Thinning out facilities: a Benders decomposition approach for the uncapacitated facility location problem with separable convex costs

Matteo Fischetti[1], Ivana Ljubić[2], Markus Sinnl[2]

[1] Department of Information Engineering, University of Padua, Italy, matteo.fischetti@unipd.it
[2] Department of Statistics and Operations Research, University of Vienna, Austria, {ivana.ljubic,markus.sinnl}@univie.ac.at

April 1, 2015

Abstract

The Uncapacitated Facility Location (UFL) problem is one of the most famous and most studied problems in the Operations Research literature. Given a set of potential facility locations, and a set of customers, the goal is to find a subset of facility locations to open, and to allocate each customer to a single open facility, so that the facility opening plus customer allocation costs are minimized. For each customer, the allocation cost is assumed to be a linear or separable convex quadratic function. In this paper we "'thin out" the classical models from the literature, and use generalized Benders cuts to replace a huge number of allocation variables by a small number of continuous variables that model the customer allocation cost directly. Although the idea of using Benders cuts for UFL is not new (at least, when linear costs are considered), it was apparently never computationally investigated in recent years. Instead, we show that the approach allows for a significant boost in the performance of a Mixed-Integer Programming solver, and report the optimal solution of a large set of previously unsolved benchmark. In particular, dramatic speedups are achieved for UFL's with separable quadratic allocation costs—which turns out to be much easier than their linear counterpart when our approach is used.

The Uncapacitated Facility Location (UFL) problem is one of the most famous and most studied problems in the Operations Research literature. Given a set I of potential facility locations, and a set J of customers, the goal is to find a subset of facility locations to open, and to allocate each customer to a single open facility, so that the facility opening plus customer allocation costs are minimized. In its classical version, the allocation cost for each customer is assumed to be a linear function of the demand served by open facilities. The problem can be easily formulated as a compact Mixed-Integer Linear Program (MILP). In the last 50 years, two variants of this model (with aggregated and disaggregated constraints) have been traditionally used in the literature. Both variants however rely on a huge number of allocation variables, which is one of the main

reasons why UFL still imposes a challenge for modern general-purpose MILP solvers and Lagrangian relaxation techniques are generally preferred [4, 31, 38].

Recent experience with the Steiner Tree Problem [9] showed that some known MILP models can be "thinned out" by removing unnecessary variables and constraints, with a very significant performance boost. In this paper we apply the same approach to UFL, and use (generalized) Benders cuts to actually thin out the classical models. In the resulting formulation, the huge number of allocation variables is replaced by a linear number of continuous variables that model the customer allocation cost directly, or even by a single continuos variable. For UFL with linear costs, our Benders model is compact as it involves $|I| + |J|$ variables and $|J| \cdot (|I| + 1)$ constraints.

In addition to UFL, we also study the *quadratic UFL* (qUFL) problem in which customer demands are equal to one and allocation costs are proportional to the square of the fraction of the demand allocated to a given facility. This problem, also known as *separable convex quadratic UFL*, has been introduced in [19]. Due to its simple structure, qUFL has been a subject of intensive studies in the recent years. Many of the recently proposed methods for mixed integer nonlinear programming (MINLP) consider qUFL as an inevitable part of their computational studies, see e.g., [6, 8, 20, 21]. The importance of understanding computational techniques for qUFL is also confirmed by its presence in the Conic Benchmark library (CBLIB [14]), which is a library of the most relevant and challenging benchmark problems for conic optimization.

Although the idea of using Benders cuts for UFL with linear costs is definitely not new and can be considered folklore, to the best of our knowledge it was not computationally investigated in recent years. The outcome of our research is twofold:

1. On the one hand, the "thinning out" approach allows for a very significant boost in the performance of the MILP solver, as we were able to solve to proven optimality 7 previously unsolved benchmark instances for UFL, and to improve the best-known heuristic value for 22 additional instances. These instances were out of reach for any state-of-the-art MILP solver, as the underlying models would involve tens of millions of variables and constraints.

2. On the other hand, speedups of 4 orders of magnitude or more are reported for qUFL with respect to previous solution methods—to our surprise, qUFL turned out to be much easier than its linear counterpart when our approach is properly implemented.

The paper is organized as follows. The classical UFL models are reported in Section 1 for both the linear and separable quadratic cost cases. Nonlinear Benders cuts for convex optimization are reviewed in Section 2. Section 3 outlines our overall solution scheme, with a discussion of actual implementation issues that played a fundamental role for the effectiveness of our final solution algorithm. Computational results for UFL with both linear and separable quadratic

allocation costs are given in Section 4, while concluding remarks and possible future works are finally addressed in Section 5.

1 MIP models

Let I be the index set of facility locations ($|I| = n$), let $f_i \geq 0$ be opening costs for each facility $i \in I$, and let J be the index set of customers ($|J| = m$) with allocation costs $c_{ij} \geq 0$ defined for each pair $(i, j) \in I \times J$. We will 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 MILP formulation for UFL, $n + m \cdot n$ variables are used to model the problem. For each $i \in I$, binary variable y_i is set to one if facility i is open, and to zero, otherwise. For each $i \in I$ and $j \in J$, allocation variable x_{ij} is set to one, if customer j is served by facility i, and to zero otherwise.

Linear case

The classical UFL model for the linear case then reads

$$\min \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1}$$

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

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

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

$$y_i \in \{0, 1\} \qquad \forall i \in I$$

The objective is to minimize the sum of facility opening costs, plus the customer allocation costs. Constraints (2) make sure that every customer is assigned to exactly one facility, and capacity constraints (3) make sure that allocation to a facility i is only possible if this facility is open. Note that the integrality condition on variables x_{ij} is redundant, i.e., for any integer y^* each customer j will set $x^*_{ij} = 1$ for the closest facility i with $y^*_i = 1$.

The model shown above is known as the disaggregated formulation, whereas in its aggregated counterpart $m \cdot n$ constraints of type (3) are replaced by n weaker constraints $\sum_{j \in J} x_{ij} \leq m \cdot y_i$, for all $i \in I$.

There is a large body of work available in the existing literature on exact and heuristic approaches for UFL. Recent survey articles focus on applications of facility location in design of distribution systems [25], or supply chain management [36], and a more general survey on UFL is given in [39]. Approaches applied to UFL range from approximation algorithms [33], over (semi-) Lagrangian relaxations [4], and metaheuristics (see, e.g., a survey in [3]), to branch-and-bound based algorithms in which lower bounds are calculated using dual ascent procedures (see, e.g. the most recent one in [32]). The state-of-the-art exact algorithm for UFL is given in [38]: the algorithm is based on a message passing approach

in which a metaheuristic calculates the upper bounds and passes the information to a branch-and-bound algorithm in which lower bounds are obtained by a Lagrangian relaxation solved using a bundle method. A more comprehensive overview of the most relevant recent literature can also be found in [38].

Separable convex quadratic case

UFL with separable convex quadratic allocation costs (denoted as qUFL in the following), has been introduced in [19]. In this version of the problem, one assumes $c_{ij} > 0$ for all $i \in I$ and $j \in J$, and the allocation costs are proportional to the square of a customer's demand served by an open facility. More precisely, the objective function (1) is replaced by:

$$\min \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}^2 \tag{4}$$

Contrarily to the linear case, each variable x_{ij} will now represent the fraction of demand of customer j served by facility i (in general, these values will no longer assume integer values in the optimal solutions).

Given a binary vector y^*, there is a simple closed formula to compute optimal allocation values for each customer. More specifically, for each customer $j \in J$ an optimal solution will set $x_{ij}^* = 0$ for all i with $y_i^* = 0$, and $x_{ij}^* = \delta_j^*/c_{ij}$ for all i with $y_i^* = 1$, where the normalization factor $\delta_j^* = 1/\sum_{i \in I: y_i^* = 1}(1/c_{ij})$ guarantees the fulfillment of (2); see e.g. [19] for details.

Sophisticated linearization and bounding techniques for qUFL have been proposed in [19]. More recently, a very tight convex (second-order cone) MIP formulation has been given in [20], namely:

$$\min \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} z_{ij} \tag{5}$$

$$\text{s.t.} \sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{6}$$

$$x_{ij} \leq y_i \qquad \forall i \in I, j \in J \tag{7}$$

$$x_{ij}^2 \leq z_{ij}\, y_i \qquad \forall i \in I, j \in J \tag{8}$$

$$x_{ij} \geq 0 \qquad \forall i \in I, j \in J \tag{9}$$

$$z_{ij} \geq 0 \qquad \forall i \in I, j \in J \tag{10}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I \tag{11}$$

where (8) are second-order cone (hence convex) constraints as the right-hand-side is the product of nonnegative variables. The model above is called *perspective reformulation* (see, e.g., [13]) as it strengthens the obvious definition of the z_{ij} variables through constraints $x_{ij}^2 \leq z_{ij}$, by replacing the left-hand side convex function x_{ij}^2 with its perspective defined by $y_i(x_{ij}/y_i)^2$ if $y_i > 0$, and zero if $y_i = 0$; see again [20] for details.

Computational experience reported in [20] shows that continuous relaxation of model (5)-(11), though very large, produces much tighter lower bounds than its counterpart based on (4), so we used it in our study.

2 Benders decomposition for convex optimization

Despite this broad set of solution approaches for UFL, it seems that Benders-like decomposition methods have been neglected so far—at least from a computational viewpoint. As already mentioned, the aim of the present paper is to close this gap and asses the computational performance of Benders decomposition for UFL and its quadratic counterpart with separable convex objective function.

Our overall framework works as follows: as x_{ij} variables are a bottleneck for MIP solvers, we just remove them from the model, and introduce in the objective function a new set of continuous variables w_j representing the allocation-cost for all $j \in J$. The resulting *master problem* is then given by

$$\min \sum_{i \in I} f_i y_i + \sum_{j \in J} w_j \tag{12}$$

$$\text{s.t.} \sum_{i \in I} y_i \geq 2 \tag{13}$$

$$w_j \geq \Phi_j(y) \qquad \forall j \in J \tag{14}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{15}$$

where the convex function $\Phi_j(y)$ appearing in (14) gives the minimum allocation cost for customer j for any given (possibly noninteger) point $y \in [0,1]^I$ with $\sum_{i \in I} y_i \geq 2$ (which is the only case of interest for branch-and-cut separation).

