Benders decomposition without separability: a computational study for capacitated facility location problems

Matteo Fischetti[a,*], Ivana Ljubić[b], Markus Sinnl[c]

[a]*Department of Information Engineering, University of Padua, Italy*
[b]*ESSEC Business School of Paris, France*
[c]*Department of Statistics and Operations Research, University of Vienna, Austria*

Abstract

Benders is one of the most famous decomposition tools for Mathematical Programming, and it is the method of choice e.g., in Mixed-Integer Stochastic Programming. Its hallmark is the capability of decomposing certain types of models into smaller subproblems, each of which can be solved individually to produce local information (notably, cutting planes) to be exploited by a centralized "master" problem. As its name suggests, the power of the technique comes essentially from the decomposition effect, i.e., the separability of the problem into a master problem and several smaller subproblems. In this paper we address the question of whether the Benders approach can be useful even without separability of the subproblem, i.e., when its application yields a single subproblem of the same size as the original problem. In particular, we focus on the capacitated facility location problem, in two variants: the classical linear case, and a "congested" case where the objective function contains convex but non-separable quadratic terms. We show how to embed the Benders approach within a modern branch-and-cut mixed-integer programming solver, addressing explicitly all the ingredients that are instrumental for its success. In particular, we discuss some computational aspects that are related to the negative effects derived from the lack of separability. Extensive computational results on various classes of instances from the literature are reported, with a comparison with the state-of-the-art exact and heuristic algorithms. The outcome is that a clever but simple implementation of the Benders approach can be very effective even without separability, as its performance is comparable and sometimes even better than that of the most effective and sophisticated algorithms proposed in the previous literature.

Keywords: Benders decomposition, facility location, congested capacitated facility location, perspective reformulation, cutting planes, branch-and-cut, mixed-integer convex programming

---

*Corresponding author
Email addresses: `matteo.fischetti@unipd.it` (Matteo Fischetti), `ljubic@essec.edu` (Ivana Ljubić), `markus.sinnl@univie.ac.at` (Markus Sinnl)

1. Introduction

In this article we study the multiple-allocation (or multiple-source) Capacitated Facility Location (CFL) problem and its quadratic variant known as the *congested CFL* problem. In both problems we are given a set of customers and potential facility locations. Customers have to be served by open facilities, and their demands have to be satisfied by the limited capacity of open facilities. Demand can be split and a customer can be partially served by several locations. Multiple allocations arise, for example, when customers correspond to population areas and not all the individuals from the same area have to be served by the same facility. In the problem variant with linear objective function (known as classical CFL), the solution cost, which is to be minimized, is defined as the sum of costs for opening facilities plus the costs for allocating customer demands to open facilities. In congested CFL, introduced by Desrochers et al. [10], possible congestions at open facilities imply *diseconomies of scale* expressed through convex cost functions. These convex costs arise due to a possibly larger number of overtime workers, usage of more costly materials, or by neglecting or postponing equipment maintenance schedules [20]. Hence, the goal is now to minimize the sum of facility opening costs, customer allocation costs plus the *congestion costs* (typically defined as service/production costs at facilities and/or waiting times at facilities). In congested CFL considered throughout this paper, the convex cost are modeled using quadratic functions. In congested CFL, a more balanced solution is sought after, i.e., a solution that takes into consideration expected waiting times such that open facilities are not "overloaded" by customer demands. The congestion is usually not measured in the same units as the opening and allocation costs (which can be easily expressed in currencies), hence, it is not surprising that congestion costs (being quadratic), grow much faster than the remaining (linear) part of the objective function. Consequently, it may happen that the optimal solution requires to open too many facilities, so as to keep their load at minimum. Since decision makers prefer to have a control over the number of open facilities, a p-median constraint is usually added to the problem formulation, where p is a given input parameter.

Previous work. A large body of work is available for classical CFL. We address only a small portion of published articles, and refer the reader to a more comprehensive literature overview available in [22] and a more recent one in [12]. To the best of our knowledge, the state-of-the-art exact methods for CFL are: 1) a branch-and-cut-and-price algorithm (B&C&P) by Avella and Boccia [2], which is based on the idea of reformulating CFL as a fixed charge network design problem, and separating the mixed dicut inequalities, and 2) a Lagrangian-based branch-and-bound by Görtz and Klose [17]. State-of-the-art heuristics are those proposed by Avella et al. [3] (a Lagrangian relaxation combined with a cutting plane method) and an MIP-based heuristic called *kernel search* by Guastaroba and Speranza [18]. On the contrary, congested CFL did not receive that much attention: besides the original article [10], in which the problem has been approached by column generation and optimal solutions have been reported for very small instances, the problem has been only recently considered in the master thesis [27]. However, there are plenty of models in which congestion issues at facilities are modeled using queuing theory, or considering different objective functions. For more details, see e.g., a recent survey by Boffey et al. [7] and recent literature overviews given in Aboolian et al. [1], Zhang et al. [31].

Main motivation and our contribution. Applying Benders decomposition to CFL appears very natural as it follows the "recipe" given by Benders in the early 60's [6]: keep the integer variables, and relax the "complicated" continuous variables. A careful look into the recent literature suggests however that, since the work of Wentges [30] in the 90's, and despite a wide range of approaches applied to CFL, the Benders decomposition method was not considered the solution method of choice.

The main motivation of the present article was therefore to rethink the well-known concept of Benders decomposition for CFL, considering both linear and convex nonlinear objective functions. Our goal was to provide guidelines for a modern implementation of Benders decomposition based on the MIP technology which was not available back in the 70's and 80's when most of the related articles have been published.

For classical (i.e., linear) CFL, our method is competitive and sometimes better than the state-of-the-art exact approaches available in the current literature: a branch-and-cut-and-price algorithm by Avella and Boccia [2] and a Lagrangian-based branch-and-bound by Görtz and Klose [17]. For the largest available CFL instances, our method has been run as a heuristic and successfully compared with the kernel search heuristic by Guastaroba and Speranza [18], which was shown to beat all previous CFL heuristics. For the congested case, we show how to obtain a much tighter convex Mixed-Integer Non-Linear Programming (MINLP) formulation using the perspective reformulation [15, 19]. We then demonstrate that our new decomposition approach computationally dominates the perspective reformulation solved using a state-of-the-art commercial solver based on second-order cone cuts.

2. Mixed-Integer Programming models

More formally, CFL is stated as follows: Given a set J of potential facility locations, and a set I of customers, let $d_i \geq 0$ denote demand of a customer $i \in I$, and $s_j \geq 0$ capacity of a facility $j \in J$. Opening a facility $j \in J$ costs $f_j \geq 0$, and serving a single unit of demand of customer i by facility j costs $c_{ij} \geq 0$, for all $i \in I, j \in J$. The goal is to find a subset of facility locations to open, and to serve all customers, so that the facility opening plus customer allocation costs are minimized. Let us assume without loss of generality that each customer can be allocated to every facility (if this is not the case, we will assume $c_{ij} = \infty$). In the traditional compact Mixed-Integer Linear Programming (MILP) formulation for CFL, $|J| + |I| \cdot |J|$ variables are used to model the problem. For each $j \in J$, binary variable y_j is set to one if facility j is open, and to zero, otherwise. For each $i \in I$ and $j \in J$, the continuous allocation variable $x_{ij} \geq 0$ will denote the fraction of the demand of customer $i \in I$ served by facility $j \in J$.

2.1. CFL with linear objective function

The classical CFL model reads

$$\min \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} d_i c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ij} = 1 \qquad \forall i \in I \tag{2}$$

$$x_{ij} \le y_j \qquad \forall i \in I, j \in J \tag{3}$$

$$\sum_{i\in I} d_i x_{ij} \le s_j y_j \qquad \forall j \in J \tag{4}$$

$$x_{ij} \ge 0 \qquad \forall i \in I, j \in J \tag{5}$$

$$y_j \in \{0,1\} \qquad \forall j \in J \tag{6}$$

The objective is to minimize the sum of facility opening costs, plus the customer allocation costs. Assignment constraints (2) make sure that complete customer demand is satisfied, and capacity constraints (4) make sure that the capacity of an open facility cannot be exceeded. Constraints (3) state that allocation to a facility j is only possible if this facility is open. They are redundant in the MILP formulation, but are known to significantly strengthen the quality of Linear Programming (LP) relaxation bounds. Note that, because of capacity constraints (4), the integrality condition on variables x_{ij} is not automatically satisfied by an optimal solution, hence one should add it explicitly in case single-source CFL is considered—a variant not covered in the present paper.

CFL is NP-hard, however, when a set of open facilities is fixed, the problem of finding optimal customer allocations boils down to a transportation problem that can be solved efficiently using specialized algorithms based on e.g., minimum-cost flows. Furthermore, the following well-known and very useful constraints are sufficient to guarantee that a chosen subset of open facilities may produce a feasible solution:

$$\sum_{j\in J} s_j y_j \ge \sum_{i\in I} d_i.$$

These constraints turn out to be very important for our Benders reformulation as well.

