European Journal of Operational Research 313 (2024) 937–953

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A Huff-like location model with quality adjustment and/or closing of existing facilities☆

Boglárka G.-Tóth[a,*], Laura Anton-Sanchez[b], José Fernández[c]

[a] Dpt. Computational Optimization, University of Szeged, Árpád square 2. 6720-Szeged, Hungary
[b] Center of Operations Research, Miguel Hernandez University of Elche, Avda. de la Universidad, s/n, 03202-Elche (Alicante), Spain
[c] Dpt. Statistics and Operations Research, Faculty of Mathematics, University of Murcia, 30100-Espinardo (Murcia), Spain

ARTICLE INFO

Article history:
Received 29 April 2022
Accepted 28 August 2023
Available online 9 September 2023

Keywords:
Location
MINLP
Branch-and-bound
Heuristic
Hybrid algorithm

ABSTRACT

The problem of an expanding chain in a given area is considered. It may locate a new facility, vary the quality of its existing facilities, close some of them, or a combination of all these possibilities, whatever is the best to maximize its profit, given a budget for the expansion. A new competitive location and design model is proposed that allows all these possibilities. The resulting model is a difficult to solve MINLP problem. A branch-and-bound method based on interval analysis is proposed to cope with it. The method can solve medium-size problems in a reasonable amount of CPU time. An ad-hoc heuristic and a hybrid method that usually find a near-optimal solution in a fraction of time of the exact method are also proposed. Some computational studies are presented to show the performance of the algorithms.

1. Introduction

In this research a continuous *competitive* facility location problem is investigated. This means that there are other facilities in the same region offering the same product or service and a competition exists to maximize market share or profit. See Ashtiani (2016), Drezner (1995), Drezner (2014), Eiselt, Marianov, & Drezner (2015), Hakimi (1990), Plastria (2001) and the references therein for a review of the topic. But when a chain is already operating in the area and wants to expand its presence, in addition to locating *new* facilities it may also invest in its *existing* facilities, increasing or decreasing their quality, or closing some of them. However, to our knowledge, in none of the models in the literature on competitive location the existing facilities of the locating chain are allowed to vary their quality or to be closed. That is exactly what this paper explores. The most related papers on this topic are the following ones. In Saidani, Chu, & Chen (2012) a continuous competitive single facility location model with foresight is analysed. In this model the competitors' facilities may adjust their quality, but the locating chain does not own any existing facility and an unlimited budget is assumed. A related leader-follower *discrete* model is considered in Küçükaydin, Aras, & Altinel (2012). The leader may open an unknown number of facilities, and the follower reacts by adjusting the quality of its existing facilities, closing some of them or opening some new facilities. Again, the leader is a new entrant and an unlimited budget is assumed. In both papers, the customer choice rule is probabilistic, i.e., the demand at a demand point is distributed among all the facilities according to their attraction. In the *discrete* model introduced in Drezner, Drezner, & Kalczynski (2012) a different approach is followed. It is assumed that customers patronize a facility only if they are within its 'radius of influence'. This can be interpreted as a surrogate of the quality. If customers are in the sphere of influence of more than one facility, their demand is equally divided among them. The locating chain owns some of the existing facilities, it has a given budget to expand its presence in the area, and it may open new facilities (in a discrete set of potential locations) or improve the quality (increase the radius of influence) of its existing ones. See Fernández & Hendrix (2013) for a deeper review of these papers.

Section 2 shows that the continuous location model introduced in this paper is a non-convex mixed-integer nonlinear programming (MINLP) problem. Many researchers have been working on the topic of designing and implementing practical algorithms for

☆ This research has been supported by grants from the Spanish Ministry of Economy and Competitiveness (MTM2015-70260-P), grants PID2021-122344NB-I00 and PID2021-123278OB-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF A way of making Europe", Fundación Séneca (The Agency of Science and Technology of the Region of Murcia, 20817/PI/18), Junta de Andalucía (P18-RT-1193), in part financed by the European Regional Development Fund (ERDF), project RED2018-102363-T and grant PID2019-105952GB-I00 of the Spanish Ministry of Science and Innovation. The research has also received funding from the national project TKP2021-NVA-09 and OTKA grant FK146652 of the Ministry for Innovation and Technology, Hungary with EU-ERF funds.