Note that we require to open, at least, 2 facilities in (13), as the the single-facility case can be easy handled in a preprocessing phase. Actually, when the number of facilities is not too large, one can even require to open 3 or more facilities after having efficiently enumerated all 1- and 2-facility solutions (in our implementation, this latter option is activated only for instances with linear costs and $n \leq 1000$).

Due to the convexity of the $\Phi_j(y)$'s, master problem (12)-(15) is in fact a convex MINLP that can be solved as a MILP by a branch-and-cut approach where linear outer-approximations of constraints (14) are generated on the fly as follows. Let $y^* \in [0,1]^I$ be a given (possibly noninteger) point with $\sum_{i \in I} y_i^* \geq 2$, and consider a generic customer j. Barring subscript j to ease notation, because of convexity, function $\Phi(y)$ can be underestimated by a supporting hyperplane at y^*, so we can write

$$w \geq \Phi(y) \geq \Phi(y^*) + \sum_{i \in I} \frac{\partial \Phi(y^*)}{\partial y_i}(y_i - y_i^*) \tag{16}$$

Note that (16) can be interpreted as a sensitivity analysis of $\Phi(y)$ in y^*, so it is not surprising that its computation actually requires dual information according to the following scheme.

Each term $\Phi(y)$ is computed by solving a convex *slave subproblem*. For the sake of generality, let this slave be generically written as

$$\Phi(y) = \min\{f(x,y) : g_k(x,y) \leq 0,\ k = 1, \cdots, K,\ x \in X\} \tag{17}$$

where f and $g_1, \cdots, g_k$ are convex and twice differentiable functions and X is a closed convex set.

Given y^*, one can solve efficiently the slave problem (17) for $y = y^*$. Let x^* be the optimal primal solution found, and let $u_k^* \geq 0$ be the optimal dual variables associated with $g_k(x,y) \leq 0$. (In our notation, dual variables correspond to Lagrangian multipliers and are nonnegative even for minimization problems with $\leq$ constraints.)

Using Lagrangian duality and KKT conditions, and assuming constraint qualifications hold, Geoffrion [15] (see also [1]) proved that

$$\frac{\partial \Phi(y^*)}{\partial y_i} = \frac{\partial f(x^*, y^*)}{\partial y_i} + \sum_{k=1}^{K} u_k^* \frac{\partial g_k(x^*, y^*)}{\partial y_i} \tag{18}$$

An intuitive explanation of this result is as follows. By Lagrangian duality and because of convexity (that plays a fundamental role here), for a given optimal dual vector $(u_1^*, \cdots, u_K^*)$ the local behavior of $\Phi(y)$ in y^* is determined by the Lagrangian function in u^*, namely for y sufficiently close to y^* one has

$$\Phi(y) \approx \min\{f(x,y) + \sum_{k=1}^{K} u_k^* g_k(x,y) : x \in X\} = f(x^*, y^*) + \sum_{k=1}^{K} u_k^* g_k(x^*, y)$$

hence taking partial derivatives of the the right-hand side leads to (18).

The above considerations lead to the following *Generalized Benders (GB) cuts* [15] to be used within a branch-and-cut MIP solver:

$$w \geq \Phi(y^*) + \sum_{i \in I} \alpha_i^* (y_i - y_i^*)$$

where

$$\alpha_i^* = \frac{\partial f(x^*, y^*)}{\partial y_i} + \sum_{k=1}^{K} u_k^* \frac{\partial g_k(x^*, y^*)}{\partial y_i}$$

2.1 Benders cuts for linear case

For the linear case, Benders decomposition has been already mentioned in [34] along with the fact that the family of Benders cuts is of polynomial size [7]. However, for the last 30+ years, this result has been forgotten and, to the best of our knowledge and to our surprise, no computational studies were done to asses the practical usefulness of this approach.

For the linear case, a generic slave has the form of the following knapsack problem (in minimization form)

$$\Phi(y^*) = \min \sum_{i \in I} c_i x_i \tag{19}$$

$$\text{s.t.} \sum_{i \in I} x_i = 1 \tag{20}$$

$$x_i \leq y_i^* \qquad \forall i \in I \tag{21}$$

$$x_i \geq 0 \qquad \forall i \in I \tag{22}$$

For a given $y^* \in [0,1]^I$ with $\sum_{i \in I} y_i^* \geq 2$, let x^* be an optimal primal solution, β^* be the optimal dual variable for constraint (20), and u_i^* be nonnegative optimal dual variables for constraints (21). The Lagrangian function in (x^*, β^*, u^*) then reads

$$\sum_{i \in I} c_i x_i^* + \beta^* (1 - \sum_{i \in I} x_i^*) + \sum_{i \in I} u_i^* (x_i^* - y_i)$$

hence

$$\frac{\partial \Phi(y^*)}{\partial y_i} = -u_i^*$$

so the Benders cut reads

$$w \geq \Phi(y^*) - \sum_{i \in I} u_i^* (y_i - y_i^*)$$

and only requires the computation of the optimal dual variables $u_i^* \geq 0$ associated with constraints (21).

It is well known from knapsack theory that the following *Dantzig algorithm* produces an optimal primal solution x^* and optimal dual variables $u_i^* \geq 0$ associated with constraints (21) for a given $y^* \in [0,1]^I$ with $\sum_{i \in I} y_i^* \geq 2$.

Let us assume costs have been sorted to get $c_1 \leq \cdots \leq c_n$, and let the index of the *critical item* be defined as the index $k \in I$ such that

$$\sum_{i=1}^{k-1} y_i^* < 1 \leq \sum_{i=1}^{k} y_i^*$$

Then an optimal primal solution x^* is obtained by setting

$$x_i^* = \begin{cases} y_i^* & \text{for } i < k \\ 1 - \sum_{i=1}^{k-1} y_i^* & \text{for } i = k \\ 0 & \text{for } j > k \end{cases} \qquad i \in I$$

As to the dual solution, the optimal dual variable for constraint (20) is $\beta^* = c_k$, while the optimal dual variables for constraint (21) are $u_i^* = c_k - c_i$ for $i < k$, and $u_i^* = 0$ for $i \geq k$.

Optimality comes from the feasibility of the primal and dual solutions for their problems, and from the fact that the primal cost $\sum_{i\in I} c_i x_i^* = \sum_{i=1}^{k-1} c_i y_i^* + c_k(1-\sum_{i=1}^{k-1} y_i^*)$ is equal to the dual cost $\beta^* - \sum_{i=1}^{n} u_i^* y_i^* = c_k - \sum_{i=1}^{k-1}(c_k - c_i)y_i^*$. It then follows that

$$\Phi(y^*) = c_k - \sum_{i=1}^{k-1}(c_k - c_i)y_i^*$$

so there are only n distinct Benders cuts for a given customer j, one for each $k \in I$, each of the form

$$w + \sum_{i=1}^{k-1}(c_k - c_i)y_i \geq c_k \quad \forall k \in I$$

(assuming locations have been permuted so as to have $c_1 \leq \cdots \leq c_n$).

2.2 Generalized Benders cuts for quadratic case

In this case, for a given y^*, the slave is the convex program (recall that $c_i > 0$ for all $i \in I$).

$$\Phi(y^*) = \min \sum_{i\in I} c_i z_i \tag{23}$$

$$\text{s.t.} \sum_{i\in I} x_i = 1 \tag{24}$$

$$x_i \leq y_i^* \qquad \forall i \in I \tag{25}$$

$$x_i^2 \leq z_i\, y_i^* \qquad \forall i \in I \tag{26}$$

$$x_i \geq 0 \qquad \forall i \in I \tag{27}$$

$$z_i \geq 0 \qquad \forall i \in I \tag{28}$$

To avoid technicalities, we assume $y_i^* > 0$ for all $i \in I$, and refer to Section 3.1 for the general case. Also, we relax $=$ into $\geq$ in (24), and we observe that non-negativity constraints (27) and (28) are redundant and can be relaxed.

Let (x^*, z^*) be an optimal primal solution, and β^*, u_i^* and v_i^* be nonnegative optimal dual variables for constraints (24), (25) and (26), respectively. The Lagrangian function in $(x^*, z^*, \beta^*, u^*, v^*)$ reads

$$\sum_{i\in I} c_i z_i^* + \beta^*(1 - \sum_{i\in I} x_i^*) + \sum_{i\in I} u_i^*(x_i^* - y_i) + \sum_{i\in I} v_i^*({x_i^*}^2 - z_i^* y_i)$$

hence

$$\frac{\partial \Phi(y^*)}{\partial y_i} = -u_i^* - v_i^* z_i^*$$

and the GB cut reads

$$w \geq \Phi(y^*) - \sum_{i\in I} \left(u_i^* + v_i^* z_i^*\right)\left(y_i - y_i^*\right) \tag{29}$$

As in the linear case, one can derive a specialized algorithm for a fast (and numerically accurate) solution of the slave problem, and for the construction of the corresponding GB cut (29).

Observe that, because of the positive costs in (23) and of (26), the optimal z^* satisfies $z_i^* = (x_i^*)^2/y_i^*$ for all $i \in I$. Being y^* fixed, we can just remove those variables and work on the associated problem

$$\Phi(y^*) = \min \sum_{i \in I} \gamma_i x_i^2 \tag{30}$$

$$\text{s.t.} \sum_{i \in I} x_i \geq 1 \tag{31}$$

$$x_i \leq y_i^* \qquad \forall i \in I \tag{32}$$

with respect to the new "perspective costs" $\gamma_i = c_i/y_i^* > 0$.