2.2. Congested CFL with convex objective function

Similarly, congested CFL can be modeled as the following Mixed-Integer Non-Linear Program (MINLP)

$$\min \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} d_i c_{ij} x_{ij} + \sum_{j\in J} F(\sum_{k\in I} d_k x_{kj}) \sum_{i\in I} d_i x_{ij} \tag{7}$$

$$\text{s.t.} \quad \sum_{j\in J} y_j = p \tag{8}$$

$$(2) - (6)$$

where $F(.)$ is the *penalty function*, which is assumed to be nonnegative, continuous and convex for nonnegative arguments. This function penalizes every additional "unit" being served by a given facility. Depending on the application, function F is used here to model the service/production costs at the facilities or the waiting times—both are expected to increase with the increasing demand in a convex fashion. Desrochers et al. [10] show that, under these assumptions, the objective function remains convex in (x, y) for all nonnegative values of x, hence we are dealing with a convex MINLP. Throughout this paper, for the ease of exposition, we will assume that $F(t)$ is a linear function, say $F(t) = at + b$, with nonnegative input coefficients a and b, though our decomposition approach (cf. Section 3) can be extended to more general cases. To prevent opening too many facilities and to allow the decision makers to control the number of open facilities, p-median constraint (8) is added to the model [10].

Using aggregated variables to model the exact demand served by facility j (namely: $v_j = \sum_{i \in I} d_i x_{ij}$, that we will refer to as *facility load*), the *congestion cost* at a facility $j \in F$ is given as $F(v_j) \cdot v_j$. Consequently, the objective function can be written as:

$$\min \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} d_i c_{ij} x_{ij} + \sum_{j \in J} F(v_j) v_j$$

Our model then reads

$$\min \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} d_i c_{ij} x_{ij} + b \sum_{j \in J} v_j + a \sum_{j \in J} v_j^2 \tag{9}$$

$$\text{s.t.} \quad \sum_{j \in J} y_j = p \tag{10}$$

$$\sum_{i \in I} d_i x_{ij} = v_j \qquad j \in J \tag{11}$$

$$\sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I \tag{12}$$

$$x_{ij} \leq y_j \qquad \forall i \in I, j \in J \tag{13}$$

$$v_j \leq s_j y_j \qquad \forall j \in J \tag{14}$$

$$x_{ij} \geq 0 \qquad \forall i \in I, j \in J \tag{15}$$

$$y_j \in \{0, 1\} \qquad \forall j \in J \tag{16}$$

To obtain a tighter model one can derive the so-called *perspective reformulation* [15, 19] of the problem, in which each quadratic term v_j^2 in the objective function is replaced by a nonnegative variable z_j, namely:

$$\min\{\sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} d_i c_{ij} x_{ij} + b \sum_{j \in J} v_j + a \sum_{j \in J} z_j \mid v_j^2 \leq z_j y_j, \forall j \in J,\ z \geq 0,\ \text{(10)-(16)}\ \} \tag{17}$$

The second-order cone (SOC) constraints $v_j^2 \leq z_j y_j$, together with the minimization of the objective function, guarantee that the quadratic load v_j^2 of a facility j is zero if the facility is closed, and it is z_j if the facility is open. Note that, for a given v_j, the smaller $y_j > 0$ the larger z_j, meaning that small fractional values of y_j are penalized by the objective function. As a result, the model automatically tends to favor integral values for the y variables, which explains why the perspective reformulation of a problem is typically much tighter than the standard one (see, e.g. [15, 19]). We therefore use (17) in our computational study.

3. Modern Benders

In this section we describe the basic steps for the design of a Benders decomposition approach to be embedded in a modern MIP solver.

3.1. An easy derivation of Benders cuts for convex problems

For the sake of generality, let the problem of interest (an MILP or a convex MINLP) be restated as

$$\min f(x,y) \tag{18}$$

$$\text{s.t. } g(x,y) \leq 0 \tag{19}$$

$$y \text{ integer} \tag{20}$$

where $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, and functions $f : \mathbb{R}^{m+n} \mapsto \mathbb{R}$ and $g : \mathbb{R}^{m+n} \mapsto \mathbb{R}^p$ are assumed to be differentiable and convex. Let $Ay \leq b$ denote the subset of linear inequalities in (19) that do not involve x, including variable bounds on the y variables (if any). To simplify our treatment, we assume that

$$S := \{y : Ay \leq b\}$$

is a nonempty polytope, while the convex sets $X(y) := \{x : g(x,y) \leq 0\}$ are nonempty, closed and bounded for all $y \in S$, as it happens in our CFL case. Problem (18)-(20) can trivially be restated as the *master problem* in the y space

$$\min w \tag{21}$$

$$\text{s.t. } w \geq \Phi(y) \tag{22}$$

$$Ay \leq b \tag{23}$$

$$y \text{ integer} \tag{24}$$

where

$$\Phi(y) := \min_{x \in X(y)} f(x,y)$$

is the convex function expressing the optimal solution value of the problem (18)-(20) as a function of y, and w is a continuous variable that captures its value in the objective function. (Problems where $X(y)$

can be empty for some $y \in S$ can be handled similarly, by adding the feasibility condition $0 \geq \Psi(y) := \min\{1^T s \,|\, g(x,y) \leq s,\, s \geq 0\}$ to (21)-(24).)

Note that $\Phi(y)$, though convex, is nonlinear even in case f and g are. For example, the toy continuous problem $\min\{x \mid x \geq y,\, x \geq -y, y \in [-1,1]\}$ leads to the master $\min\{\Phi(y) \mid y \in [-1,1]\}$ where $\Phi(y) = |y|$ is clearly convex but nonlinear, and attains its minimum in $y = 0$, i.e., in a point that is not of vertex of $S = [-1,1]$.

Master problem (21)-(24) is therefore a convex MINLP that can be solved by an LP-based branch-and-cut approach where the integrality requirement on y is relaxed and $\Phi(y)$ is approximated by linear cuts to be generated on the fly and added to the current LP-relaxation; see [8] for a recent introduction to modern MINLP solution techniques. A crucial point is therefore the efficient generation of the approximation cuts, whose correctness exploits the convexity assumption. To this end, consider a (possibly noninteger) solution y^* of the LP relaxation of the current master. Because of convexity, $\Phi(y)$ can be underestimated by a supporting hyperplane at y^*, so we can write the following linear cut

$$w \geq \Phi(y) \geq \Phi(y^*) + \xi(y^*)^T (y - y^*) \tag{25}$$

Here $\xi(y^*)$ denotes a subgradient of Φ in y^* that can be computed (if constraint qualifications hold) as

$$\xi(y^*) = \nabla_y f(x^*, y^*) + u^* \nabla_y g(x^*, y^*) \tag{26}$$

where x^* and u^* are optimal primal and (Lagrangian) dual solutions of the convex problem obtained from (18)-(20) by replacing y with the given y^*, and ∇_y is the gradient operator with respect to the y variables; see Geoffrion [16] for details.

The above formula involves the computation of partial derivatives of f and g with respect to the y_j's, so it is problem specific and sometimes cumbersome to apply. We next introduce a very simple reformulation that makes its implementation straightforward. By definition, $\Phi(y)$ can be computed by solving the convex problem (referred to as the *slave problem* in the literature)

$$\Phi(y) = \min f(x, q) \tag{27}$$

$$\text{s.t. } g(x, q) \leq 0 \tag{28}$$

$$y - q = 0 \tag{29}$$

The variable-fixing equation (29) is meant to be imposed as $y \leq q \leq y$ by just modifying the lower and upper bounds on the q variables, so it can be handled very efficiently by the solver in a preprocessing phase when y is given.

By construction, y only appears in the trivial constraint (29), hence the subgradient in (26) is just

$$\xi(y^*) = r^* \tag{30}$$

where r^* is an optimal dual vector associated to (29), e.g., the vector of reduced costs when this constraint is imposed as $y^* \leq q \leq y^*$. This leads to the (generalized) Benders cut

$$w \geq \Phi(y^*) + \sum_{j=1}^{n} r_j^*(y_j - y_j^*) \tag{31}$$

where each (possibly negative) reduced cost r_j^* defines a lower bound on the increase of the objective function $\Phi(y^*)$ when y_j^* increases—note that each reduced cost r_j^* is the optimal dual variable associated with the bound constraint $q_j \geq y_j^*$ (if $r_j^* \geq 0$) or $q_j \leq y_j^*$ (if $r_j^* < 0$).

3.2. Benders for CFL

Following Benders general recipe, all continuous variables are projected out of the model, and the overall objective function is replaced by a single continuous variable w. The resulting master problem is then given by

$$\min w \tag{32}$$

$$\text{s.t. } w \geq \Phi(y) \tag{33}$$

$$\sum_{j \in J} s_j y_j \geq \sum_{i \in I} d_i \tag{34}$$

$$y_j \in \{0, 1\} \qquad \forall j \in J \tag{35}$$

where $\Phi(y)$ in the convex function expressing the solution cost of the best-possible assignment compatible with the given (possibly noninteger) $y \in [0,1]^J$. Of course, for congested CFL one needs to extend the master problem by the p-median constraint (8).

Recall that constraint (34) guarantees that the overall capacity of open facilities is sufficient to accommodate the overall customer demand. This latter constraint plays a very important role in our model, as it acts as a global feasibility cut ensuring that any solution y of the master admits a feasible x in the original model, hence $\Phi(y)$ is well defined for any master (possibly noninteger) solution y. As a consequence, no so-called *Benders feasibility cuts* are needed in our setting.