* Corresponding author.
E-mail addresses: boglarka@inf.szte.hu (B. G.-Tóth), l.anton@umh.es (L. Anton-Sanchez), josefdez@um.es (J. Fernández).

https://doi.org/10.1016/j.ejor.2023.08.054
0377-2217/

coping with MINLP problems during the last decades (see Belotti et al., 2013; Bonami, Klinç, & Linderoth, 2012; Burer & Letchford, 2012; Trespalacios & Grossmann, 2014 and the references therein). Some software packages are available for both convex, like AlphaECP (Westerlund & Pörn, 2002), BONMIN (Bonami et al., 2005), Knitro (Byrd, Nocedal, & Waltz, 2006) or MINOTAUR (Mahajan, Leyffer, Linderoth, Luedtke, & Munson, 2020) and non-convex MINLP problems, like ANTIGONE (Misener & Floudas, 2014), BARON (Sahinidis, 2017), COUENNE (Belotti et al., 2020) or SCIP (Vigerske & Gleixner, 2018). However, since the available packages seem not be very successful at solving the model introduced here, solution methods are also proposed in this paper.

This paper investigates up to what extent varying the qualities of the existing facilities and closing them affects the solution of the problem, both in profit and in location and quality of the new facility (in case it is open). The main contributions of this work are: (1) a new *continuous* location and design model is introduced in Section 2. The *expanding* chain has a *limited budget*, it may locate a new facility anywhere in *a given region of the plane*, *vary the quality of its existing facilities* or *close some of them*. The customer choice rule is assumed to be *probabilistic*. (2) A branch-and-bound method based on interval analysis is proposed in Section 3. It is suitable for medium-size instances and can also be applied to many other MINLP problems. (3) An ad-hoc heuristic to tackle the problem is proposed in Section 4. (4) In Section 5 a hybrid heuristic that makes use of the interval B&B method's ability to discard big areas of the search region at early stages of the iterative process, is introduced. The hybridizing idea could also be applied to other heuristic algorithms. After some computational studies in Section 6, which show the usefulness of the approaches, the paper ends in Section 7 with the main conclusions and pointing lines for future research.

2. A location and design model for firm expansion

The model for firm expansion we introduce in this paper extends the pure-location model in Fernández, Pelegrín, Plastria, & Tóth (2007), where only the possibility of opening one new facility is considered. In practice, the expanding chain has more options. It could also consider improving the quality of (some of) its existing facilities. Or downgrading some of them so as to allocate the budget invested in them to other facilities: if a facility has no competitors too close, then maybe a decrease in its quality, even though it provokes a decrease in its attraction, may not lead to a big loss of its market share; it can even win more profit due to the lower operational costs; and, even if it is worse for that particular facility, it can be better for the chain as a whole. In fact, it may be the case that some of the existing facilities are not profitable at all, and closing them and investing their budget in other facilities (or in the new one, if it is finally open) may be the best strategy.

The utility provided by a facility to a customer is measured as quality divided by (a function of the) distance, while customers are assumed to follow the probabilistic choice rule, as in Fernández et al. (2007).

The quality of a facility was first estimated as its floor area in the seminal paper by Huff (1964). Jain & Mahajan (1979) and Nakanishi & Cooper (1974) in their Multiplicative Competitive Interaction (MCI) model consider other store characteristics, whose input parameters can be estimated in turn via a least squares approach (see more on this in Nakanishi & Cooper, 1982 and Chapter 5 in Cooper & Nakanishi, 1989). In practice, in some cases, the expanding chain can choose the characteristics that determine the quality of the facilities only from a finite set of possibilities, as in modular capacity facilities (see Correia & Captivo, 2003). In such cases, the quality variables would be discrete ones. But this analysis falls outside the scope of this paper, and we will assume here those variables to be continuous, as in most of the literature. Notice that this does not affect the main purpose of the paper, i.e., to investigate up to what extent the possibility of varying the quality of the existing facilities or closing them may affect the profit obtained by the chain and its locational decision.