Once the primal solutions x^* and the optimal dual variables u^* associated with (32) have been determined, the optimal primal variables z_i^* for model (23)-(28) are easily computed as

$$z_i^* = (x_i^*)^2/y_i^* \text{ for all } i \in I$$

while the optimal dual variables $v_i^* \geq 0$ corresponding to (26) can be computed as c_i/y_i^* so as to satisfy the first order condition $c_i - v_i^* y_i^* = 0$.

Finally, the most violated GB cut for the given y^* can be computed as in (29).

Implementing an ad-hoc QP algorithm

Fast algorithms for solving simple QP problems with a knapsack-like constraint and box constraints are available in the literature (see, e.g., [22, 37]). To make our results reproducible, we next describe the actual algorithm we used for solving QP problem (30)-(32) and, for the sake of completeness, provide a proof of its correctness.

Let us consider first the following *residual problem* where the x variables have no upper bounds, $R \subseteq I$ is a given *residual set* of facilities, and $r \in [0, 1]$ is a given *residual request*:

$$\min \sum_{i \in R} \gamma_i x_i^2 \tag{33}$$

$$\text{s.t.} \sum_{i \in R} x_i \geq r \tag{34}$$

Let $\beta^* \geq 0$ be the dual multiplier for inequality (34). It is well known [19]—and easy to prove using KKT optimality conditions—that an optimal primal-dual pair (x^*, β^*) can be computed as:

Input : A point $y^* > 0$ with $\sum_{i \in I} y_i^* \geq 2$ along with the cost vector $\gamma > 0$

Output: Optimal value `vopt`, optimal primal solution $x^* \geq 0$, and optimal dual solution $(u^*, \beta^*) \geq 0$ for the quadratic model (30)-(32)

```
/* compute optimal primal solution x* along with β*      */
R ← I;
r ← 1;
again ← TRUE;
while ( r > 0  and  again ) do
    β* ← 2 r / Σ_{i∈R} (1/γ_i);
    again ← FALSE ;
    foreach i ∈ R do
        x_i* ← β* / (2 γ_i);
        if  ( x_i* > y_i* ) then
            x_i* ← y_i*;
            r ← r − y_i*;
            R ← R \ {i};
            again ← TRUE;
        end
    end
end
/* compute optimal value vopt and dual solution u*      */
if ( r ≤ 0 or R = ∅ ) then  β* ← 0 ;
vopt ← 0;
for i ∈ I do
    if  i ∈ R  then u_i* ← 0 else u_i* ← β* − 2 γ_i y_i*;
    vopt ← vopt  +  γ_i (x_i*)^2 ;
end
```

Algorithm 1: Our specialized QP solver

$$\beta^* = \frac{2r}{\sum_{i \in R} 1/\gamma_i} \qquad\qquad x_i^* = \frac{\beta^*}{2\,\gamma_i} \text{ for all } i \in R \tag{35}$$

Our algorithm to solve the quadratic model with bounded variables (30)-(32) derives the optimal solution by subsequently solving the residual problem (33)-(34) for various values of r and R; see Algorithm 1.

The primal optimal solution x^* is computed at Steps 1-17 together with β^*. To this end, we first set $R = I$ and $r = 1$ (Steps 2-3) and use (35) to compute the optimal values of x_i^* (and β^*) by disregarding their upper bounds in (32). If it happens that all upper bounds are in fact fulfilled, we are done. Otherwise, for each $x_i^* > y_i^*$ we clip x_i^* to y_i^* (Step 11), update r and R accordingly (Steps 12-13), and repeat. The optimal nonnegative dual variables u_i^* corresponding to (32) are finally defined at Step 22.

To prove the correctness of Algorithm 1, we need the following

Lemma 2.1. *Values β^* computed at each iteration of Step 6 define a strictly monotonically increasing sequence.*

Proof. Let us assume, without loss of generality, that in a given iteration only a single i is removed from R. Let β' and β'' be the value of β^* at Step 6 before and after the removal of i from R arising at Step 13, respectively, and let r' and r'' be the corresponding values of r. To simplify notation, let us define $\rho_i = 1/\gamma_i$ and $d = \sum_{t \in R}(1/\gamma_t)$. We have to show that $\beta'' > \beta'$, where $\beta' = 2r'/d$ and $\beta'' = 2(r' - y_i^*)/(d - \rho_i)$. To this end, observe that the removal of i from R implies $x_i^* = \beta'/(2\gamma_i) > y_i^*$ at Step 10, i.e., $2y_i^* < \beta'\rho_i$ holds. So

$$\beta'' = \frac{2r' - 2y_i^*}{d - \rho_i} > \frac{2r' - \beta'\rho_i}{d - \rho_i} = \beta' \; \frac{2r'/\beta' - \rho_i}{d - \rho_i} = \beta' \; \frac{d - \rho_i}{d - \rho_i} = \beta'$$

as claimed. □

Theorem 2.1. *Algorithm 1 returns an optimal primal solution x^* and an optimal dual solution (u^*, β^*) of problem* (30)-(32).

Proof. The Lagrangian function for problem (30)-(32) is defined as

$$L(x, u, \beta) = \sum_{i \in I} \gamma_i {x_i}^2 + \beta\,(1 - \sum_{i \in I} x_i) + \sum_{i \in I} u_i\,(x_i - y_i^*)$$

As we have a convex quadratic problem with linear inequalities, all we have to show is that (x^*, u^*, β^*) satisfies the KKT conditions:

$$\nabla_x L(x^*, u^*, \beta^*) = 0 \tag{36}$$

$$(x^*, u^*, \beta^*) \geq 0 \tag{37}$$

$$\sum_{i \in I} x_i^* \geq 1 \tag{38}$$

$$\beta^*\;(1 - \sum_{i \in I} x_i^*) = 0 \tag{39}$$

$$u_i^*\;(x_i^* - y_i^*) = 0 \quad \forall i \in I \tag{40}$$

Let notation R and r refer to the quantities available at the end of the algorithm. For condition (36), we have two cases:

- $i \in R$: in this case, $u_i^* = 0$ (Step 22) and $x_i^* = \beta^* / (2\gamma_i)$ (Step 9), hence $\partial L(x^*, u^*, \beta^*)/\partial x_i = 2\gamma_i x_i^* - \beta^* = 0$;
- $i \notin R$: in this case, $u_i^* = \beta^* - 2\gamma_i y_i^*$ (Step 22) and $x_i^* = y_i^*$ (Step 11), hence $\partial L(x^*, u^*, \beta^*)/\partial x_i = 2\gamma_i x_i^* - \beta^* + u_i^* = 0$.

Conditions (37) are obvious except for u_i^* and $i \notin R$, when $u_i^* = \beta^* - 2\,\gamma_i\, y_i^*$ is defined at Step 22. Let β' be the value of β^* computed at Step 6 at the iteration when i is removed from R at Step 13. Then $x_i^* = \beta'/(2\,\gamma_i) > y_i^*$ at Step 10. Due to Lemma 2.1, $\beta^* \geq \beta'$. By combining this with the rewritten form $\beta' > 2\,\gamma_i\, y_i^*$ of the previous inequality, we obtain $\beta^* > 2\,\gamma_i\, y_i^*$, from which $u_i^* \geq 0$ follows.

Also obvious is the complementary slackness condition (39), while (40) derives from $u_i^* = 0$ for all $i \in R$ (Step 22) and $x_i^* = y_i^*$ for all $i \notin R$ (Step 11). □

Thinning out w_j's variables in the master

In the aim of thinning out irrelevant variables from the master, one can think of replacing all w_j's by a single continuous variable

$$w_{sum} = \sum_{j \in J} w_j$$

and of aggregating the individual GB cuts (2) for each customer $j \in J$, namely

$$w_j \geq \Phi_j(y^*) + \sum_{i \in I} \alpha_{ij}^* (y_i - y_i^*) \tag{41}$$

into a single *cumulative GB cut*

$$w_{sum} \geq \sum_{j \in J} \Phi_j(y^*) + \sum_{i \in I} (\sum_{j \in J} \alpha_{ij}^*)(y_i - y_i^*) \tag{42}$$

In this setting, for each y^* of the master one generates (at most) one violated cut, as opposed to the (at most) $|J|$ individual cuts that can be generated by keeping the disaggregated variables w_j's.

In what follows, the model with the individual $w_j's$ variables will be called the *fat model*, while the model with the single w_{sum} variable will be referred to as the *slim model*.

The slim model has both pros and cons with respect to the fat one.

From the positive side, one does not really need to know the individual w_j's values associated with a given y, as the actual solution cost only depends on their sum. By removing the individual w_j's one can therefore remove potentially-disturbing information. In addition, by using the single w_{sum} variable one expects to generate fewer cuts during the branch-and-cut solution process, hence

the LP relaxations of the master will be smaller in terms of variables and (hopefully) of explicitly-generated GB cuts.

From the negative side, cuts (42) tend to be denser than (41). In addition, we know that each cumulative cut (42) is just implied (à la Farkas) by their individual counterparts (41), and people working with MIP's have a Pavlov's unconditioned aversion to aggregated formulations. However, in our setting the quality of the lower bound of the master is theoretically the same as they both are the projection on different w-subspaces of the same formulation, so we are not really losing anything in terms of lower bound if we opt for the cumulative cut.

As it will be discussed in the next section, what matters is in fact the availability of a sound cutting plane scheme that can effectively produce good lower bounds even when a single cut is generated at each separation call. As a matter of fact, at least for quadratic UFL, the slim variant is much more effective than the fat one; see Subsection 3.4 for details.

3 The overall solution framework

Once the GB cut separators for both the linear and quadratic cases have been implemented, one has to design the overall solution framework for the master MILP problem.

A natural choice is to use a state-of-the-art commercial MILP solver—we used IBM ILOG Cplex 12.6 in our implementation. Using a state-of-the-art MILP solver does in fact simplify a lot the implementation, as one relies on a very robust and efficient external framework for the parametric solution of the node LP's, primal heuristics, generation of general MILP cuts, branching, cut pool handling, etc. So, for a quick shot one only has to plug the GB cut separator in the framework, and see how it works.