3.3. Benders cut separation

Within a modern MIP solver, cuts are possibly separated at every node of the branching tree. Besides internal general-purpose cuts, the user can design his/her own separation function that receives a (possibly noninteger) master solution (y^*, w^*) on input, and tries to generate a violated cut. In our setting, we are interested in generating violated Benders cuts of the form (31) derived from the convex relaxation obtained by removing integrality constraints on y variables. Separation of Benders cuts is therefore performed for both, fractional and integer values of y^*. To this end, for the given y^* the convex slave problem (27)-(29) is solved for $y = y^*$ to get the optimal value $\Phi(y^*)$ and the associated reduced cost vector r^*. Note that the solution of the latter problem can be rather time consuming, as it does not decompose into smaller subproblems as it happens in typical application of Benders decomposition. Nevertheless, as y is fixed, the explicit constraints $x_{ij} \le y_j$ are automatically preprocessed and become just implicit upper bounds on the x variables. In addition, for congested CFL the SOC constraints $v_j^2 \le z_j y_j$ become regular quadratic constraints. As a consequence, the solution of the slave subproblem can be orders of magnitude faster than the solution of the convex relaxation where y is not fixed and only the integrality requirement is relaxed, making the Benders approach appealing.

We observe that the solution of the slave (27)-(29) for $y = y^*$ could take advantage of the specific structure of the CFL problem. In particular, for linear CFL the slave is in fact a transportation problem that could be solved by a specialized algorithm. According to our experience, however, using a general-purpose LP solver such as the dual simplex method has the advantage of a better warm-start mechanism, so after some testing we decided not to use any specialized code—thus making the overall implementation simpler.

3.4. Benders cuts selection

Many authors pointed out that the slave problem is typically dual degenerate, which means that many optimal dual solutions exist (see, e.g., [25, 26, 28, 30]). Since each optimal dual solution leads to different reduced costs and hence to a different Benders cut, the question on how to choose the "best" Benders cuts to be inserted into the master is a key of an efficient implementation. In our implementation strategy, we do not insist on generating Pareto optimal cuts (like e.g., in [25]) but just apply the very basic method of the previous subsection. For linear CFL only, we apply the following simple recomputation of the optimal reduced costs (and hence of the Benders cut coefficients) that according to our experience produced more stable cuts.

Let u^* be the vector of dual variables associated to assignment constraints (2) found by the LP solver applied to the slave subproblem, i.e., to the original model (1)-(6) with the variable-fixing constraint $y^* \le y \le y^*$. Our idea is to fix u^* and to recompute optimal reduced costs by solving a series of continuous knapsack problems. Recall that $\Phi(y)$ denotes the optimal value of (1)-(6) as a function of y. As already observed by many authors (see, e.g. [9]), by Lagrangian duality we can write

$$\Phi(y) \ge \sum_{i \in I} u_i^* + \min_{0 \le x \le 1} \left\{ \sum_{j \in J} f_j y_j + \sum_{j \in J} \sum_{i \in I} (d_i c_{ij} - u_i^*) x_{ij} \mid (3), (4) \right\} = \sum_{i \in I} u_i^* + \sum_{j \in J} (f_j + \mathrm{KP}_{u^*}^j)\, y_j$$

where the $|J|$ independent continuous knapsack problems

$$\mathrm{KP}_{u^*}^j := \min\{\sum_i (d_i c_{ij} - u_i^*) z_i \mid \sum_{i \in I} d_i z_i \le s_j,\, 0 \le z \le 1\}$$

can be solved very efficiently (and in a numerically very clean way) by a simple sorting algorithm. This leads to the Benders cut

$$w \geq \sum_{i \in I} u_i^* + \sum_{j \in J} (f_j + \mathrm{KP}^j_{u^*})\, y_j$$

Note that the above procedure would work for any vector u of Lagrangian multipliers associated with (2), though the maximum cut violation is achieved for an optimal dual vector u^* for the point y^* to be separated.

3.5. Initial cuts before branch-and-cut

We have seen that the master problem (21)-(24) calls for the minimization of a convex function $\Phi(y)$ over a nonempty polytope S, under the integrality requirement (24). Branch-and-cut solution methods relax the integrality condition and use enumeration to enforce it. A crucial step is how to solve effectively, by cutting planes, the convex problem $\min_{y \in S} \Phi(y)$ arising at the root node. As a matter of fact, in many cases reported in the literature (and perhaps in many more not reported ones) such a solution approach turns out to be so slow that it makes the overall Benders method simply impractical.

As already discussed, many authors attributed slow convergence to dual degeneracy, hence focused on how to select a "best possible" Benders cut for the given y^* at hand. According to our experience, however, cut selection is important but the cut loop strategy adopted at the root node is the real culprit of inefficiency. Following Kelley's recipe [21], which is standard in MIP solvers, at each cut loop iteration one generates one or more cuts that are violated by the current LP-optimal solution y^*, adds them to the current relaxation, and reoptimizes it to get the new optimal solution y^* to cut at the next iteration. This approach works well in many cases, but it is known to be extremely inefficient when Φ and S have a shape that leads to a zig-zagging y^* trajectory, as it happens e.g. in Lagrangian dual optimization or in column generation [29]. In these cases, "stabilized" approaches such as the bundle method [24] are known to outperform Kelley's one by a large margin. Thus, the implementation of stabilized cutting plane (at least) at the root node is expected to be of crucial importance in Benders decomposition, in particular when a single cut is separated for each y^* as it happens in our CFL implementation.

Following our previous proposal for uncapacitated facility location [14], we did not implement a bundle method but a simple in-out variant very much in the spirit of [5, 13, 26]. The resulting cutting plane procedure is applied before starting the branch-and-cut, i.e., before the root node. It is aimed at quickly determining a hopefully small set of Benders cuts that brings the master LP relaxation value as close as possible to the "real optimal value" $\min_{y \in S} \Phi(y)$.

At each cut loop iteration, we have two points in the y space: the optimal solution y^* of the current master LP (as in Kelley's method), and a stabilizing point $\widetilde{y}$ inside the convex feasible set S. This point is initialized by solving the convex problem $\max\{\sum_{j \in J} y_j \mid y \in S\}$ through an internal point method (barrier without crossover).

At each iteration, we move $\widetilde{y}$ towards y^* by setting $\widetilde{y} = \alpha\,\widetilde{y} + (1-\alpha)\,y^*$ and then apply our Benders-cut separator to the "intermediate point" $\lambda\,y^* + (1-\lambda)\,\widetilde{y}$, where parameters $\lambda \in (0,1]$ and $\alpha \in (0,1]$ are discussed later on. The generated Benders cut is statically added to the current master LP. After 5 consecutive iterations in which the LP bound does not improve, parameter λ is reset to 1 (so we are back to Kelley) and the cut loop continues. After 5 more consecutive iterations without improvement, the procedure is aborted, the LP is solved once again and all cuts with a positive slack are removed. To speedup computation, slack cuts in the current master LP are also removed at every 5-th iteration.

Our computational experience with CFL (both linear and congested) confirms the findings of [14], and shows that the above stabilization approach—though very simple—works well and quickly produces lower bounds very close to the best-possible one.

As to the choice of parameters α and λ, for congested CFL we adopted the values originally proposed in [14], namely $\alpha = 0.5$ and $\lambda = 0.2$. For linear CFL, instead, we noticed that an overall speedup can be obtained by a more conservative approach producing a sequence of very similar points to be separated. This is because, for the linear case, Benders cuts are generated by solving the slave subproblem through the dual simplex method, each time starting from the optimal LP base of the previous separation. So, the more similar the points to separate, the faster the cut generation. Hence we defined $\alpha = 0.9$ and $\lambda = 0.1$ to increase the attraction grip of the internal point $\widetilde{y}$. A similar approach is instead not worthwhile for the congested CFL, where the slave subproblem is nonlinear and is solved by using a barrier method that starts from scratch at each call.

3.6. Enforcing slave separability by blurring

To mitigate the use of the time-consuming Benders cut separation that requires the solution of a single but large slave problem, for the linear case we tested the following CFL-specific strategy that tries to enforce slave separability in a heuristic way. To this end, we observe that slave decomposition into smaller subproblems is inhibited by the presence of the capacity constraints (4): if we temporarily relax them, we get an uncapacitated problem for which Benders cuts can be derived very efficiently by using simple ad-hoc algorithms [14]. So we can think of a two-level cut strategy where "blurred" Benders cuts are quickly generated from the uncapacitated model, and the generation of the capacitated Benders cuts is activated only as a last resort, i.e., when the given solution y^* is integer and is going to update the incumbent—in which case we need the "real Benders cut" to assess the correct value of $\Phi(y^*)$. Of course, the approach makes sense only for those instances where the capacity constraints (though non-redundant) are not very tight, meaning that only a few "real Benders cuts" need to be generated. According to our computational analysis, the approach is in fact very successful for just a few instances in our testbed, while it does not pay off in general.

4. Implementation details

In this section we describe some implementation details that play an important role in the design of an effective code. The description is based on the actual MIP solver we used, namely IBM ILOG Cplex 12.6.1, but it extends easily to other solvers.

4.1. Multi-threading

Modern MIP solvers exploit multi-threading, so it is important not to lose this important feature when embedding user-specific functions such as Benders-cut generation. Cplex's default is quite conservative, and it switches to single-thread as soon as the user installs "potentially risky" callbacks. In our implementation, we reset the number of threads to the one provided by the hardware, say NT ($NT = 4$ for our computer), and select the opportunistic mode (`CPX_PARAM_PARALLELMODE`=-1) to better exploit parallelism. Also, to be thread-safe, at startup we create NT copies (clones) of the slave problem (27)-(29), and let each thread work on its own copy to produce its Benders cuts. As the clone is not destroyed but just modified at each separation call, warm-start is automatically applied without the need of explicitly saving/loading optimal LP bases (this of course applies to the linear case only). As to the cut initialization before branch-and-cut described in Subsection 3.5, it is intrinsically sequential—in particular, for the linear case as the dual simplex method does not exploit parallelization. So we decided to run NT multiple copies of it, in parallel, with slightly modified α and λ parameters to produce different search paths. When the first such run terminates, all other runs are aborted, and all the generated Benders cuts are statically added to the master which is solved once again to obtain the final lower bound. Finally, slack cuts are purged to avoid overloading the initial master.