To introduce the formal model, let us start with the notation:

Index sets	
i	subscript for demand points, $i = 1, \dots, i_{\max}$.
j	subscript for the existing facilities, $j = 1, \dots, j_{\max}$ (the first k of them ($0 < k < j_{\max}$) are owned by the expanding chain).
Variables	
f_0	coordinates of the new facility, $f_0 = (f_0^1, f_0^2)$.
α_0	quality of the new facility.
α_j	quality of the jth existing facility owned by the expanding chain, $j = 1, \dots, k$. At present, $\alpha_j = \tilde{\alpha}_j$.
y_j	binary variable which is 0 if facility j is closed ($j = 1, \dots, k$) or not open ($j = 0$); and 1 otherwise.
ns	variables of the problem, $ns = (f_0, \alpha_0, \alpha_1, \dots, \alpha_k, y_0, \dots, y_k)$.
Parameters	
p_i	coordinates of demand point i.
w_i	annual demand at p_i.
f_j	coordinates of existing facility j.
d_{ij}	distance between p_i and f_j.
$g_i(\cdot)$	a non-decreasing (and non-negative) function.
$d_i^{\min}$	radius of the circle around p_i where the location is not allowed.
$\alpha_{\min}$	minimum quality level for the chain-owned facilities.
$\alpha_{\max}^j$	maximum quality level that f_j may have, $j = 0, \dots, k$.
$\tilde{\alpha}_j$	present quality of existing facility f_j, $j = 1, \dots, j_{\max}$.
$\tilde{u}_{ij}$	utility that p_i perceives from f_j, $i = 1, \dots, i_{\max}$, $j = k+1, \dots, j_{\max}$. In this paper, $\tilde{u}_{ij} = \tilde{\alpha}_j / g_i(d_{ij})$.
S	area where the new facility can be set up.
A_j	annualized cost corresponding to the opening of the jth facility, $j = 1, \dots, k$.
C_j	cost of closing facility f_j, $j = 1, \dots, k$.
B	annual chain's budget for all costs.
Computed values	
$d_i(f_0)$	distance between the new facility and p_i.
$u_{i0}(f_0, \alpha_0)$	utility that p_i perceives from the new facility, $u_{i0}(f_0, \alpha_0) = \alpha_0 / g_i(d_i(f_0))$.
$u_{ij}(\alpha_j)$	utility that p_i perceives from f_j, $i = 1, \dots, i_{\max}$, $j = 1, \dots, k$. $u_{ij}(\alpha_j) = \alpha_j / g_i(d_{ij})$.
$G(f_0, \alpha_0)$	annualized cost of opening the new facility f_0 with quality α_0.
$V_j(\alpha_j)$	annualized cost of varying the quality of f_j to α_j, $j = 1, \dots, k$.
$M(ns)$	annual market share captured by the chain.
$F(M(ns))$	annual sales of the chain.
$R_j(\alpha_j)$	annual operating cost of the jth facility, $j = 0, \dots, k$.
$T(ns)$	annual total cost of the chain.
$\Pi(ns)$	annual profit obtained by the chain.

Before solving the problem, facility f_j, $j = 1, \dots, k$, is already opened. This usually implies that its area can hardly be modified, which in turn means that its quality can only be increased up to a level $\alpha_{\max}^j$ from its present value $\tilde{\alpha}_j$. Hence, we will assume that $\alpha_j \in [\alpha_{\min}, \alpha_{\max}^j]$ if the facility remains open.