This simple approach is in fact rather effective, as it already allowed us to solve previously unsolved UFL instances for both linear and quadratic cases. However, we obtained much better results (in particular, for the quadratic case) by adding some more ingredients to the basic recipe, as outlined below.

3.1 Numerically accurate GB cuts for the quadratic case

In the quadratic case, when the point y^* to separate contains zero entries we face theoretical issues in the definition of the "perspective costs" $\gamma_i = c_i/y_i^*$ and hence of the most-violated GB cut. Although in theory there are elegant ways to cope with the non-differentiability at such points (see [20]), in practice we observed that the GB cut becomes numerically unreliable even when $y_i^* \approx 0$ for some i, hence the risk of producing invalid cuts becomes a real issue.

We have therefore implemented a very simple way to encompass the above difficulty which is based on the fact that, in our separation framework, y^* is in fact a known quantity—while in the compact MILP (5)-(11) it plays the role of a variable. We introduced a (not too small) internal threshold $\varepsilon = 10^{-5}$ to

decide whether a given y^* is close enough to integrality. Then, for each $i \in I$ with $y_i^* < \varepsilon$ we just resist the temptation and do not apply the perspective strengthening of x_{ij}^2. This means that we replace $x_{ij}^2 \leq z_{ij}\ y_i$ by $x_{ij}^2 \leq z_{ij}$ in (26). In this way we have $\gamma_i = c_i$ in (30), and the dual variable v_i^* will no longer appear in the GB cut as the new constraint does not depend on y anymore.

Although not strictly required, we also apply the same procedure when $y_i^* > 1 - \varepsilon$, as in this case what we gain from the perspective strengthening is negligible. In a sense, we view the components of y^* that are "almost integer" as good enough, and we do not believe it is worth to penalize them through the (numerically risky) perspective reformulation.

There is a second way to deal with zero entries in y^*, that we call $2\,\varepsilon$ *trick*: just replace each y_i^* with $y_i^* + 2\,\varepsilon$ and apply GB separation to this perturbed point—an idea used by many authors, including [5]. This is mathematically correct because the GB cut we generate is always valid—though its violation with respect to original y^* is slightly underestimated. Although it may appear naive, this approach is rather effective and can significantly improve the overall convergence of the cut loop, as shown in the next subsection. In fact, this idea is closely related to the approach proposed by [18] of "perturbing" perspective inequalities by moving y towards a point "sufficiently inside" the perspective cone—the main difference being that we face a much simpler situation where y^* is given, whereas in [18] one needs to treat y as a variable.

3.2 Cut loop stabilization and the curse of Kelley

This is perhaps the most important ingredient in our recipe. In the MIP community, a lot of attention is generally paid to the polyhedral strength of the generated cuts (e.g., the fact of being facet defining), giving for granted that the external cutting plane loop will follow Kelley's scheme [24]. According to this scheme, at each cut loop iteration one generates one or more cuts that are violated by the current (fractional) solution y^*, adds them to the current relaxation, reoptimizes it and gets a new optimal solution y^* to be cut at the next iteration.

However, the convergence behavior of the overall cut loop heavily depends also on the strategy for generating the next point to cut. As a matter of fact, the famous ellipsoid method has a very good (theoretical) convergence property, but the single valid cut it generates at each iteration can be very weak in polyhedral terms (actually, it does not even need to be violated but just tight at the separation point). This is because the role of the cut is not to let integer points emerge as vertices of the LP relaxation, but just to exclude a large subspace from further considerations. Evidently, one can design a sound cutting plane loop even with very weak/dense cuts, provided that the "Kelley's curse" is escaped and a different cutting plane scheme is adopted.

In the LP relaxation of our MILP master problem, we have a very simple set of constraints in the y space (essentially, only the 0-1 bounds on the variables), and we have to minimize the convex function $\sum_{j \in J} \Phi_j(y)$. This setting is very close (in fact, identical) to the one arising in the usual Lagrangian dual

minimization (assuming a convex primal problem in maximization form), where "stabilized" approaches such as the bundle method [30] are known to outperform Kelley's one by a large margin. Thus, the implementation of stabilized cutting plane at the root node (at least) is expected to be of crucial importance, in particular when a single (possibly weak) cut is separated for each point, as it is the case of our slim master model with a single w_{sum} variable.

In our current version, we did not implement a real bundle method, but a simple in-out variant very much in the spirit of [5, 12]. At each cut loop iteration, we have two points: the optimal solution (y^*, w^*) of the current master LP (as in Kelley's method), and a stabilizing point $(\tilde{y}, \tilde{w})$ which is initialized to $\tilde{y} = (1, \cdots, 1)$ and $\tilde{w} = (\Phi_1(\tilde{y}), \cdots, \Phi_m(\tilde{y}))$ (or to $\tilde{w}_{sum} = \sum_{j \in J} \Phi_j(\tilde{y})$ in case the slim model is used).

At each step, we move $(\tilde{y}, \tilde{w})$ towards (y^*, w^*) by setting

$$(\tilde{y}, \tilde{w}) = 0.5\,(\tilde{y}, \tilde{w}) + 0.5\,(y^*, w^*)$$

and then apply our GB separator in the attempt to cut the "intermediate point"

$$\lambda\,(y^*, w^*) + (1 - \lambda)\,(\tilde{y}, \tilde{w}) + \delta(1, \cdots, 1)$$

Parameters $\lambda \in (0, 1]$ and $\delta \geq 0$ are initially set to 0.2 and 2 ε, meaning that we initially try to cut off points very close to the stabilizer. If a violated cut is found, it is statically added to the current LP. After 5 consecutive iterations in which the LP bound does not improve, parameter λ is reset to 1 and the cut loop continues. After 5 more consecutive iterations with no LP bound improvement, parameter δ is reset to 0 (so we are back to Kelley's scheme). After 5 more consecutive iterations without improvement, the procedure is aborted and all cuts with a positive slack in the final LP are removed. To speedup computation, slack cuts from the current LP are also removed at every 5-th iteration.

According to our computational experience, the above cut loop is very effective for the slim model where GB separation returns (at most) one single cut at each call, in particular for the quadratic case where the difference with respect to the straightforward Kelley's is striking.

In Figure 1 we plot the behavior of three cut loop strategies (single-thread run) on a sample instance with quadratic costs, namely, the Koerkel-Ghosh [26] instance `gs250a-1` with 250 locations and 250 clients whose optimal value is 12,633.858555. All methods are based on exactly the same GB separator, and only differ on the policy to select the point y^* to cut at each iteration. `Kelley` refers to the standard approach, while `Kelley+` adopts the $2\,\varepsilon$ trick and slightly perturbs the point to cut before invoking the GB separator. Finally, `inout` refers to our simple scheme using the stabilizer $(\tilde{y}, \tilde{w})$ as described above. Note that in the horizontal axis we had to use a logarithmic scale due to the dramatically different performance of the three methods. We consider both the fat (top subfigure) and the slim (bottom subfigure) models.

When the fat model is used (top subfigure), `Kelley` has a very poor performance: we stopped it after 15,000 sec.s (still at the root node) with a lower bound of just 1,781.035992, and more than 100,000 cuts generated. `Kelley+`

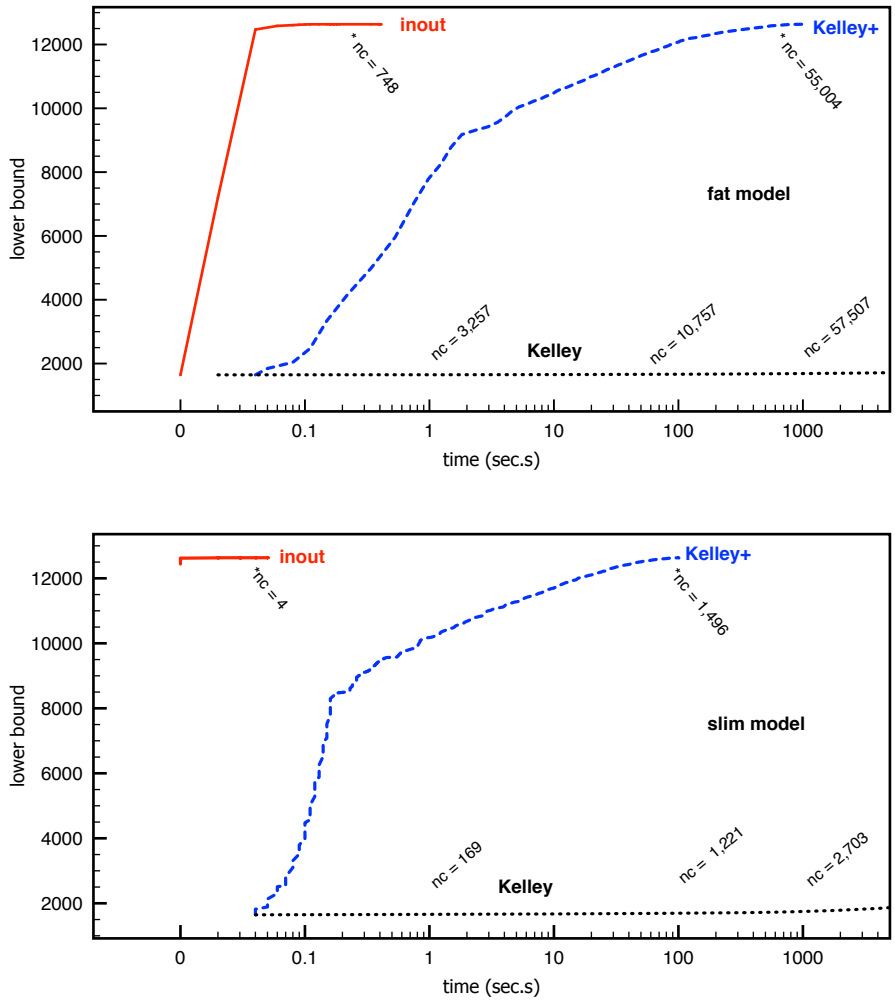

Figure 1: Performance of three cut loop strategies with fat (top subfigure) and slim (bottom subfigure) models on the sample Koerkel-Ghosh [26] instance `gs250a-1` with $n = m = 250$ (quadratic costs). We compare standard `Kelley`, `Kelley+` (i.e., Kelley enhanced by the $2\,\varepsilon$ trick), and our `inout` scheme. Label `*nc` reports the number of generated cuts at the end of the root node. Time axis given in logarithmic scale.

has a much better performance than `Kelley`, showing the importance of the $2\,\varepsilon$ trick. Its root node takes 948.86 sec.s to generate 55,004 cuts, and produces a very tight bound of 12,633.118973. Enumeration to prove optimality takes 48 branching nodes and is completed at time 978.53. The performance of `inout` is however dramatically better: its root node requires just 0.19 sec.s and produces a lower bound of 12,633.280331 with 748 cuts, while enumeration ends at time 0.41 after 6 nodes.