4.2. Tree search strategy and heuristics

As already observed, in our setting the slave problem does not decompose into smaller pieces, so each Benders cut separation is quite time consuming. As a matter of fact, for some large instances more than 90% of the overall computing time can be spent within the cut separation function. This is a very unusual setting for a general-purpose MIP solver, so one may expect that its default parameter tuning is inappropriate. In particular, it makes sense to use very aggressive parameters for internal heuristics, and for exploiting time-consuming strategies such as full strong branching and best-bound search that are not in the Cplex's default. To be more specific, in our implementation we set to 2 (aggressive) the level of all Cplex's internal cuts. We also selected the full-strong branching strategy (`CPX_PARAM_VARSEL`=3) and the pure best-bound search strategy (`CPX_PARAM_NODESEL`=1, `CPX_PARAM_BBINTERVAL`=1, and `CPX_PARAM_BTTOL`=0.0). As to heuristics, we decided to apply RINS heuristic at every node (`CPX_PARAM_RINSHEUR`=1) to feed our Benders-cut separator with low-cost integer solutions.

4.3. Tailing off

At each node of the branch-and-cut tree, we do not allow for more than 20 consecutive calls of the Benders separation function (100 for the root node). This is obtained by maintaing a counter K (one for each thread) which is reset to 0 when a node is first processed, and is increased by 1 before invoking the separation function. To limit computing time, we apply the following give-up mechanism. After having solved the current slave problem and possibly added the generated Benders cut to the master, we retrieve the primal optimal value UB (say) of the slave problem that was just solved. As UB is an upper bound on the best bound that can be obtained at the current node by Benders cuts, we set $K = +\infty$ in case $K \geq 2$ and UB is strictly smaller than the value of the current incumbent, meaning that we have no hope to prune the current node (unless the incumbent is updated, which is a very rare event). This policy turned out to speedup the overall computation, though we are aware of the fact that it prevents the generation of Benders cuts that could be useful to improve the bound in other nodes. Even with the above setting, it may happen that the fraction of computing time spent for separation is excessive, so we skip the separation of fractional solutions (within the so-called `usercut callback`) when this fraction exceeds 50% of the current computing time.

To ensure correctness, integer solutions that can update the incumbent are always separated (within the so-called `lazyconstraint callback`) so their actual cost is recomputed exactly.

4.4. Restart

By design, our branch-and-cut scheme learns a lot of information (notably: Benders cuts and heuristic solutions) during execution, and in particular after the root node. On the other hand, the root node plays a special role for what concerns preprocessing, internal cut generation and initial heuristics, so the more information is available at the root node the better. In particular, it is very important to start with a very good heuristic solution and with a rich family of Benders cuts, to favor root-node variable fixing and generation of internal cuts. As already mentioned, to escape Kelley's cutting plane scheme we implemented our own stabilized cutting plane procedure before branch-and-cut, so the generated Benders cuts are immediately available at the root node. In addition, we found useful to increase the information available at the root node by a simple restart mechanism. Namely, before entering the final branch-and-cut run we stop the execution right after the root node (by setting `CPX_PARAM_NODELIM`=1), add the generated Benders cuts (saved in our own data structure) as static cuts to the master model, update the incumbent, and repeat. This restart mechanism is applied twice before entering the final run (`CPX_PARAM_NODELIM`=∞).

Internal parameters. For the master problem, we used the following numerical tolerances: `CPX_PARAM_EPINT`= 0.0, `CPX_PARAM_EPRHS`=1e-7, and `CPX_PARAM_EPGAP`=1e-6. To be consistent, for the slave we need an increased precision so we set `CPX_PARAM_EPRHS = CPX_PARAM_EPGAP`=1e-9. In addition, when solving the slave problem for the congested CFL we had to set `CPX_PARAM_CALCQCPDUALS`=2 to skip some preprocessing reductions that would inhibit the computation of the optimal dual solution (and reduced costs) needed to derive the Benders cut.

5. Computational study

In this section we report on our computational experience on four classes of CFL instances from the recent literature. When it comes to linear CFL, we compare the performance of our Benders approach against state-of-the-art exact and heuristic methods published in the last few years. For congested CFL, we compare our generalized Benders decomposition framework with the perspective reformulation solved by the commercial solver IBM ILOG Cplex 12.6.1.

Our algorithm has been implemented in C, and derived on top of IBM ILOG Cplex 12.6.1 callable library. The computational study is conducted on a cluster of identical machines each consisting of an Intel Xeon E3-1220V2 @ 3.1GHz, with 16GB of RAM each. This processor was launched by Intel in 2012 and is credited for 1,892 Mflop/s in the Linpack benchmark report of Dongarra [11]. Computing times reported are wall-clock seconds and refer to 4-thread runs. The default timelimit is set to 50,000 seconds, unless stated otherwise. In our Benders implementation, all Cplex parameters not mentioned in the previous section were left at their default values. For Cplex stand-alone runs, e.g., the perspective reformulation, we set `CPX_PARAM_EPGAP`=1e-6 and left all other parameters at their default values.

The following abbreviations are used in the tables presented throughout this section: the total computing time in wall-clock seconds ($t[s]$), the time needed to solve the LP-relaxation at the root node ($t_r[s]$), the total number of enumerated branch-and-bound nodes (*nodes*), the percentage gap at the root node ($g_r[\%]$), and the percentage gap obtained after reaching the timelimit ($g[\%]$). The gaps are computed as $100(z^{UB} - z^{LB})/z^{UB}$, where z^{UB} is the optimal or best objective value found by our Benders approach and z^{LB} is the appropriate lower bound for the gap. This calculation of gaps is consistent with the one from the recent literature, see, e.g., [18].

5.1. Benchmark instances

The following four sets of benchmark instances are considered in our study:

- `cap*`: This is a subset of non-trivial instances from the OR-Library [4]. The set consists of 12 instances with $|I| = 1,000$ and $|J| = 100$. Note that the remaining (smaller) instances available in this library are left out, since they can be easily solved by modern MIP solvers in fractions of a second.

- `GK`: These instances have been generated by Görtz and Klose [17, 23] and tested by the branch-and-price and Lagrangian-based branch-and-bound from [23] and [17], respectively. Instances are generated following the procedure proposed by Cornuéjols et al. [9]: customers and potential facilities are uniformly randomly placed in a unit square. Euclidean distances multiplied by 10 represent allocation costs per unit of flow. Facility opening costs are generated as $f_j = U[0,90]+U[100,110]\sqrt{s_j}$, where $U[a,b]$ stands for a uniformly distributed random number from $[a,b)$. Customer demands and capacities of facilities are drawn uniformly at random from $[5,35]$ and $[10,160]$, respectively. Finally, the following scaling factor is used to rescale the capacities and create different subclasses of instances:

$$r = \sum_{j\in J} s_j / \sum_{i\in I} d_i.$$

Smaller instances from this set comprise a group of 75 instances with $|J| \times |I| \in \{100 \times 100, 100 \times 200, 100 \times 500, 200 \times 200, 200 \times 500\}$ and consider values of $r \in \{3,5,10\}$. Larger instances from this set (introduced later in [17]) comprise a group of 120 instances with $|J| \times |I| \in \{300 \times 300, 300 \times 1500, 500 \times 500, 600 \times 1,500, 700 \times 700, 1,000 \times 1,000\}$ and the values of $r \in \{5,10,15,20\}$. Instance generator has been provided by [17] and is available online (`http://home.imf.au.dk/aklose/CFLP/generator.tgz`).

- `i*`: These instances were used to test the branch-and-cut-and-price algorithm by Avella and Boccia [2] and are available at `http://www.ing.unisannio.it/boccia/CFLP.htm`. They are generated following the same procedure proposed in [9]: the set consists of 100 instances of size $|J| \times |I| \in \{300 \times 300, 300 \times 1500, 500 \times 500, 700 \times 700, 1,000 \times 1,000\}$ and $r \in \{5,10,15,20\}$.

- `p*`: These instances have been addressed by the heuristic algorithms of Avella et al. [3] and Guastaroba and Speranza [18]. They consist of three groups, denoted by test bed A, B and C, containing 150, 145 and 150 instances, respectively. All test beds are with $|J| \times |I| \in \{800 \times 4,400, 1,000 \times 1,000, 1,000 \times 4,000, 1,200 \times 3,000, 2,000 \times 2,000\}$ and $r \in \{1.1, 1.5, 2, 3, 5, 10\}$. The only difference between the test beds concerns allocation costs: these are two orders of magnitude smaller than facility opening costs for test bed A, one order of magnitude smaller than facility opening costs for test bed B, and of the same order as the facility opening costs for test bed C. The first two test beds are publicly available at `http://wpage.unina.it/sforza/test/`, the last one at `http://or-brescia.unibs.it/instances/instances_clfp`. To the best of our knowledge, these instances were not solved to optimality by any of the previously proposed approaches. Only heuristic results from Avella et al. [3] and Guastaroba and Speranza [18] are available. Instances `p*` contain up to 2,000 facilities and 4,000 customers, hence the underlying compact models consist of millions of variables and constraints. This is the first attempt to solve the largest instances of this class to provable optimality.