The annual market share captured by the expanding chain is given by

$$M(ns) = \sum_{i=1}^{i_{\max}} w_i \frac{y_0 u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{k} y_j u_{ij}(\alpha_j)}{y_0 u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{k} y_j u_{ij}(\alpha_j) + \sum_{j=k+1}^{j_{\max}} \tilde{u}_{ij}}. \quad (1)$$

There are several costs to be taken into account: the annualized cost $G(f_0, \alpha_0)$ of opening the new facility at location f_0 with quality α_0; the opening or closing costs, A_j and C_j, respectively; the quality changing cost, $V_j(\alpha_j)$, of existing facilities, $j = 1, \dots, k$; and the operating cost of all the facilities, $R_j(\alpha_j)$, $j = 0, \dots, k$. Thus, the

annual cost of the expanding chain is given by

$$T(ns) = \sum_{j=1}^{k} \Big(y_j\big(A_j + R_j(\alpha_j) + V_j(\alpha_j)\big) + (1 - y_j)C_j\Big) \tag{2}$$

$$+ y_0\Big(G(f_0, \alpha_0) + R_0(\alpha_0)\Big), \tag{3}$$

and it is required that $T(ns) \leq B$. In some cases the expanding chain may limit the expenditures that may affect its liquidity or the investment to be financed, instead of the costs.

The profit maximization problem (P) can be stated as follows:

$$\max \Pi(ns) = F(M(ns)) - T(ns) \tag{4}$$

$$\text{s.t.} \qquad f_0 \in S \subset \mathbb{R}^2 \tag{5}$$

$$d_i(f_0) \geq d_i^{\min}, \qquad i = 1, \ldots, i_{\max} \tag{6}$$

$$y_j \alpha_{\min} \leq \alpha_j \leq y_j \alpha_{\max}^j, \qquad j = 0, \ldots, k \tag{7}$$

$$T(ns) \leq B \tag{8}$$

$$y_j \in \{0, 1\}, \qquad j = 0, \ldots, k \tag{9}$$

The differentiable function $F(\cdot)$ converts the market share into sales revenues, hence $\Pi(ns)$ measures the annual profit of the expanding chain. By S we denote the region of the plane where the new facility can be located. The constraint (6), with $d_i^{\min} > 0$, is included to prevent the new facility to be located on top of a demand point. If needed, additional constraints could be added so that the feasible set delimits the feasible region as close to reality as possible. However, notice that $d_i^{\min}$ can be set arbitrarily small if no forbidden areas are to be included in the model. Constraint (7) guarantees that if a (new or existing) facility is opened ($y_j = 1$) then $\alpha_j \in [\alpha_{\min}, \alpha_{\max}^j]$, and if it is not opened ($y_j = 0$) then $\alpha_j = 0$.

The functional form of F and G should be tailored for each real problem. In practice, determining F should be an easy task taking into account the experience of the managers of the expanding chain. As for the form of G, it could be more tricky, depending on the particular type of facility to be installed and on the knowledge of the managers. Usually F is a strictly increasing differentiable function, and G should increase as the location of the new facility approaches to one of the demand points, since close to them the opening cost of the facility will be higher (the cost of buying or renting the location increases). On the other hand, G should be a nondecreasing and convex function in α_0, since the more quality we require of the facility, the higher the costs will be, at an increasing rate. Possible expressions for both functions can be found in Fernández et al. (2007). In this paper we will use $F(M(f_0, \alpha_0)) = c \cdot M(f_0, \alpha_0)$, where c is the sales revenues per unit of goods sold, and $G(f_0, \alpha_0) = G_1(f_0) + G_2(\alpha_0)$, with $G_1(f_0) = \sum_{i=1}^{i_{\max}} \Phi_i(d_i(f_0))$, with $\Phi_i(d_i(f_0)) = w_i/((d_i(f_0))^{\varphi_{i0}} + \varphi_{i1}), \varphi_{i0}, \varphi_{i1} > 0$ given parameters, and $G_2(\alpha_0) = e^{\frac{\alpha_0}{\beta_0} + \beta_1} - e^{\beta_1}$, with $\beta_0 > 0$ and β_1 given values. In any case, the particular choice of functions F and G does not affect the study of the aim of the paper. $d_i(f_0)$ is a distance predicting function tailored to the road network of the area (see Fernández, Fernández, & Pelegrín, 2002).