The performance differences are even more striking for the slim model (bottom subfigure). `Kelley` has again a very poor performance: we stopped it after 500 nodes (8,208.67 sec.s) with a lower bound of 1,875.111861 and 6,037 cuts generated. `Kelley+` has a better performance: its root node takes 98.27 sec.s to generate 1,496 cuts, and produces a bound of 12,632.092567; enumeration to prove optimality takes 44 branching nodes and is completed at time 100.66. The performance of `inout` is so good that its plot is barely visible in the figure: its root node requires 0.04 sec.s and produces a lower bound of 12,633.101453 after adding only 4 cuts, while enumeration ends at time 0.06 after 11 nodes.

3.3 Cut loop along the tree

At each branch-decision node, the MILP framework automatically invokes our GB cut separator (within a so-called `user cut callback`) just before branching, after having solved the current node LP and having added possible violated cuts from its internal cut pool and/or generated by internal procedures. The current LP solution (y^*, w^*) is passed to our GB cut separator, that possibly returns one or more violated cuts.

The violated GB cuts returned by our separation procedure, if any, are added to the internal cut pool and eventually to the current-node LP, which is reoptimized to provide a new solution (y^*, w^*), and the approach is iterated. Thus, the solver natively implements the Kelley cutting plane scheme, which is not necessarily the best possible option for weak/dense cuts. Implementing a more clever cut loop within Cplex is however not immediate, so we preferred to keep the default Kelley's scheme within the MILP solver.

To avoid tailing off, we imposed a limit of 20 consecutive cut loop iterations for each node (2,000 for the root node).

Although globally valid, at each node all generated cuts are added as "local cuts" as this allows the solver to remove more cuts from its pool—this turned out to be mainly useful in the case with linear costs, where lots of cuts are generated.

3.4 Slim or fat model?

According to our tests, the slim version with w_{sum} variable is a clear winner for the quadratic case. In our view, this is due to two main reasons. First of all, the stabilized cut loop of the previous subsection performs very well for the quadratic case, and allows us to compute a really tight root-node lower bound with a very small LP with just $n + 1$ variables and very few cuts. Second, the

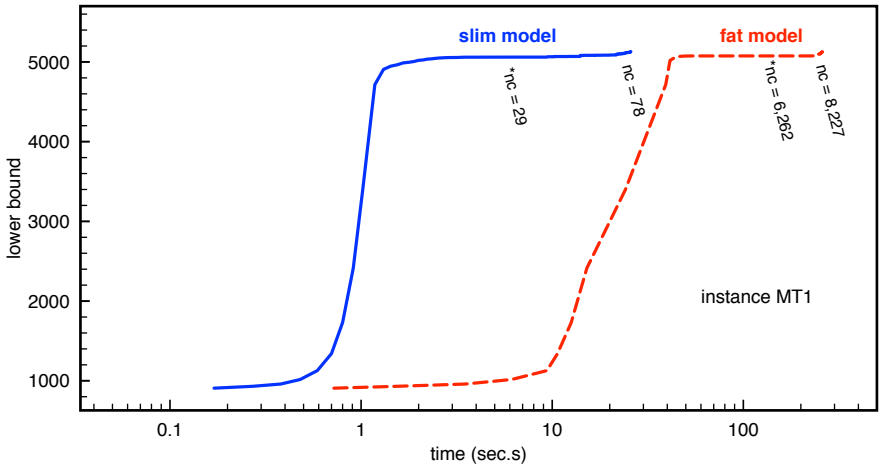

Figure 2: Comparison of the fat and slim models (quadratic costs) on the sample instance `MT1` with $n = m = 2000$ when the in-out cut loop is used at the root node. Label `*nc` reports the number of generated cuts at the end of the root node, while label `nc` gives the same information at the end of the branch-and-cut algorithm. Time axis is in logarithmic scale.

lower bound is so tight that very few branching nodes are generated, so the fact that the poor Kelley's cut loop is applied at the non-root nodes is not really an issue.

On the contrary, for the linear case the fat version with the individual w_j's variables turns out to be much better. A main reason is that the root-node lower bound is not really tight, so a considerable amount of nodes are enumerated anyway. So Kelley's curse prevents an effective recomputation of the node bounds when a single cut at the time is generated (w_{sum} version), while it performs reasonably well when several cuts are generated (version with individual w_j's). Another reason is that the stabilized cut loop at the root with a single cut at the time (w_{sum} version) is more effective when the convex function to optimize is smooth, which is true only for the quadratic case—while in the linear case Benders's cuts have a discrete nature.

In Figure 2 we plot the behavior of the fat and slim models on the sample instance `MT1` with $n = m = 2000$ and quadratic costs and 4-thread run. At the root node, our in-out cut loop scheme is adopted in both cases. Note that times are reported in logarithmic scale.

3.5 Optimality cuts for integer solutions

Within the MILP branch-and-cut framework, master solutions $\tilde{y}$ with integer $\tilde{y}_i$'s can be generated by primal heuristics or when the current-node solution

happens to be integer. In all cases, before updating the incumbent the solution needs to be checked for validity, hence it is passed to a so-called `lazy cut callback` that certificates its validity, or returns one or more valid cuts that prevent the incumbent update. In the latter case, the violated cuts could be obtained by applying our GB separator, that however can return several cuts in case the model with individual w_j's is used, thus overloading the cut pool with cuts that are likely be active only at the current point $\tilde{y}$.

Instead, we found it is computationally more efficient to generate a single *optimality cut for integer solutions* within the `lazy cut callback`, namely

$$w_{sum} \geq (\sum_{j \in J} \Phi_j(\tilde{y}))\ (1 - \sum_{i \in I : \tilde{y}_i = 0} y_i) \tag{43}$$

stating the obvious property that an allocation cost smaller than $\sum_{j \in J} \Phi_j(\tilde{y})$ can only be obtained by opening one or more additional facilities.

As the cut involves the w_{sum} variable, in the model with individual w_j's we added variable w_{sum} to the master together with its defining equation $w_{sum} = \sum_{j \in J} w_j$.

Of course, GB cuts are still generated within the `user cut callback` for cutting the fractional optimal LP solutions at the various branching nodes.

3.6 Primal heuristics

For the linear case we implemented the following primal heuristics:

a) at each branching node, within the so-called `heuristic callback`, a simple rounding heuristic is applied. Given the current LP solution y^*, we consider all possible thresholds $\theta \in \{y^*_i : i \in I\}$, in increasing order, round down all y^*_i's below θ and up all the other y^*_i's, and evaluate the cost of the integer solution found. By using a parametric technique for cost recomputation (working for the linear case only), this approach just requires $O(\sigma \log \sigma + \sigma\, m)$ time, where σ is the number of nonzero entries in y^*, so it is very fast.

b) local branching: right after the cut loop at the root node, we apply the local branching heuristic [10] with neighborhood radius starting from $k = 5$ and then increased to 10, using a small time limit for each call (5000 ticks, corresponding to approximately 5 sec.s on our hardware). The heuristic is aborted when no improved solution is found in the neighborhood of size 10.

c) proximity search: right after local branching, we apply proximity search heuristic [11] until no improved solution can be found within the small time limit imposed for each call (5000 ticks).

As to the quadratic case, according to our experience the performance of our method is so good that there is not really need to design specific primal heuristics. So for the quadratic case only the rounding heuristic a) above is applied, with a fixed threshold $\theta = \max\{y^*_i : i \in I\} - 0.2$.

3.7 Numerical tolerances and cut validity

In our implementation, we were very conservative and used very small tolerances for numerical and integrality tests. To be specific, we set Cplex's integrality tolerance `CPX_PARAM_EPINT` to 0, and the optimality/violation tolerances `CPX_PARAM_EPGAP` and `CPX_PARAM_EPRHS` to 10^{-9}. Our own internal numerical tolerance was set to 10^{-9} as well.

To assert the validity of the GB cuts we generate in our code, we implemented the following check in the spirit of Margot's proposal [35]. At the end of the run (or at the time limit), we fix all (binary and continuous) variables to their value in the incumbent solution, and enter a cut loop where (1) we apply our GB cut separator to a point y^* obtained from the incumbent by a small random perturbation (applied to 80% random entries) ranging from 10^{-9} to 10^{-1}, (2) we statically add the generated cuts to the current MILP, and (3) we optimize the MILP to verify its feasibility. If the current MILP becomes infeasible, we write down the current model in a file for further analysis and report a failure, otherwise we repeat for 10,000 times. The above check was applied extensively during the development of our code, and helped up in detecting and correcting some tolerance issues present in the earlier versions. Needless to say, our final code passed the test for all instances we tried.

