05	38	3.21	5494	1.92%(1)	60.1	31	7.86
BinG_100	92.98	100	41	5.07	1271	—	22.95	41	8.9
BinG_120	93.87	100	48	7.81	1458	—	30.93	52	23.42
BinG_140	91.91	99.35	60	17.78	1066	9.86%(7)	60	62	30.41
BinG_150	93.42	99.38	67	20.32	1009	7.89%(6)	60	66	39.67
BinG_170	96.55	99.89	69	28.75	527	$***$	60	71	50.13
BinG_200	97.43	100	93	57.69	415	$***$	60	1	71.18
Min	77	92.86	4	0.03				4	0.05
Max	97.43	100	93	57.69				71	71.18
Average	91.66	98.65	37.07	9.75				32.27	16.68
BinT_12	—	—	1	0.02	10	—	0.01	1	0.06
BinT_21	—	—	1	0.07	2066	—	0.06	1	0.31
BinT_30	—	—	1	0.21	8686	10%(1)^a	> 120	1	0.89
BinT_51	—	—	1	1.34	2085	5.88%(1)	> 120	1	3.58
BinT_60	—	—	1	1.94	3919	5%(1)	> 120	1	21.41
BinT_72	—	—	1	3.76	3280	4.16%(1)	> 120	1	34.21
BinT_81	—	—	1	6.41	10276	3.70%(1)	> 120	1	37.68
BinT_90	—	—	1	12.09	2439	3.33%(1)	> 120	1	56.54
BinT_102	—	—	1	15.69	9865	2.94%(1)	> 120	1	78.62
BinT_111	—	—	14	22.61	3130	2.70%(1)	> 120	#(99.72%)^d	> 120
BinT_120	—	—	20	34.03	7098	2.50%(1)	> 120	#(99.90%)	> 120
BinT_141	—	—	36	55.96	5303	2.12%(1)	> 120	#(99.50%)	> 120
BinT_150	—	—	34	88.8	5116	$***$	> 120	#(99.52%)	> 120
BinT_180	—	—	17	118.92	5594	$***$	> 120	#(82.75%)	> 120
Min			1	0.02^c	10	2.12^c			0.06^c
Max			36	15.69^c	10276	10^c			78.62^c
Average			9.29	4.61^c	4919.07	4.23^c			25.92^c
Cflss_5_15	92.07	100	1	0.1	1	—	0.02	1	0.55
Cflss_5_20	89.5	97	109	3.74	3	—	0.03	101	5.43
Cflss_5_25	84.8	99.96	5	0.73	1	—	0.04	3	0.68
Cflss_5_30	90.86	99.59	11	1.38	1	—	0.05	11	2.51
vCflss_8_24	87.87	100	1	0.58	1	—	0.06	1	2.1
Cflss_8_32	89.59	99.94	21	2.11	1	—	0.07	37	5.5
Cflss_8_40	95.87	99.98	25	3.71	16	—	0.21	27	20.33
Cflss_8_48	80.32	99.9	41	12.75	46	—	0.57	180	69.9
Cflss_10_30	91	99.03	193	14.51	44	—	0.56	192	30.4
Cflss_10_40	92.61	100	1	1.79	14	—	0.13	1	9.93
Cflss_10_50	90	99.98	47	8.92	40	—	0.67	55	29.78
Cflss_10_60	85.32	99.52	39	28.38	100	—	1.84	175	219.06
Min	80.32	97	1	0.1	1		0.02	1	0.55
Max	95.87	100	193	28.38	100		1.84	192	219.06
Average	89.15	99.58	41.17	6.56	22.33		0.35	65.33	33.01
Average over BinG &CFLSS Instances	90.41	99.12	39.12	8.16				48.85	24.85

$(.)^a$: Difference between lower and upper bound.
$(.)^b$: All CPU's in minutes.
$(.)^c$: Taken over the first 9 instances (those solved successfully by AC-BP and K-BP).
$***$: Failed to find a feasible solution within the allowed time.
: Did not get past node 0. $(.)^d$: found lower bound as a percentage of best lower bound.
— : Gap = 0.