5.2. Results for classical CFL

Results on `cap*` *instances*

For this set of instances, aside from giving results for our Benders approach, we also present results obtained by using our "blurred" option described in Section 3. Recall that the latter approach consists of generating *uncapacitated* Benders cuts in the `user-cutcallback` and in the initial in-out procedure. To ensure feasibility, capacitated Benders cuts are only separated in the `lazy-cutcallback`. Table 1 compares the computing times and the number of branch-and-bound nodes for the following four approaches: compact model solved by Cplex, Benders, "blurred" Benders, and the approach by Görtz and Klose [17]. We first observe that Benders is one order of magnitude faster than Cplex, and that "blurred" Benders outperforms Cplex by an even larger margin, with speedup factors between 10 and 90. This speedup is quite remarkable, in particular after noticing that "blurred" Benders sometimes requires ten times more branch-and-bound nodes than the remaining three approaches. There are two factors for the success of the "blurred" Benders in case of `cap*` instances. First, uncapacitated Benders cuts are separated in a combinatorial fashion, thanks to the separability of the slave, see [14], and the time-consuming separation of the unseparable slave is performed only to cut off infeasible integer solutions. Second, the capacity requirements on facilities are, in case of `cap*` instances, not very tight in the underlying optimal solutions. Hence, uncapacitated Benders cuts successfully guide the branch-and-cut process, and allow for a faster exploration of the search space. Unfortunately, the other instances considered in our study do not exhibit the same structure, and the success of "blurring" was not confirmed in the remaining cases.

In Table 1 we also give the original computing times reported in Görtz and Klose [17], that refers to an Intel Pentium D930 @ 3Ghz and IBM ILOG Cplex 8.0 (the latter being used as an LP solver, so it is not much slower than our 12.6.1 version). According to [17], this hardware is comparable with an Intel Pentium 4 @ 3.06GHz ranked at 1,414 Mflop/s in the Linpack benchmark report of Dongarra [11], while our own hardware is ranked 1,892 Mflop/s. One can therefore conclude that Benders and Lagrangian-based branch-and-bound are two competitive approaches for `cap*`.

Results on `i*` *instances*

This set of instances has been previously tested by the two best exact methods from the literature: B&C&P by Avella and Boccia [2] and B&B by Görtz and Klose [17]. In Table 2 we compare our Benders code with these two approaches, and with the compact model solved by IBM ILOG Cplex 12.6.1. Instances are grouped according to their size and the value of r, each group thus contains five instances. Computing times and number of branch-and-bound nodes, averaged per group, are reported. We again provide the computing times as originally reported in the respective papers on slower systems: results from Avella and Boccia [2] were obtained on an Intel Pentium IV @ 1.7GHz (credited for 796 Mflop/s in Dongarra [11]) and IBM ILOG Cplex 8.1, while those for Görtz and Klose [17] were obtained on an Intel Pentium D930 @ 3GHz

Table 1: Comparing the computing times in seconds and the number of branch-and-bound nodes for two Benders decomposition settings, the compact model solved by IBM ILOG Cplex 12.6.1 and the approach by Görtz and Klose [17]. Computing times for [17] refer to a slower hardware and to an earlier version of Cplex. Column *nodes* gives the number of branch-and-bound nodes, and $t[s]$ the computing time in seconds.

			Cplex		Benders		"Blurred" Benders		B&B [17]	
name	$\|J\|$	$\|I\|$	nodes	$t[s]$	nodes	$t[s]$	nodes	$t[s]$	nodes	$t[s]$
capa1	100	1000	5	28.96	8	10.30	71	0.90	2	2.91
capa2	100	1000	0	12.50	2	6.52	31	0.96	7	2.89
capa3	100	1000	7	24.36	5	3.96	8	0.68	9	2.18
capa4	100	1000	0	6.76	0	1.34	0	0.65	1	0.84
capb1	100	1000	0	7.06	0	8.10	5	0.57	1	1.66
capb2	100	1000	23	108.67	33	10.34	344	4.20	27	11.06
capb3	100	1000	24	110.38	27	7.99	197	2.29	29	11.49
capb4	100	1000	15	54.09	6	3.42	61	0.66	17	4.32
capc1	100	1000	23	28.77	10	5.06	49	0.87	9	3.40
capc2	100	1000	55	66.80	33	5.55	199	1.55	50	12.63
capc3	100	1000	9	27.14	5	2.87	15	0.53	11	6.01
capc4	100	1000	5	11.68	0	1.77	8	0.52	5	2.40

(credited for 1,414 Mflop/s) and IBM ILOG Cplex 8.0. As already mentioned, our own results have been obtained on an Intel Xeon E3-1220V2 @ 3.1GHz (credited for 1,892 Mflop/s) and IBM ILOG Cplex 12.6.1.

Surprisingly, IBM ILOG Cplex 12.6.1 solves all the `i*` instances to optimality. The remaining three methods have difficulties with the largest ones of size $1,000 \times 1,000$. There are two instances with $r = 15$ that were not solved by Avella and Boccia [2] within their imposed timelimit of 100,000 seconds. One instance with $r = 5$ was unsolved by Görtz and Klose [17]. Similarly, our method does not manage to prove the optimality for a single instance with $r = 15$ (although its final gap remains below 0.08%). Following [17], average computing times reported in Table 2 do not take in consideration these unsolved cases. Comparing Benders and Cplex with respect to the average computing times and for instances of different sizes, we observe that Benders is much faster than Cplex in all cases, except for size $1,000 \times 1,000$ and $r = 15$ where Benders and Cplex are comparable. We want to point out that it would be possible to tune our code to speedup convergence for some special cases, but this was not the intention of our computational study. Instead, we prefer to provide a stable and robust implementation, despite the unsatisfactory performance in a very few cases.

The overall results for `i*` indicate that Benders often outperforms the B&C&P, and that it is competitive with Lagrangian-based B&B. The latter method is often faster than Benders, despite the fact that it sometimes requires 10 times more branch-and-bound nodes for proving optimality. Indeed, B&B is faster in exploring the search space because Lagrangian relaxation solved at each node of the B&B tree boils down to a simple knapsack problem (of a small size) that can be efficiently solved in pseudo-polynomial time, whereas Benders suffers from the non-separability of the slave.

Finally, it is interesting that B&B and Benders are complementary, when it comes to efficiency in solving the largest among these instances. Whereas the most difficult among $1,000 \times 1,000$ instances for B&B are those with tight capacities ($r = 5$), they are the easiest ones for Benders, and vice versa, those with rather lose capacities ($r = 15$) are the easiest for B&B and the most difficult ones for Benders. Instances with very lose capacities ($r = 20$) are consistently the easiest ones for all four considered approaches.

Results on `GK` *instances*

For this family of instances, we first demonstrate the power of decomposition by comparing the computing times required for solving the root node relaxation by Benders and Cplex. Figure 1a shows a performance profile over 120 instances from this family: a point with coordinates (x, y) in this plot indicates that the computing time of y instances was $\leq x$ seconds. Whereas for solving the LP-relaxation at the root node Benders requires at most 170 seconds, for 13 instances Cplex requires more than one hour, and for 5 of them even more than two hours. For about 50% of instances, Cplex and Benders at the root node are competitive, but for the largest ones from this family with $|J| \times |I| \in \{700 \times 700, 600 \times 1,500, 1,000 \times 1,000\}$ Benders can clearly draw an advantage from the decomposition, by projecting out a large number of allocation variables. Accompanying Figure 1b reports performance profiles for the gaps obtained by Cplex and Benders at the root node (denoted by Cplex-root and Benders-root, respectively). We observe that the obtained root gaps are almost identical, which confirms the effectiveness of the initial cut selection strategy described in Subsection 3.5. Notice that these instances exhibit extremely tight LP-relaxation gaps—more than 50% of them have LP-relaxation gaps $\leq 0.5\%$, and the largest LP-relaxation gap is $< 1\%$.

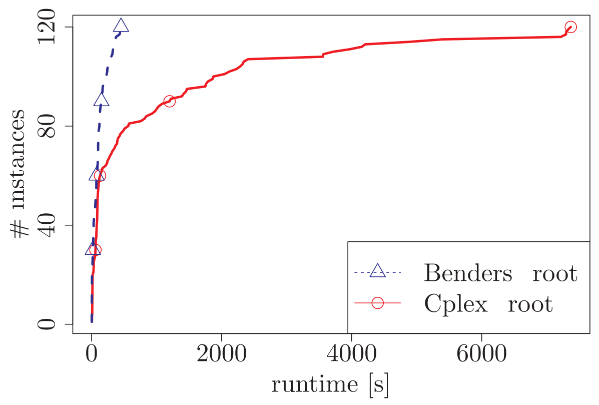
Besides comparing the root node relaxations, we also ran Benders and Cplex with a timelimit of 3,600 seconds; the obtained percentage gaps are also reported in Figure 1b. For 75% of the instances, after one hour of computing time, Benders delivers solutions with optimality gaps $\leq 0.2\%$, whereas Cplex does not manage to solve even the LP-relaxation for 13 out of 120 cases.