4 Computational results

In this section we report on our computational experience on a subset of most difficult instances for UFL. As to qUFL, we consider instances used in the previous literature, extended by a family of much larger instances that cannot be approached by existing methods (mainly due to the fact that the underlying models would consist of millions of variables and constraints). The computational study is conducted on a cluster of identical machines each consisting of an Intel Xeon E3-1220V2 CPU running at 3.10 GHz, with 16GB of RAM each. Reported times reported are wall-clock seconds and refer to 4-thread runs.

We report computational results with basic parameter settings, without tuning our code with respect to proximity search and/or local branching parameters that could theoretically further improve the obtained results.

4.1 Benchmark instances

The set of UFL benchmark instances used in this paper stems from the UFLLIB [23], which is a well-established library of instances for capacitated and uncapacitated facility location problems. The library is a diverse collection of bipartite graphs, some of them being just small and easily solvable cases. In our study on UFL, we focus on a subset instances from UFLLIB representing the most challenging ones even for the most recent state-of-the-art approaches, like the ones proposed by [32, 38, 4]. These are randomly generated instances M* (proposed in [27]) and KG (proposed in [16, 26]). Instances M* are of size 100×100 up to 2000×2000, whereas KG instances can be divided into three groups, with

$n = m \in \{250, 500, 750\}$. Within each KG group, there are two classes, symmetric and asymmetric ones, denoted by gs* and ga*, respectively. Additionally, each class contains three subclasses, "a", "b" and "c", representing different cost settings: in "a", allocation costs are an order of magnitude higher than the facility opening costs; in "b", these costs are of the same order; and in "c", facility opening costs are an order of magnitude higher than the allocation costs. As we will report below, these differences in the costs structure significantly influence their computationally difficulty.

For testing the impact of Benders decomposition to the separable quadratic case, we consider two families of benchmark instances: (1) UFLLIB instances mentioned above plus the instances from the ORLIB (with original allocation costs equal to 0 being replaced by 10^{-5}), and (2) randomly generated instances used in previous computational studies in [6, 19, 20]. The latter instances are randomly generated graphs with potential facility and customer locations being placed uniformly at random within a unit square, with allocation costs calculated as the Euclidean distance multiplied by a factor of 50, and facility opening costs generated uniformly in $[1, 100]$. Tables shown in this section and in the Appendix report the total running time in wall-clock seconds ($t[s]$), the time needed to solve the LP-relaxation at the root node ($t_{root}[s]$), the total number of branch and bound nodes needed to prove the optimality ($nodes$) and the percentage gap at the root node ($g_r[\%]$) and the bound at the root node ($rootbound$).

4.2 Linear costs

For solving UFL, we opted for the branch-up-first setting (Cplex's parameter `CPX_PARAM_BRDIR` set to 1), as this tends to produce branching trees with fewer open nodes—hence reducing the overhead incurred when writing node files.

Table 1 summarizes results for the previously unsolved UFL instances for which we were able to prove optimality. In total, optimal solutions are provided for seven previously unsolved instances.

Two of them belong to the benchmark set of 18 instances proposed by Barahona-Chudak [2]. The semi-Lagrangian approach by [4] solved 16 of them, whereas the approach by [38] managed to solve only 8 of these 16. We provide the optimal values for the remaining two unsolved instances, namely, 2500-10 and 3000-100, of size 2500×2500 and 3000×3000, respectively.

Concerning the 30 instances of size 250×250 proposed by Koerkel-Gosh [26], 27 were solved by [38], and an additional one (ga250a-1) was solved by [4]. We now provide optimal values for the two previously unsolved instances, namely, ga250a-3 and ga250a-5. Among the KG instances of size 500×500, optimal values were known only for 7 (out of 10) instances of the subclass g$*$500c$*$. We were able to prove the optimality for the missing 3 instances from this subclass, namely ga500c-5 and gs500c-3 and gs500c-5.

The 50 KG instances of subclasses g$*$500a, g$*$500b, g$*$750$*$ still remain out of reach for existing exact methods. However, we managed to improve the best known upper bounds for 22 of these instances. To this end, we slightly

inst.	bestknown	opt	$t[s]$	rootbound	$t_{root}[s]$	$g_r[\%]$	nodes
ga250a-3	257985	257953	493.49	257554.773407	12.77	0.15	200184
ga250a-5	258225	258190	585.93	257790.245068	9.65	0.15	229446
ga500c-5	621313	621313	9226.86	601500.282332	12.31	3.19	195191
gs500c-3	621204	621204	11448.19	601980.526816	13.44	3.09	194657
gs500c-5	623180	623180	26828.91	603115.401650	14.20	3.22	270147
2500-10	3101800	3099907	824.76	3097480.189279	104.67	0.08	1362
3000-100	1602335	1602154	225.25	1601733.816607	82.67	0.03	441

Table 1: Previously unsolved UFL instances solved to optimality using our approach (linear costs).

modified the initial local-branching heuristic described in Subsection 3.6 by removing the 5000-tick timelimit for each subMIP solution. We implemented two local branching variants: in variant (A), the neighborhood radius is 5 or 10, while in variant (B) the radius is 2, 4, 6, 8 or 10. For both versions, if no improved solution can be found in the neighborhood of size 10, we randomize the incumbent and start again from the smallest radius. To better exploit the 4-core architecture of our PC's, we concurrently ran 4 times our algorithm in 1-thread mode, with 4 different input random seeds, and collected the best solution found. Our results are summarized in Table 2, and refer to a timelimit of 600 and 3600 wall-clock sec.s for each instance, respectively. Column *bestknown* gives the previous best upper bound from the literature, while UB is the heuristic value we computed within the given time limit. The remaining columns report a 0/1 flag saying whether we strictly improved (*win*) or matched (*tie*) the previous best known solution. Finally, columns under the *Best* header refer to the best of the two 3600-sec.s runs with variants (A) and (B). According to the table, 600 sec.s are enough for variant (A) to strictly improve 14 (and match 13) best-known solutions, while in 3600 sec.s we strictly improved 19 (and matched 21) solutions. Running variant (B) for 3600 more sec.s slightly improved our results, and leads to 22 strictly improved (and 22 matched) solutions. For the 6 instances (out of 50) where our UB is worse that *bestknown*, the gap between these two values is always below 0.04%.

4.3 Separable quadratic costs

For testing qUFL, we first report results obtained on a set of smaller randomly generated instances created according to the procedure described in [19]. Table 3 reports the average values over 10 instances generated with the fixed number of facilities and customers (shown in columns n and m, respectively). We compare our slim and fat models against the perspective reformulation proposed in [20]. The latter model was solved by Cplex after setting parameter `CPX_PARAM_MIQCPSTRAT` to 1, meaning that a QCP relaxation is solved at each node—this setting turned out to be much better than the default one where cone cuts are generated.

The obtained results clearly demonstrate the power of Benders decomposition for quadratic separable objective functions. Recall that the main bottleneck of the perspective reformulation of [20] is its $O(n \cdot m)$ number of variables and constraints. Table 3 shows that both our formulations scale extremely well with the increasing size of the input data. All problem instances with up to 250 facilities and 250 customers shown in this table could be solved within a fraction of a second. Compared to the performance of the perspective reformulation, our models allow for computational speedups of up to four orders of magnitude. For $n, m \geq 200$, the perspective reformulation already hits the memory limit, and it is even impossible to solve the QP/QCP relaxation at the root node.

In our next experiment, we decided to push our decomposition approaches to their limits, and for that purpose we created a set of much larger instances following the same graph generation procedure used for the instances shown in Table 3. To this end, we considered input graphs with $n \in \{500, 1000, 2000\}$ and $m \in \{500, 1000, 5000, 10000\}$. Table 4 shows the comparison of the performance of our slim and fat models. We notice that fat model is outperformed by slim model, and that the difference in the running times increases with the increasing number of customers. For example, the fat model is only a few times slower than slim for $m \leq n$, but for the larger values of m, slim model is significantly faster than its fat counterpart. Solving even the largest instances of this group (with 2000 facilities and 10000 customers) using our slim approach requires only about 5 minutes on average. The quality of the root gap (that is computed by our in-out algorithm followed by the usual root-node processing) is quite remarkable—the average root gap is consistently below 0.04%. The small difference in the quality of the LP-relaxation gaps between the two models can be explained by tailing off. Comparing the time required to solve the LP-relaxation at the root node, we observe that our both models spent most of their computing time at the root node (except for the largest instances). The required number of branch-and-bound nodes remains quite moderate for all instances, except for the largest ones, where its average value exceeds 10000.

To our big surprise, even the most difficult UFLLIB instances (e.g., those from KG of size 500×500 and 750×750), are easily solvable as qUFL's by our slim model. More precisely, all of M and KG instances can be solved to optimality in about half a minute or much less. The most difficult instance appears to be MT1 (of size 2000×2000) for which our slim model requires just 81.99 seconds. As a comparison, the same instance requires 4889.77 sec.s and 10675 branching nodes when given on input to our UFL code (linear case). This behavior is of course explained by the fact that, for a same input, the lower bounds are typically much tighter for qUFL than for (linear) UFL, so much fewer branching nodes are required.

A summary of the obtained results on KG instances (containing average values for each subclass of five instances) is shown in Table 5. More detailed results are reported in the Appendix.