Table 3. Comparsion between AC-BP and K-BP: Warm starting

Problem	A-BP				K-BP			
	SP0	SPrest	CPU0	CPUrest	SP0	SPrest	CPU0	CPUrest
BinG_10	16	4	0.01	0	12	3	0.03	0.01
BinG_20	36	2.62	0.03	0	33	1.88	0.1	0.01
BinG_30	50	2.93	0.07	0	55	2.4	0.23	0.01
BinG_40	69	2.71	0.12	0.01	76	2.58	0.43	0.01
BinG_50	86	2.57	0.17	0.01	104	2.59	0.84	0.01
BinG_60	94	2.59	0.24	0.01	120	3.44	1.18	0.02
BinG_70	103	2.58	0.34	0.02	158	2.79	2.37	0.02
BinG_80	156	2.93	0.9	0.03	228	1.64	7.4	0.03
BinG_90	162	2.14	1.06	0.02	201	3.57	5.06	0.05
BinG_100	188	2.38	1.59	0.03	208	1.82	4.97	0.04
BinG_120	207	2	2.77	0.03	307	2.45	14.93	0.07
BinG_140	241	2.68	4.64	0.13	326	3.98	17.02	0.1
BinG_150	252	2.67	5.73	0.11	371	3.15	22.3	0.12
BinG_170	339	2.18	11.08	0.08	442	2.09	28.46	0.09
BinG_200	425	2	24.01	0.1	564	—	70.1	—
Min	16	2	0.01	0	12	1.64	0.03	0.01
Max	425	4	24.01	0.13	564	3.98	70.1	0.12
Average	161.6	2.6	3.52	0.04	213.67	2.67	7.52	0.04
BinT_12	22	—	0.02		24	—	0.06	—
BinT_21	48	—	0.07		62	—	0.3	—
BinT_30	74	—	0.2		96	—	0.87	—
BinT_51	142	—	1.3		167	—	3.53	—
BinT_60	178	—	1.88		299	—	21.24	—
BinT_72	210	—	3.66		336	—	33.95	—
BinT_81	247	—	6.26		355	—	37.38	—
BinT_90	268	—	11.92		409	—	56.14	—
BinT_102	301	—	15.47		471	—	78.07	—
BinT_111	351	2	17.56	0.18	521	—	120	#
BinT_120	382	2	26.85	0.15	537	—	120	#
BinT_141	456	2	40.89	0.17	554	—	120	#
BinT_150	495	2	67.32	0.31	564	—	120	#
BinT_180	614	2	95.84	0.68	539	—	120	#
Min	22	2	0.02	0.15	24	—	0.06	—
Max	614	2	95.84	0.68	564	—	120	—
Average	244.15	2	14.88	0.2	338.08	—	59.4	—
Cflss_5_15	18	—	0.1	—	114	—	0.55	
Cflss_5_20	24	6.25	0.13	0.03	66	9.65	0.31	0.05
Cflss_5_25	32	9	0.3	0.11	88	9.5	0.49	0.09
Cflss_5_30	41	8	0.53	0.08	125	12.3	0.84	0.17
Cflss_8_24	29	—	0.58	—	190	—	2.1	
Cflss_8_32	29	6	0.66	0.07	139	8.11	1.48	0.11
Cflss_8_40	42	6.79	1.21	0.1	276	11.73	4.25	0.62
Cflss_8_48	58	11.7	2.15	0.26	257	16.33	4.69	0.36
Cflss_10_30	38	5.73	0.77	0.07	93	6.93	1.77	0.15
Cflss_10_40	49	—	1.79	—	336	—	9.93	
Cflss_10_50	44	7.04	2.01	0.15	162	10.69	4.42	0.47
Cflss_10_60	62	8.63	4.49	0.63	232	17.51	8.03	1.21
Min	18	5.73	0.1	0.03	66	6.93	0.31	0.05
Max	62	11.7	4.49	0.63	336	17.51	9.93	1.21
Average	38.83	7.68	1.23	0.17	173.17	11.42	3.24	0.36
Average over all instances	148.19	7.43	6.54	0.14	241.64	7.05	23.39	0.2