We recall that these instances were introduced by Görtz and Klose [17] who report optimal solutions for all of them, obtained within a timelimit of 50,000 seconds. After running Benders with a timelimit of 50,000 seconds we did not manage to solve all the instances from this family to provable optimality. However, the obtained gaps were extremely small: over all 120 instances, the average and the maximum gap that we obtained are 0.08% and 0.4%, respectively. The main difficulty of the Benders approach remains closing the last per-mills of the gap in most of the cases. More precisely, from 120 instances of this set, 38 instances are

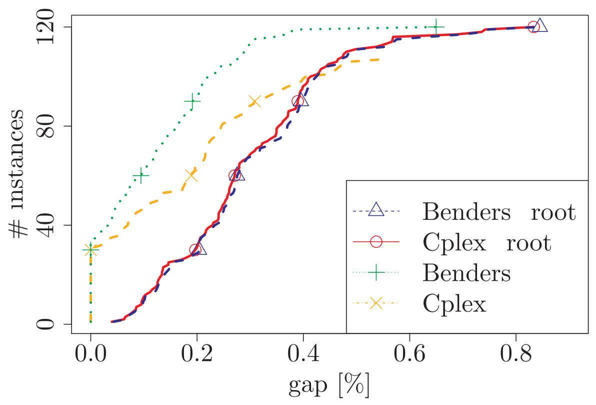
Table 2: Comparing Benders with the state-of-the-art approaches and IBM ILOG Cplex 12.6.1 on the set of `i*` instances. Column *nodes* gives the average number of branch-and-bound nodes, and $t[s]$ the average computing time in seconds per group. Computing times for [17] and [2] refer to slower computers.

		B&B[17]		B&C&P[2]		Benders		Cplex	
$\|J\|$	$\|I\|$	nodes	$t[s]$	nodes	$t[s]$	nodes	$t[s]$	nodes	$t[s]$
$r = 5$									
300	300	657	19.00	436	588.48	112	7.05	506	32.15
500	500	2867	209.33	1258	3098.64	261	38.56	1289	458.90
700	700	28709	3717.48	1696	9084.24	1061	182.65	1631	1682.41
1000	1000	46893	15635.69	4123	37439.57	9823	2636.25	4743	10152.00
300	1500	547	235.71	32	1672.66	159	49.47	76	56.32
$r = 10$									
300	300	535	12.53	131	402.61	139	12.05	314	45.12
500	500	2431	120.56	476	1824.64	342	85.10	659	632.04
700	700	8729	899.63	808	11064.76	1511	721.01	2241	3699.03
1000	1000	26267	5096.53	3370	60865.83	5361	6053.80	4468	18705.30
300	1500	19	27.45	9	852.97	16	10.75	14	38.74
$r = 15$									
300	300	239	4.82	52	178.47	55	5.82	110	28.24
500	500	358	18.69	65	669.44	146	34.64	166	185.24
700	700	8423	674.53	356	4991.52	1004	639.26	718	1328.44
1000	1000	47956	7323.29	2070	65974.26	4380	14182.78	3173	13024.17
300	1500	9	23.25	2	408.66	9	9.30	5	29.52
$r = 20$									
300	300	60	1.75	18	142.42	27	3.35	35	18.07
500	500	264	13.13	47	481.54	85	19.65	116	105.16
700	700	221	22.83	40	1066.26	190	75.45	123	325.01
1000	1000	2362	328.71	354	9514.38	859	1469.08	638	2956.70
300	1500	3	14.82	2	293.97	2	5.96	1	23.96

solved to optimality, for 31 the gap is $< 0.05\%$, for 40 the gap is between 0.05% and 0.2% and for only 11 of them, the gap is between 0.2% and 0.4%.



(a) Computing times at the root node.



(b) Relative gaps to the best known upper bounds.

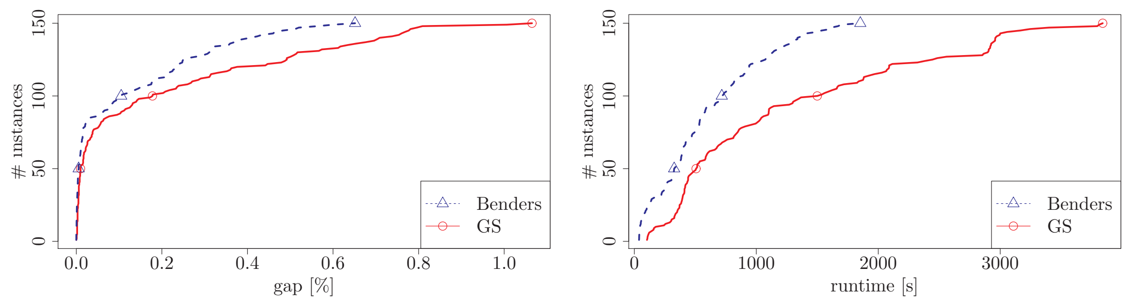
Figure 1: `GK` instances: Comparing the performance of Benders and IBM ILOG Cplex 12.6.1 at the root node, and after a timelimit of 3600 seconds.

Results on `p*` instances

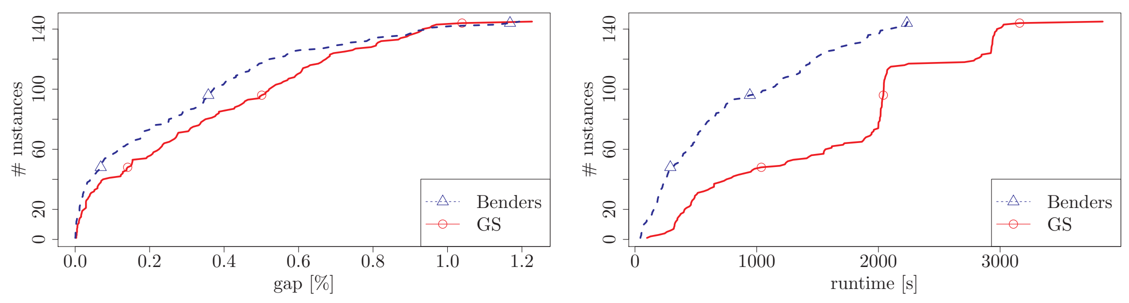
This set represents the family of largest benchmark instances, with up to 2,000 facilities and 4,000 customers, that, prior to this work, were not tackled by exact approaches. For this family of 445 instances, Benders was able to prove optimality in 210 cases (within our default timelimit of 50,000 seconds).

In the following, we first compare Benders with the kernel search heuristic of Guastaroba and Speranza [18] (denoted by GS in the following), that was shown to significantly improve the previously published heuristic results by Avella et al. [3]. For that purpose, we stop Benders after enumerating 10 branch-and-bound nodes and report the obtained percentage gaps and computing times. Figure 2 shows performance profiles considering the percentage gaps and the computing times, separated by test beds A, B and C. Results for [18] have been obtained using an Intel Xeon @ 2.27GHz (which is about 30% slower than our computer) and IBM ILOG Cplex 12.2. One observes that Benders always produces smaller gaps in shorter computing times for all three test beds. The difference is especially pronounced for test bed C: the worst gaps obtained by Benders are $\leq 2\%$ (and in 80% of all instances even $\leq 1\%$), whereas GS gaps are $> 2\%$ for 70% of all instances, and can be as large as 8%.

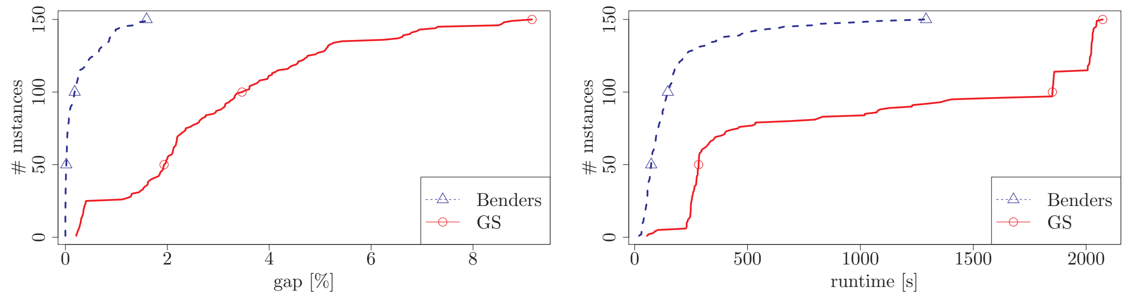
The idea of enumerating only 10 B&B nodes by Benders was mainly to demonstrate its ability to serve as an efficient heuristic outperforming other state-of-the-art CFL heuristics. Even as a heuristic, Benders manages to prove optimality for 5 (out of 150) instances of test bed A, for 0 (out of 145) instances of test bed B and for 15 (out of 150) instances of test bed C. By increasing the timelimit to 3,600 seconds (50,000 seconds, respectively), Benders proves optimality for 86 (97) instances from test bed A, 22 (40) from test bed B, and 68 (73) for test bed C.



(a) Testbed A



(b) Testbed B



(c) Testbed C

Figure 2: p$*$ instances: Comparing the performance of Benders and the kernel search GS heuristic of Guastaroba and Speranza [18]. Benders has been stopped after ten branch-and-bound nodes.