		Local Branching (A)						Local Branching (B)			Best		
		600 seconds			3600 seconds			3600 seconds			7200 seconds		
instance	bestknown	UB	win	tie	UB	win	tie	UB	win	tie	UB	win	tie
ga500a-1	511422	511401	1	0	511383	1	0	511388	1	0	511383	1	0
ga500a-2	511333	511288	1	0	511255	1	0	511255	1	0	511255	1	0
ga500a-3	510817	510810	1	0	510810	1	0	510810	1	0	510810	1	0
ga500a-4	511047	511008	1	0	511008	1	0	511008	1	0	511008	1	0
ga500a-5	511258	511239	1	0	511239	1	0	511239	1	0	511239	1	0
ga500b-1	538060	538656	0	0	538060	0	1	538060	0	1	538060	0	1
ga500b-2	537850	537850	0	1	537850	0	1	537850	0	1	537850	0	1
ga500b-3	538077	538144	0	0	537924	1	0	537924	1	0	537924	1	0
ga500b-4	537925	538038	0	0	537925	0	1	537925	0	1	537925	0	1
ga500b-5	537482	537642	0	0	537482	0	1	537482	0	1	537482	0	1
gs500a-1	511229	511201	1	0	511188	1	0	511188	1	0	511188	1	0
gs500a-2	511179	511179	0	1	511179	0	1	511179	0	1	511179	0	1
gs500a-3	511120	511129	0	0	511112	1	0	511112	1	0	511112	1	0
gs500a-4	511137	511137	0	1	511137	0	1	511137	0	1	511137	0	1
gs500a-5	511293	511293	0	1	511293	0	1	511293	0	1	511293	0	1
gs500b-1	537931	537941	0	0	537931	0	1	537931	0	1	537931	0	1
gs500b-2	537763	537823	0	0	537763	0	1	537763	0	1	537763	0	1
gs500b-3	537874	538095	0	0	537926	0	0	537854	1	0	537854	1	0
gs500b-4	537742	537779	0	0	537742	0	1	537779	0	0	537742	0	1
gs500b-5	538270	538270	0	1	538270	0	1	538270	0	1	538270	0	1
ga750a-1	763576	763537	1	0	763537	1	0	763528	1	0	763528	1	0
ga750a-2	763674	763679	0	0	763653	1	0	763674	0	1	763653	1	0
ga750a-3	763765	763748	1	0	763697	1	0	763699	1	0	763697	1	0
ga750a-4	764033	764043	0	0	763945	1	0	763976	1	0	763945	1	0
ga750a-5	763905	763857	1	0	763794	1	0	763786	1	0	763786	1	0
ga750b-1	796480	796506	0	0	796454	1	0	796454	1	0	796454	1	0
ga750b-2	796056	796003	1	0	795963	1	0	795963	1	0	795963	1	0
ga750b-3	796130	796439	0	0	796384	0	0	796359	0	0	796359	0	0
ga750b-4	797080	797013	1	0	797013	1	0	797013	1	0	797013	1	0
ga750b-5	796387	796549	0	0	796549	0	0	796549	0	0	796549	0	0
ga750c-1	902026	902026	0	1	902026	0	1	902026	0	1	902026	0	1
ga750c-2	899651	899732	0	0	899651	0	1	899732	0	0	899651	0	1
ga750c-3	900010	900019	0	0	900019	0	0	900019	0	0	900019	0	0
ga750c-4	900044	900044	0	1	900044	0	1	900044	0	1	900044	0	1
ga750c-5	899235	899235	0	1	899235	0	1	899235	0	1	899235	0	1
gs750a-1	763671	763683	0	0	763683	0	0	763671	0	1	763671	0	1
gs750a-2	763548	763590	0	0	763552	0	0	763548	0	1	763548	0	1
gs750a-3	763764	763759	1	0	763748	1	0	763727	1	0	763727	1	0
gs750a-4	763887	763942	0	0	763932	0	0	763922	0	0	763922	0	0
gs750a-5	763616	763614	1	0	763614	1	0	763616	0	1	763614	1	0
gs750b-1	797026	797688	0	0	797347	0	0	797329	0	0	797329	0	0
gs750b-2	796170	796498	0	0	796170	0	1	796170	0	1	796170	0	1
gs750b-3	796589	796589	0	1	796589	0	1	796589	0	1	796589	0	1
gs750b-4	796734	797020	0	0	797020	0	0	797087	0	0	797020	0	0
gs750b-5	796365	796365	0	1	796365	0	1	796365	0	1	796365	0	1
gs750c-1	900454	900454	0	1	900454	0	1	900363	1	0	900363	1	0
gs750c-2	897886	897886	0	1	897886	0	1	897886	0	1	897886	0	1
gs750c-3	901714	901947	0	0	901786	0	0	901656	1	0	901656	1	0
gs750c-4	901339	901239	1	0	901239	1	0	901239	1	0	901239	1	0
gs750c-5	900216	900216	0	1	900216	0	1	900216	0	1	900216	0	1
sum			14	13		19	21		20	22		22	22

Table 2: Local branching for UFL on the 50 unsolved KG instances (linear costs). We strictly improved 22 (and matched 22 more) best known values from the literature.

		Our slim model				Our fat model				Perspective reformulation [20]			
n	m	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes
10	30	0.01	0.21	0.01	0.0	0.01	0.26	0.00	0.0	1.97	0.00	1.79	3.6
10	50	0.01	0.17	0.01	0.0	0.03	0.17	0.02	0.0	3.59	0.00	3.25	4.4
10	100	0.03	0.30	0.02	2.5	0.09	0.22	0.05	1.8	7.23	0.00	6.21	6.2
10	200	0.03	0.25	0.02	2.0	0.23	0.22	0.13	3.6	16.50	0.00	14.23	6.1
20	30	0.03	0.49	0.01	2.9	0.03	0.42	0.02	2.8	5.20	0.16	4.38	6.1
20	50	0.03	0.39	0.02	3.1	0.05	0.32	0.03	3.0	8.21	0.04	6.98	6.1
20	100	0.04	0.31	0.02	4.5	0.12	0.28	0.06	4.6	19.96	0.19	16.00	8.2
20	200	0.05	0.18	0.03	5.0	0.21	0.15	0.11	5.6	32.91	0.18	23.12	8.7
30	30	0.03	0.38	0.01	1.5	0.03	0.27	0.01	1.0	11.12	0.01	8.75	7.1
30	50	0.03	0.29	0.02	1.4	0.04	0.17	0.03	1.0	17.45	0.00	15.41	3.6
30	100	0.05	0.21	0.03	2.5	0.10	0.22	0.06	2.7	27.51	0.18	19.66	7.1
30	200	0.07	0.25	0.05	4.8	0.26	0.23	0.15	5.6	69.49	0.21	39.58	9.8
40	30	0.03	0.38	0.02	1.8	0.03	0.39	0.01	1.9	17.58	0.00	14.24	5.8
40	50	0.04	0.24	0.02	2.9	0.05	0.22	0.03	3.2	25.52	0.01	20.33	4.9
40	100	0.05	0.22	0.03	3.8	0.12	0.16	0.06	3.8	53.79	0.19	35.76	9.2
40	200	0.09	0.14	0.06	6.1	0.27	0.14	0.12	5.6	104.94	0.16	56.04	10.1
50	50	0.04	0.14	0.03	1.6	0.05	0.13	0.03	1.6	41.08	0.03	33.04	4.4
50	100	0.06	0.13	0.04	2.6	0.10	0.11	0.06	2.7	88.09	0.15	50.02	8.4
50	200	0.11	0.13	0.08	6.7	0.29	0.12	0.16	7.5	159.82	0.14	71.45	11.0
60	50	0.05	0.29	0.03	3.1	0.06	0.27	0.03	2.6	56.64	0.26	37.71	6.4
60	100	0.06	0.12	0.04	1.6	0.10	0.10	0.06	1.5	99.44	0.16	54.41	6.6
60	200	0.12	0.11	0.09	4.6	0.25	0.11	0.15	5.2	280.67	0.14	113.92	15.4
70	30	0.05	0.28	0.03	2.8	0.04	0.26	0.03	2.4	47.65	0.37	25.32	10.5
70	50	0.06	0.23	0.04	3.1	0.06	0.21	0.03	2.7	79.52	0.25	49.09	6.7
70	100	0.09	0.23	0.07	4.3	0.14	0.20	0.09	4.2	186.35	0.27	81.57	10.1
70	200	0.09	0.04	0.08	0.8	0.19	0.03	0.13	1.4	293.19	0.04	176.88	5.9
80	30	0.05	0.22	0.03	1.7	0.05	0.16	0.03	1.1	59.25	0.39	36.57	6.9
80	50	0.08	0.36	0.05	5.7	0.07	0.34	0.04	5.5	108.71	0.57	42.71	13.5
80	100	0.10	0.21	0.08	5.9	0.14	0.21	0.08	5.3	245.21	0.28	104.64	10.0
80	200	0.14	0.13	0.11	5.2	0.27	0.14	0.16	6.4	462.30	2.06	160.77	12.0
100	100	0.23	0.21	0.19	6.6	0.16	0.20	0.10	6.0	482.12	8.63	174.57	15.3
150	150	0.24	0.17	0.19	7.8	0.32	0.16	0.20	9.0	2012.07	3.15	648.82	14.1
200	200	0.33	0.06	0.28	6.7	0.45	0.06	0.32	4.1	—	—	—	—
250	250	0.46	0.05	0.42	4.3	0.71	0.04	0.60	4.1	—	—	—	—

Table 3: Comparing our slim and fat models with the perspective reformulation [20], on a set of randomly generated qUFL instances proposed in [19, 20]. Perspective reformulation hits memory limit for $n, m \geq 200$.

		Our slim model				Our fat model			
n	m	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes
500	500	1.39	0.03	1.31	16.2	3.30	0.03	2.82	9.5
500	1000	3.02	0.03	2.75	54.7	8.90	0.03	7.81	20.8
500	5000	11.59	0.01	10.41	87.2	132.89	0.02	127.27	32.4
500	10000	36.98	0.01	22.09	558.2	673.93	0.02	646.97	106.5
1000	500	3.80	0.04	3.32	76.0	4.60	0.04	3.86	26.1
1000	1000	5.78	0.03	5.25	65.3	15.18	0.03	13.74	28.2
1000	5000	20.70	0.01	19.32	44.3	193.76	0.02	181.87	180.3
1000	10000	64.01	0.01	34.74	603.0	799.02	0.02	748.56	399.8
2000	500	6.73	0.03	6.10	66.7	8.95	0.03	7.83	29.8
2000	1000	14.86	0.02	12.72	194.4	35.41	0.02	32.65	65.9
2000	5000	115.09	0.01	42.07	1649.0	405.85	0.02	361.69	629.3
2000	10000	309.36	0.01	76.88	10735.8	2646.69	0.03	1246.60	13114.0

Table 4: Comparing the performance of slim versus fat model on a larger set of benchmark instances for qUFL generated as in [19, 20].