: Did not get past node 0.
[a] : SPrest as a percent of SP0
[b] : CPUrest as a percent of CPU0
—: No branching was done.

average 58% the CPU time required by K-BP. Cplex-MIP 7.5, on the other hand, fails to solve 8 out of the 15 instances. What is remarkable, is that it fails to find even a single feasible solution to the largest two instances. Although the allowed CPU time for the BinT instances was doubled, Cplex-MIP 7.5 solves the two smallest instances and fails to find a feasible solution to the two largest instances. K-BP fails to get past the first node on the five largest instances, while AC-BP solves all the instances to optimality exploring an average of 9.29 nodes. For CFLSS instances, Cplex-MIP 7.5 performs extremely well solving most instances within a maximum of two seconds, outperforming both AC-BP and K-BP. First it is worth mentioning that Cplex-MIP is written in C/C++ while AC-BP and K-BP are written in Matlab. Programs written in Matlab are much slower than those written in C/C++ by a factor that is over 30. When comparing AC-BP against K-BP, AC-BP explores less nodes (41.17 vs 65.33) and requires less computational time (6.56 vs 33.01).

Table 3 gives a direct comparison of K-BP and AC-BP based on the average number of subproblems called at the first node and at the rest of the nodes. It displays the number of calls to the subproblem at the root node (SP0), the average number of calls to the subproblem at the rest of the nodes (SPrest), the CPU time at the root node (CPU0) and the average CPU at the rest of the nodes (CPUrest) both for AC-BP and K-BP.

The table reveals that AC-BP is consistently better than K-BP in making fewer calls to the subproblems and requiring less computational time at node 0. AC-BP calls an average of 148.19 subproblems and takes an average CPU time of 6.54 minutes, while K-BP calls an average of 241.64 subproblems and takes an average CPU time of 23.39 minutes. When it comes to the efficient use of past information, K-BP and AC-BP call around 7 subproblems at nodes other than the root node with K-BP requiring slightly more computational time. Given these observations, we can safely make these remarks:

- While AC-BP and K-BP explore comparable number of nodes, AC-BP is more efficient at processing each node. It calls fewer subproblems and requires less computational time. This implies that AC-BP has a clear advantage over K-BP as it does not compromise in any other compartment of the algorithm (warm starting, efficient branching) but improves in one: the solution of the master problems.
- The fact that AC-BP and K-BP explore comparable number of nodes reveals that branching is done efficiently under the two cases (despite the fact that columns are generated and ranked differently).
- As AC-BP spends less time per node, it is clear that the efficiency of AC-BP is mainly attributed to ACCPM.
- Cplex-MIP on one hand and AC-BP and K-BP on the other hand perform differently on different problems. This suggests that branch-and-cut methods, as in Cplex-MIP, is suitable for CFLSS problems while branch-and-price, as in AC-BP and K-BP, are suitable for bin-packing problems.

5. Conclusion

It is widely observed that despite the potential of interior-point methods in dealing with large-scale linear programs, they were not widely used to solve large-scale integer programs. According to [19], this is because " it is cumbersome to reoptimize an LP with an

interior-point code after rows or columns have been added" [19]. In this paper we show that it is possible to overcome this "cumbersomeness" by using a cutting plane method (ACCPM) and a Lagrangean bounding scheme. We show that information in the form of generated cuts, incumbent and lower bound is efficiently exploited in subsequent nodes both by the search scheme and by ACCPM. Recentering when adding or deleting cuts is done fairly quickly using a primal and dual interior point methods, respectively. In addition, the initialization of the primal Newton's method is done rigorously using a primal-dual approach. Finally, ACCPM provides dual information during the course of the iterations that are exploited by a dual heuristic and used to guide the branching rule.