5.3. Results for congested CFL

To assess the efficacy and computational limitations of generalized Benders decomposition for congested CFL, we consider the group of `i*` instances introduced above with up to 1,000 facilities and 1,000 customers. The number of open facilities p, which is specified as part of the input, is defined as $p = \lfloor \pi |J| \rfloor$, where $\pi \in \{0.4, 0.6, 0.8\}$. Since no previous computational studies are available, we focus on comparing the performance of Benders against the perspective reformulation (cf. Section 2.2) which is solved by IBM ILOG Cplex 12.6.1

using second order cone constraints. Tables 3 and 4 summarize the obtained results. Each line corresponds to the first instance from the i* group with a fixed scaling ratio $r \in \{5, 10, 15, 20\}$ (namely, instances i*_1, i*_6, i*_11, and i*_16) and a fixed value of π. For each instance we report the optimal solution value, the percentage gap and the computing time at the root node, plus the final percentage gap and the overall computing time. Runs for which the timelimit of 50,000 seconds is reached, are marked with TL in the computing-time column. Note that some runs for the perspective reformulation were aborted due to memory problems before any meaningful output was produced, in which case "–" entries are shown in the tables.

Recall that the relative tolerance CPX_PARAM_EPGAP for the master in Benders and for Cplex is set to 1e-6 in our experiments, i.e., as soon as the percentage gap between the lower and upper bound is $< 0.0001\%$, the corresponding method terminates and reports the obtained solution as optimal. The OPT values reported in the tables are those obtained by Benders.

The obtained results indicate that Benders beats the perspective reformulation by a large margin: out of the 60 considered instances, Benders solves 45 to optimality, and for the remaining 15 it reaches the timelimit with the final gaps $\leq 0.3\%$. On the contrary, the perspective reformulation delivers optimal solutions for only 21 cases, while for 23 instances the timelimit is reached with final gaps as large as 35%, and for 16 instances memory problems are experienced (and no lower or upper bounds have been reported by Cplex). Besides, the performance of Cplex is quite unstable, with differences in computing times (for instances of the same size) varying by a factor of 100 or more (cf. the first two lines of Table 3). On the contrary, Benders' performance remains stable with gaps at the root node being $\leq 0.3\%$ and the root node computing times staying below two hours in most of the cases.

Finally, to test the effects of the cardinality constraint (8) on the algorithmic performance, we provide Table 5 in which the perspective reformulation is compared with our Benders decomposition on the same set of benchmark instances, but without the constraint (8). One observes that in terms of the obtained final gaps and the overall CPU time, there is no significant difference in the performance of Benders, when compared to the results shown in Tables 3 and 4. Only for the instances of size 300×1500, the problem becomes easier (for both approaches), which can be explained by the fact that the optimal solution requires to open all available facilities, which is not the case for the remaining instances.

6. Conclusions

This paper provides a computational study of a modern implementation of Benders decomposition, applied to two problems with non-separable slaves: linear capacitated facility location, and its non-linear but convex variant known as congested CFL.

Our Benders implementation is rather simple and general, and was not tweaked for the specific application at hand. In particular, we introduced a simplified way of deriving optimality generalized Benders cuts for convex problems, that works well in practice and is straightforward to implement.

The resulting method is simpler and more efficient than some of the state-of-the art exact and heuristic approaches for linear CFL. The only exact method that remains competitive with Benders is another relatively simple approach proposed by Görtz and Klose [17] and based on a Lagrangian relaxation. As pointed out by its authors, this Lagrangian-based branch-and-bound can work surprisingly well because of the fast way of solving the Lagrangian relaxation in combination with an effective branching strategy. In such a setting, Benders decomposition is intrinsically penalized because of the large overhead imposed by the non-decomposable LP/QP slaves. Nevertheless, the obtained results indicate our simple Benders decomposition can be considered one of the state-of-the-art solution approaches for CFL.

As to the congested CFL problem, Benders clearly qualifies as the best-available solution method.

Acknowledgements

This research was funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-014. The first author was also supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"). The work of I. Ljubić and M. Sinnl was also supported by the Austrian Research Fund (FWF, Project P 26755-N19), while that of the last author was also supported by an STSM Grant from COST Action TD1207. Thanks are also due to three anonymous referees for their helpful comments.

[1] R. Aboolian, O. Berman, and D. Krass. Profit maximizing distributed service system design with congestion and elastic demand. *Transportation Science*, 46(2):247–261, 2012.

[2] P. Avella and M. Boccia. A cutting plane algorithm for the capacitated facility location problem. *Computational Optimization and Applications*, 43(1):39–65, 2009.

[3] P. Avella, M. Boccia, A. Sforza, and I. Vasil'ev. An effective heuristic for large-scale capacitated facility location problems. *Journal of Heuristics*, 15(6):597–615, 2009.

[4] J. Beasley. ORLIB, 2015. URL `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/capinfo.html`.

[5] W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.

[6] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[7] B. Boffey, R. Galvão, and L. Espejo. A review of congestion models in the location of facilities with immobile servers. *European Journal of Operational Research*, 178(3):643–662, 2007.

[8] P. Bonami, M. Kilinc, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *IMA Volumes in Mathematics and its Applications*, pages 1–39. Springer.

[9] G. Cornuéjols, R. Sridharan, and J. Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research*, 50(3):280–297, 1991.

[10] M. Desrochers, P. Marcotte, and M. Stan. The congested facility location problem. *Location Science*, 3 (1):9–23, 1995.

[11] J. J. Dongarra. Performance of various computers using standard linear equations software. 2014. URL `http://www.netlib.org/benchmark/performance.ps`.

[12] E. Fernández and M. Landete. Fixed-charge facility location problems. In G. Laporte, S. Nickel, and F. S. da Gama, editors, *Location Science*, pages 47–77. Springer, 2015.

[13] M. Fischetti and D. Salvagnin. An in-out approach to disjunctive optimization. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 136–140. Springer Berlin Heidelberg, 2010.

[14] M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning Benders Decomposition for Large Scale Facility Location. 2015. submitted.

[15] A. Frangioni and C. Gentile. A computational comparison of reformulations of the perspective relaxation: SOCP vs. cutting planes. *Operations Research Letters*, 37(3):206–210, 2009.

[16] A. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[17] S. Görtz and A. Klose. A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. *INFORMS Journal on Computing*, 24(4):597–610, 2012.

[18] G. Guastaroba and M. Speranza. Kernel search for the capacitated facility location problem. *Journal of Heuristics*, 18(6):877–917, 2012.

[19] O. Günlük and J. Linderoth. Perspective reformulation and applications. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *IMA Volumes in Mathematics and its Applications*, pages 61–92. Springer, 2012.

[20] J. Harkness and C. ReVelle. Facility location with increasing production costs. *European Journal of Operational Research*, 145(1):1–13, 2003.

[21] J. E. J. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[22] A. Klose and A. Drexl. Facility location models for distribution system design. *European Journal of Operational Research*, 162(1):4–29, 2005.

[23] A. Klose and S. Görtz. A branch-and-price algorithm for the capacitated facility location problem. *European Journal of Operational Research*, 179(3):1109–1125, 2007.

[24] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Mathematical Programming*, 69(1-3):111–147, 1995.

[25] T. L. Magnanti and R. T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29(3):464–484, 1981.

[26] J. Naoum-Sawaya and S. Elhedhli. An interior-point Benders based branch-and-cut algorithm for mixed integer programs. *Annals of Operations Research*, 210(1):33–55, 2013.

[27] S. Selfun. Outer approximation algorithms for the congested p-median problem. Master's thesis, Bilkent University, Ankara, Turkey, 2011.

[28] T. J. Van Roy. A cross decomposition algorithm for capacitated facility location. *Operations Research*, 34(1):145–163, 1986.

[29] F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005.

[30] P. Wentges. Accelerating Benders' decomposition for the capacitated facility location problem. *Mathematical Methods of Operations Research*, 44(2):267–290, 1996.

[31] Y. Zhang, O. Berman, and V. Verter. Incorporating congestion in preventive healthcare facility network design. *European Journal of Operational Research*, 198(3):922–935, 2009.

Table 3: Comparing perspective reformulation and Benders decomposition for congested CFL, 1/2. Column OPT gives the optimal solution value, $g_r[\%]$ the percentage gap at the root node, $t_r[s]$ the computing time at the root node, $g[\%]$ the final gap, and $t[s]$ the overall computing time. Runs reaching the timelimit of 50,000 seconds are indicated by TL, runs aborted due to memory limit with $--$.