5 Conclusions

The Uncapacitated Facility Location (UFL) problem is one of the most famous and studied Operations Research problems. This problem can easily be formulated as a MILP, whose size is however exceedingly large for many practical applications, making the direct use of a MILP solver rather ineffective (or even impossible). As a matter of fact, the most powerful technique at present for solving large scale UFL instances is Lagrangian relaxation, that allows one to quickly compute lower bounds that are close enough to the LP relaxation ones. The implementation of a sound Lagrangian relaxation method is however far from trivial, and a number of sophisticated ideas need to be implemented to get satisfactory results. This was the main motivation for us to consider a simpler approach built on top of an off-the-shelf MILP solver.

We therefore decided to investigate a MILP approach based on Benders decomposition, a technique that can be considered folklore but apparently not used in recent computational studies for UFL. We also addressed a nonlinear version of the problem, namely, separable convex quadratic UFL (qUFL) that has been the subject of intensive computational studies in the recent years.

From the methodological point of view, our approach uses generalized Benders cuts for convex problems, and embeds them in a branch-and-cut scheme. A number of important features are introduced, that are instrumental for the practical effectiveness of the overall approach. In particular, we discuss how to speedup the root-node cut loop through simple stabilization techniques that make it orders of magnitude faster than standard cutting plane loops.

Using our approach, we were able to solve to proven optimality 7 previously unsolved benchmark instances for UFL, and to improve the best-known heuristic value for 22 additional instances. These instances were out of reach for previous

group	$t[s]$	$g_r[\%]$	$t_{root}[s]$	nodes
ga250a	0.40	0.00	0.39	4.4
ga250b	0.28	0.03	0.22	71.2
ga250c	0.40	0.03	0.36	39.4
gs250a	0.21	0.00	0.20	3.4
gs250b	0.27	0.02	0.21	80.6
gs250c	0.46	0.03	0.42	21.6
ga500a	0.76	0.00	0.73	3.0
ga500b	2.12	0.04	1.95	58.0
ga500c	19.46	0.16	1.49	49911.6
gs500a	0.81	0.00	0.77	12.4
gs500b	2.47	0.03	2.31	72.8
gs500c	15.05	0.14	1.26	12721.6
ga750a	2.03	0.00	1.62	107.4
ga750b	2.08	0.01	1.82	65.2
ga750c	35.79	0.08	2.41	64338.0
gs750a	1.94	0.00	1.65	53.2
gs750b	3.24	0.01	1.82	414.0
gs750c	26.94	0.07	2.98	16837.0

Table 5: All KG instances for qUFL are solved to optimality by our slim model (quadratic costs). Each row shows average values over 5 instances per subclass.

MILP approaches, as the underlying models would involve tens of millions of variables and constraints, and they turned out to be too hard even for the best Lagrangian methods from the literature.

Even more interesting results are reported for qUFL: compared to previous methods, our approach enables speedups of 4 orders of magnitude or more, and allowed us to tackle much larger qUFL instances than any previous approach.

The potential impact of our work is twofold:

- On the one hand, we believe our results will motivate researchers to rethink/reinvent decomposition approaches for many optimization problems involving location, allocation and network design decisions. "Thinning out" can be done by reformulating "location subproblems" through a linear number of constraints and variables to model allocation decisions. Problems that can directly benefit from such reformulations and elimination of variables through Benders cuts are: connected facility location [17], the ring-star problem [28], the median cycle problem, or the traveling purchaser problem [29], to mention only a few.

- On the other hand, our specially designed stabilization cutting-plane schemes are of tremendous importance not only for MILPs, but also for the emerging area of convex MINLPs. Exploiting this scheme together with gener-

alized Benders decomposition and/or perspective reformulations may lead to the next boost of performance of convex MINLP solvers.

Although very closely related to UFL, application of our methods to capacitated facility location is not immediate, and will be subject of future research. We will also focus on other non-trivial MINLPs that could benefit from generalized Benders cuts, including nonlinear Stochastic Programming.

Acknowledgments

This research of the first author was 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 supported by the Austrian Research Fund (FWF, Project P 26755-N19). The work of the last author was also supported by an STSM Grant from COST Action TD1207. The authors would like to acknowledge these supports.

Appendix

Detailed computational results for both linear UFL and separable convex quadratic UFL can we found at `www.dei.unipd.it/~fisch/Appendix_UFL.pdf`

References

[1] E. Balas. A duality theorem and an algorithm for (mixed-) integer nonlinear programming. *Linear Algebra Appl.*, 4(4):341–352, 10 1971.

[2] F. Barahona and F. Chudak. Solving large scale uncapacitated facility location problems. In P. Pardalos, editor, *Approximation and Complexity in Numerical Optimization*, pages 48–62. 2000.

[3] S. Basu, M. Sharma, and P. Ghosh. Metaheuristic applications on discrete facility location problems: a survey. *OPSEARCH*, pages 1–32, 2014.

[4] C. Beltran-Royo, J.-P. Vial, and A. Alonso-Ayuso. Semi-lagrangian relaxation applied to the uncapacitated facility location problem. *Comput. Optim. Appl.*, 51(1):387–409, 2012.

[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] P. Bonami, M. Kilinc, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. *Mixed Integer Nonlinear Programming*, 154:1–39, 2012.

[7] G. Cornuejols, G. Nemhauser, and L. Wolsey. A canonical representation of simple plant location problems and its applications. *SIAM J. Algebr. Discr. Meth.*, 1(3):261–272, 1980.

[8] C. D'Ambrosio, J. Lee, and A. Wächter. A global-optimization algorithm for mixed-integer nonlinear programs having separable non-convexity. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 107–118. Springer Berlin Heidelberg, 2009.

[9] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. Thinning out Steiner trees: A node-based model for uniform edge costs. *Math. Programming Comput.*, 2015. Special Issue: 11th DIMACS Implementation Challenge on Steiner Tree Problems, submitted.

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

[11] M. Fischetti and M. Monaci. Proximity search for 0-1 mixed-integer convex programming. *J. Heuristics*, 20(6):709–731, 2014.

[12] 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.

[13] A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Math. Programming*, 106(2):225–236, 2006.

[14] H. A. Friberg. CBLIB 2014: A benchmark library for conic mixed-integer and continuous optimization. *Optimization Online*, 2014.

[15] A. Geoffrion. Generalized Benders Decomposition. *J. Optim. Theory Appl.*, 10:237–260, 1972.

[16] D. Ghosh. Neighborhood search heuristics for the uncapacitated facility location problem. *Eur. J. Oper. Res.*, 150(1):150–162, 2003.

[17] S. Gollowitzer and I. Ljubić. MIP models for connected facility location: A theoretical and computational study. *Comput. Oper. Res.*, 38(2):435–449, 2011.

[18] I. E. Grossmann and S. Lee. Generalized convex disjunctive programming: Nonlinear convex hull relaxation. *Comput. Optim. Appl.*, 26(1):83–100, 2003.

[19] O. Günlük, J. Lee, and R. Weismantel. MINLP strenghtening for separable convex quadratic transportation-cost UFL. Technical Report RC24213 (W0703-042), IBM Research Division, 2007.

[20] O. Günlük and J. Linderoth. Perspective reformulation and applications. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 61–92. Springer, 2012.

[21] H. Hijazi, P. Bonami, and A. Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS J. Comput.*, 26(1):31–44, 2014.

[22] D. S. Hochbaum and S.-P. Hong. About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Math. Programming*, 69(1-3):269–309, 1995.

[23] M. Hoefer. Ufllib, 2006.

[24] 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.

[25] A. Klose and A. Drexl. Facility location models for distribution system design. *Eur. J. Oper. Res.*, 162(1):4–29, 2005.

[26] M. Körkel. On the exact solution of large-scale simple plant location problems. *Eur. J. Oper. Res.*, 39(2):157–173, 1989.

[27] J. Kratica, D. Tošic, V. Filipović, and I. Ljubić. Solving the simple plant location problem by genetic algorithm. *RAIRO-Operations Research*, 35(01):127–142, 2001.

[28] M. Labbé, G. Laporte, I. Rodríguez-Martín, and J. J. Salazar-González. The ring star problem: Polyhedral analysis and exact algorithm. *Networks*, 43(3):177–189, 2004.

[29] G. Laporte, J. R. Ledesma, and J. S. González. A branch-and-cut algorithm for the undirected traveling purchaser problem. *Oper. Res.*, 51(66):940–951, 2003.

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

[31] A. Letchford and S. Miller. Fast bounding procedures for large instances of the simple plant location problem. *Comput. Oper. Res.*, 39(5):985–990, 2012.

[32] A. Letchford and S. Miller. An aggressive reduction scheme for the simple plant location problem. *Eur. J. Oper. Res.*, 234(3):674–682, 2014.

[33] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).

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

[35] F. Margot. Testing cut generators for mixed-integer linear programming. *Math. Programming Comput.*, 1(1):69–95, 2009.

[36] M. Melo, S. Nickel, and F. S. da Gama. Facility location and supply chain management – a review. *Eur. J. Oper. Res.*, 196(2):401–412, 2009.

[37] M. Patriksson and C. Strömberg. Algorithms for the continuous nonlinear resource allocation problem—new implementations and numerical studies. *Eur. J. Oper. Res.*, 243(3):703–722, 2015.

[38] M. Posta, J. A. Ferland, and P. Michelon. An exact cooperative method for the uncapacitated facility location problem. *Math. Programming Comput.*, 6(3):199–231, 2014.

[39] V. Verter. Uncapacitated and capacitated facility location problems. In H. A. Eiselt and V. Marianov, editors, *Foundations of Location Analysis*, volume 155 of *International Series in Operations Research & Management Science*, pages 25–37. Springer, 2011.