The resulting approach is a branch-and-price algorithm where the Lagrangean dual problem is solved using ACCPM. In a first step, we provided modifications to ACCPM, specifically the use of a primal-dual approach to recover primal feasibility and the use of the dual Newton's method to calculate the analytic centre after branching. At an equal level, we provided a new branch-and-price algorithm where columns are generated based on central prices.

AC-BP was compared against the commercial solver Cplex-MIP 7.5 and a branch-and-price algorithm that is based on Kelley's cutting plane method. AC-BP was consistently better than K-BP drawing most of its efficiency from ACCPM. On bin-packing problems, AC-BP outperformed both Cplex-MIP and K-BP despite the fact that our implementation is not fully optimized, while Cplex-MIP is a state-of-the-art commercial solver. On CFLSS problems Cplex-MIP was better. This fact suggests that branch-and-cut (Cplex-MIP) and branch-and-price (AC-BP and K-BP) methods are complementary methodologies where each is suitable for certain type of problems.

Venues for future research include the incorporation of new improvements to AC-CPM in terms of the effective management of the box constraints and the possibility of eliminating them after branching. For the branch-and-price method an obvious item on the research agenda is the use of valid cuts and variable fixing strategies as well as the testing of different branching strategies.

Acknowledgements. We thank two anonymous referees for their valuable comments that helped improve the presentation of the paper.

References

1. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price : column generation for solving huge integer programs. Oper. Res. **46**, 316–329 (1998)
2. Den Hertog, D.: Interior point approach to linear quadratic and convex programming. Kluwer, London, 1994
3. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. **40**, 342–354 (1992)
4. Desrochers, M., Desrosiers, J., Soumis, F.: A column generation approach to the urban transit crew scheduling problem. Transp. Sci. **23**, 1–13 (1989)
5. Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M.M., Soumis, F.: A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In: Crainic, T.G., Laporte, G., (eds.), Fleet Management and Logistics. Kluwer, Boston, MA, 1998, pp. 57–93
6. duMerle, O.: Interior points and cutting palnes: development and implementation of methods for convex optimization and large scale structured linear programming. Ph.D Thesis, Department of Management Studies, University of Geneva, Geneva, Switzerland, 1995 (in French)
7. duMerle, O., Hansen, P., Jaumard, B., Mladenovic, N.: An interior point algorithm for minimum sum of squares clustering. SIAM J. Sci. Comput. **21**(4), 1485–1505