					Perspective reformulation					Benders				
inst.	$\|J\|$	$\|I\|$	π	OPT	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes
1	300	300	0.4	257315.7360	0.1186	149	0.0000	233	130	0.0031	319	0.0000	383	10
1	300	300	0.6	214609.8293	0.0050	5057	0.0000	25115	17	0.0050	373	0.0000	594	30
1	300	300	0.8	219221.1886	0.4101	80	0.0000	155	49	0.0001	226	0.0000	236	0
6	300	300	0.4	273308.0002	0.0044	5675	0.0000	36598	28	0.0056	347	0.0000	583	32
6	300	300	0.6	225383.3635	0.0015	2329	0.0000	11696	9	0.0015	290	0.0000	299	3
6	300	300	0.8	228139.1799	0.0006	1825	0.0000	6196	5	0.0006	144	0.0000	165	0
11	300	300	0.4	259294.8204	0.0019	4936	0.0000	26260	17	0.0020	257	0.0000	271	3
11	300	300	0.6	216415.5633	0.0001	2257	0.0000	2257	0	0.0001	159	0.0000	167	0
11	300	300	0.8	224171.5836	0.0007	1842	0.0000	5273	5	0.0005	248	0.0000	254	0
16	300	300	0.4	256734.9041	0.0011	7417	0.0000	44495	33	0.0012	230	0.0000	281	8
16	300	300	0.6	220241.5213	0.0018	3063	0.0000	9463	6	0.0016	238	0.0000	250	3
16	300	300	0.8	231623.9657	0.0001	2888	0.0000	4641	3	0.0001	106	0.0000	115	0
1	500	500	0.4	433836.6527	2.3892	667	0.0000	1311	294	0.0017	1265	0.0000	2282	32
1	500	500	0.6	361323.3070	0.0006	50130	0.0006	TL	0	0.0007	1259	0.0000	1435	5
1	500	500	0.8	368022.2916	0.0000	17071	0.0000	17071	0	0.0000	587	0.0000	587	0
6	500	500	0.4	465717.4928	0.0005	4255	0.0005	TL	4	0.0005	823	0.0000	901	5
6	500	500	0.6	384364.0093	0.0004	11594	0.0004	TL	4	0.0006	975	0.0000	1430	12
6	500	500	0.8	393072.8259	0.0001	3000	0.0000	19734	3	0.0001	506	0.0000	532	0
11	500	500	0.4	420862.4354	0.0025	11074	0.0025	TL	5	0.0028	971	0.0000	2027	133
11	500	500	0.6	353185.6836	0.0009	3296	0.0009	TL	6	0.0010	962	0.0000	1664	32
11	500	500	0.8	366081.6812	0.0001	2698	0.0000	2698	0	0.0001	768	0.0000	814	0
16	500	500	0.4	398241.0995	0.0009	7310	0.0009	TL	5	0.0009	766	0.0000	946	9
16	500	500	0.6	345164.2574	0.0008	4956	0.0008	TL	5	0.0008	1061	0.0000	1719	17
16	500	500	0.8	367401.3250	0.0001	2984	0.0000	19396	3	0.0001	589	0.0000	639	0
1	300	1500	0.4	5307601.4506	–	–	–	–	–	0.1473	4520	0.1449	TL	242
1	300	1500	0.6	3620282.1081	–	–	–	–	–	0.0553	5284	0.0533	TL	187
1	300	1500	0.8	2807005.7213	0.0707	524	0.0000	26949	25822	0.0263	5074	0.0247	TL	391
6	300	1500	0.4	5657154.9641	41.6630	1114	16.8989	TL	1484	0.2638	4167	0.2610	TL	250
6	300	1500	0.6	3853223.4698	19.3752	592	0.0233	TL	3436	0.0881	4132	0.0862	TL	172
6	300	1500	0.8	2986886.5396	–	–	–	–	–	0.0235	4397	0.0211	TL	771
11	300	1500	0.4	5298861.8966	40.8702	930	35.3372	TL	2185	0.3654	3716	0.3635	TL	281
11	300	1500	0.6	3607873.1868	17.5410	699	0.3231	TL	2375	0.0842	4504	0.0812	TL	154
11	300	1500	0.8	2795126.8112	–	–	–	–	–	0.0291	4287	0.0279	TL	519
16	300	1500	0.4	5653309.6360	42.4055	914	26.0423	TL	1784	0.2974	4105	0.2962	TL	242
16	300	1500	0.6	3853536.5921	21.8576	428	5.7238	TL	2148	0.1387	4404	0.1351	TL	154
16	300	1500	0.8	2986872.5066	–	–	–	–	–	0.0247	4403	0.0223	TL	2183

Table 4: Comparing perspective reformulation and Benders decomposition for congested CFL, 2/2. Column OPT gives the optimal solution value, $g_r[\%]$ the percentage gap at the root node, $t_r[s]$ the computing time at the root node, $g[\%]$ the final gap and $t[s]$ the overall computing time. Runs reaching the timelimit of 50,000 seconds are indicated by TL, runs aborted due to memory limit with $--$

					Perspective reformulation					Benders				
inst.	$\|J\|$	$\|I\|$	π	OPT	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes
1	700	700	0.4	608104.4400	0.4047	1760	0.0000	3106	345	0.0007	2306	0.0000	4485	128
1	700	700	0.6	511852.0282	0.0002	49519	0.0002	TL	1	0.0003	2495	0.0000	2532	2
1	700	700	0.8	528832.2478	0.0005	11270	0.0005	TL	7	0.0005	2274	0.0000	3117	6
6	700	700	0.4	590223.9309	0.0009	17338	0.0009	TL	5	0.0009	2126	0.0000	3093	20
6	700	700	0.6	491995.9402	0.0001	12952	0.0001	TL	7	0.0002	2196	0.0000	2600	4
6	700	700	0.8	512486.5658	0.0001	11506	0.0000	11506	0	0.0001	1195	0.0000	1307	0
11	700	700	0.4	588518.4248	0.0017	50117	0.0017	TL	0	0.0021	2342	0.0000	19450	2630
11	700	700	0.6	496861.5248	0.0011	12357	0.0011	TL	7	0.0012	2462	0.0000	11539	528
11	700	700	0.8	514897.8446	0.0005	11357	0.0003	TL	8	0.0005	2327	0.0000	3175	8
16	700	700	0.4	591092.0635	0.0010	15292	0.0010	TL	6	0.0012	2326	0.0000	5530	382
16	700	700	0.6	498257.7984	0.0004	14950	0.0004	TL	6	0.0004	2002	0.0000	2294	5
16	700	700	0.8	515610.2532	0.0001	11468	0.0000	TL	8	0.0001	1227	0.0000	1332	0
1	1000	1000	0.4	831618.2005	1.0782	5785	0.0000	12826	705	0.0015	7245	0.0006	TL	3649
1	1000	1000	0.6	700140.6641	–	–	–	–	–	0.0006	6102	0.0000	19435	237
1	1000	1000	0.8	720445.3031	–	–	–	–	–	0.0004	8256	0.0000	11922	7
6	1000	1000	0.4	884498.8703	–	–	–	–	–	0.0007	5849	0.0000	10162	90
6	1000	1000	0.6	739680.3837	–	–	–	–	–	0.0002	6640	0.0000	10429	13
6	1000	1000	0.8	765867.8192	–	–	–	–	–	0.0002	6829	0.0000	7263	2
11	1000	1000	0.4	808297.2103	–	–	–	–	–	0.0012	6003	0.0001	TL	2666
11	1000	1000	0.6	692675.4305	–	–	–	–	–	0.0003	4907	0.0000	8085	22
11	1000	1000	0.8	729765.8357	–	–	–	–	–	0.0002	5631	0.0000	6353	3
16	1000	1000	0.4	852614.2315	–	–	–	–	–	0.0015	4697	0.0005	TL	4700
16	1000	1000	0.6	719272.2322	–	–	–	–	–	0.0003	5503	0.0000	9205	37
16	1000	1000	0.8	744746.7001	–	–	–	–	–	0.0001	3098	0.0000	3208	2

Table 5: Comparing perspective reformulation and Benders decomposition for the congested CFL without constraint (8). Column OPT gives the optimal solution value (OPT), $g_r[\%]$ the percentage gap at the root, $t_r[s]$ the computing time at the root, $g[\%]$ the final gap and $t[s]$ the overall computing time. Runs reaching the timelimit of 50 000 seconds are indicated by TL, runs which crashed due to memorylimit with $--$.

				Perspective reformulation					Benders				
inst.	$\|J\|$	$\|I\|$	OPT	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes	$gap_r[\%]$	$t_r[s]$	$gap[\%]$	$t[s]$	nodes
1	300	300	211070.0381	0.0018	138	0.0000	152	12	0.0015	449	0.0000	507	3
6	300	300	222243.8724	–	–	–	–	–	0.0005	203	0.0000	213	0
11	300	300	2529554.9391	–	–	–	–	–	0.0000	295	0.0000	368	0
16	300	300	219566.4352	0.0006	4438	0.0000	11135	7	0.0003	236	0.0000	260	3
1	300	1500	2529554.9567	6.6603	1695	0.0000	1695	1	0.0000	239	0.0000	300	0
6	300	1500	2361077.7437	0.0000	1539	0.0000	1941	11	0.0000	318	0.0000	398	0
11	300	1500	214763.8218	0.0000	569	0.0000	839	17	0.0008	347	0.0000	416	7
16	300	1500	2352252.4237	0.0000	611	0.0000	1017	17	0.0000	224	0.0000	282	0
1	500	500	355776.2393	0.0016	458	0.0000	727	37	0.0004	1121	0.0000	1359	8
6	500	500	380401.4018	–	–	–	–	–	0.0004	1066	0.0000	1120	3
11	500	500	350559.5589	–	–	–	–	–	0.0002	658	0.0000	690	1
16	500	500	344701.1085	0.0002	8663	0.0002	TL	5	0.0001	861	0.0000	877	0
1	700	700	506905.1205	0.0027	1458	0.0000	2367	71	0.0004	1791	0.0000	1960	0
6	700	700	488584.4417	0.0004	23214	0.0000	48525	5	0.0004	2495	0.0000	2884	6
11	700	700	493751.8204	0.0009	24636	0.0009	TL	5	0.0008	2618	0.0000	6364	117
16	700	700	494914.6701	0.0003	26437	0.0003	TL	3	0.0001	2248	0.0000	2248	2
1	1000	1000	693170.4984	0.0009	5048	0.0000	8743	269	0.0005	9504	0.0000	30432	245
6	1000	1000	734986.4967	–	–	–	–	–	0.0001	5036	0.0000	5321	0
11	1000	1000	690997.7121	–	–	–	–	–	0.0005	6443	0.0000	17972	164
16	1000	1000	714499.4930	–	–	–	–	–	0.0003	5702	0.0000	8745	10