8. Falkenauer, E.: A Hybrid grouping genetic algorithm for bin-packing. Working paper CRIF Industrial Management and Automation. CP **50**, 106–4 (1994)
9. Geoffrion, A.M.: Lagrangean relaxation for integer programming. Math. Program. Study **2**, 82–114 (1974)
10. Goffin, J.-L., Haurie, A., Vial, J.-P.: Decomposition and nondifferentiable optimization with the projective algorithm. Manage. Sci. **38**(2), 284–302 (1992)
11. Goffin, J.-L., Vial, J.-P.: Convex nondifferentiable optimization: a survey focussed on the analytic center cutting plane method. GERAD Tech. Report G-99–17 1999, p. 47
12. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem. Oper. Res. **9**, 849–859 (1961)
13. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem: Part II. Oper. Res. **11**, 863–888 (1963)
14. Goffin, J.-L., Luo, Z.-Q., Ye, Y.: Complexity Analysis of an Interior Cutting Plane Method for Convex Feasibility Problems. SIAM J. Optim. **6**, 638–652 (1996)
15. Goffin, J.-L., Vial, J.-P.: Multiple cuts in the analytic center cutting plane method. SIAM J. Optim. **11**(1), 266–288 (1999)
16. Goffin, J.-L., Vial, J.-P.: On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. Math. program. **60**, 81–92 (1993)
17. Gondzio, J.: Warm start of the primal-dual method applied in the cutting plane scheme. Math. program. **83**, 125–143 (1998)
18. Holmberg, K., Ronnqvist, M., Yuan, D.: An exact algorithm for the capacitated facility location problem with single sourcing. Eur. J. Oper. Res. **113**, 544–559 (1999)
19. Johnson, E.L., Nemhauser, G., Savelsbergh, M.W.P.: Progress in linear programming-based algorithms for integer programming. An exposition. Informs JOC **12**(1), 1–23 (2000)
20. Karmarkar, N.K.: A new polynomial-time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
21. Kelley, J.E.: The cutting plane method for solving convex programs. J. SIAM **8**, 703–712 (1960)
22. Klincewicz, J.G., Luss, H.: A Lagrangean relaxation heuristic for capacitated facility location with single-source constraints. J. Oper. Res. Soc. **37**(5), 195–500 (1986)
23. Lemarechal, C.: Bundle Methods in Nonsmooth Optimization. Nonsmooth Optimization, Proceedings of the IIASA Workshop March 28 – April 8, 1977, Lemarechal, C., Mifflin, R., (eds.), Pergamon Press, 1978
24. Martello, S., Toth, P.: Knapsack problems: Algorithms and computer implementations. Wiley Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 1990
25. Mitchell, J.E., Todd, M.J.: Solving combinatorial optimization problems using Karmarkar's algorithm. Math. Program. **56**, 245–284 (1992)
26. Mitchell, J.E.: Computational experience with an interior point cutting plane algorithm. SIAM J. Optim. **10**(4), 1212–1227 (2000)
27. Mitchell, J.E., Borchers, B.: Using an interior point method in a branch-and-bound algorithm for integer programming, 1991. Revised 1992, Rensselaer Polytechnic Institute, Troy, N.Y.
28. Mitchell, J.E., Pardalos, P., Resende, M.G.C.: Interior point methods for combinatorial optimization. Handbook of Combinatorial Optimization, Volume **1**, Kluwer Academic Publishers, 1998, pp. 189–297
29. Neebe, A.W., Rao, M.R.: An algorithm for the fixed-charge assigning users to sources problem. J. Oper. Res. Soc. **34**(11), 1107–1113 (1983)
30. Nemirovskii, A.S., Yudin, D.B.: Problem complexity and method efficiency in optimization. John Wiley, Chichester, 1983
31. Nesterov, Y.: Cutting plane methods from analytic centers: efficiency estimates. Math. program., Ser. B **69**, 149–176 (1996)
32. Pirkul, H.: Efficient algorithms for the capacitated concentrator location problem. Comput. Oper. Res. **14**(3), 197–208 (1987)
33. Ryan, D.M., Foster, B.A.: An integer programming approach to scheduling. In: Wren, A. (ed.), computer scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, North Holland, Amsterdam, 1981, pp. 169–280
34. Scholl, A., Klein, R., Jurgens, C.: Bison: a fast hybrid procedure for exactly solving the one-dimensional bin-packing problem. Comput. Oper. Res. **24**, 667–645 (1997)
35. Sonnevend, G.: An analytical center for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In: Lecture Notes in Control and Information Sciences 84, Springer, New York, 1985, pp. 866–876
36. Valerio de Carvalho, J.M.: Exact solution of bin-packing problems using column generation and branch-and-bound. Ann. Oper. Res **86**, 626–659 (1999)
37. Vance, P.H., Johnson, E.L., Nemhauser, G.L.: Solving binary cutting stock problems using column generation and branch-and-bound. Comput. Optim. Appl. **3**, 111–130 (1994)

38. Vanderbeck, F., Wolsey, L.A.: An exact algorithm for IP column generation. Oper. Res. Lett. **19**, 151–159 (1996)
39. Vanderbeck, F.: Computational study of a column generation algorithm for bin-packing and cutting stock problems. Math. Program. Ser. A **86**, 565–594 (1999)
40. Vanderbeck, F.: On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Oper. Res. **48**(1), 111–128 (2000)
41. Ye, Y.: Interior point algorithms: Theory and analysis. John Wiley & sons, 1997
42. Zhang, Y.: Solving Large Scale Linear Programs by Interior-Point Methods Under the Matlab Environment. Technical report TR96-01, university of Maryland Baltimore county, 1996