The cost function due to the change in the quality of facility f_j from its present value $\tilde{\alpha}_j$ to α_j, $V_j(\alpha_j)$, should be non-increasing in the interval $[\alpha_{\min}, \tilde{\alpha}_j)$ and non-decreasing in $(\tilde{\alpha}_j, \alpha_{\max}^j]$, as the bigger the difference $|\alpha_j - \tilde{\alpha}_j|$, the higher the investment required to do the modifications. Furthermore, one would expect $V_j(\tilde{\alpha}_j + \gamma) > V_j(\tilde{\alpha}_j - \gamma)$ for $\gamma > 0$, since upgrading the quality is more expensive than downgrading it. And similarly to G, one would expect $V_j(\alpha_j)$ to be convex in $(\tilde{\alpha}_j, \alpha_{\max}^j]$ and in $[\alpha_{\min}, \tilde{\alpha}_j)$, since the more quality we expect from the facility the higher the costs will be, at an increasing rate. Additionally, whenever a variation is made, a fixed cost $v_j > 0$ has to be paid. This fixed cost prevents too small variations; in fact, a high v_j value would prevent any change. A possible expression for $V_j(\alpha_j)$ could be the following one:

$$V_j(\alpha_j) = \begin{cases} \frac{1}{\delta_j}\big(G_2(2\tilde{\alpha}_j - \alpha_j) - G_2(\tilde{\alpha}_j)\big) + v_j & \text{if } \alpha_j < \tilde{\alpha}_j \\ 0 & \text{if } \alpha_j = \tilde{\alpha}_j \\ G_2(\alpha_j) - G_2(\tilde{\alpha}_j) + v_j & \text{if } \alpha_j > \tilde{\alpha}_j \end{cases}$$

The parameter $\delta_j > 0$ determines how much cheaper is decreasing the quality as compared to increasing it.

As for the operating cost function $R_j(\alpha_j)$, it should be nondecreasing, although its functional form may vary depending on the particular type of facility. For simplicity, in this research a linear form is assumed, $R_j(\alpha_j) = o_j \alpha_j$, with $o_j > 0$ a given constant. However, note that in some cases a facility's operating cost may depend on other factors too, for instance, the expected demand that it is supposed to serve. Nevertheless, note again that the particular choice of functions V_j and R_j does not affect the study of the aim of the paper.

Before applying the model to a particular problem, the functional forms should be selected with the help of the managers' knowledge, and the parameters of the functions should be tailored so that the functions represent the reality as close as possible (see for instance Tóth, Plastria, Fernández, & Pelegrín, 2009b and Section 5 of Fernández, G.-Tóth, Redondo, & Ortigosa, 2019).

Model (P) reduces to the one in Fernández et al. (2007) in case of setting $B = \infty$ and fixing the variables $y_j = 1\ \forall j$, and $\alpha_j = \tilde{\alpha}_j\ \forall j > 0$.

Although we restrict ourselves in this paper to the potential location of a single facility, as in Fernández et al. (2007), the model could be extended to the location of multiple facilities, as in Tóth, Fernández, Pelegrín, & Plastria (2009a) or Redondo, Fernández, García, & Ortigosa (2009b). This new problem is also related to discrete competitive location problems with random utilities (see, for instance, Benati & Hansen, 2002; Freire, Moreno, & Yushimito, 2016; Haase & Müller, 2014; Ljubić & Moreno, 2018; Mai & Lodi, 2020) since when we fix the continuous variables, the model reduces to such a type of maximum capture problem.

Problem (P) is a MINLP problem. Therefore, it is a challenge from the optimization point of view, and global optimization tools are required to cope with it.

2.1. An example

In the quasi-real example in Tóth et al. (2009b) the pure location model of Fernández et al. (2007) was applied to the location of a hypermarket in an area around the city of Murcia, in southeastern Spain. Due to the lack of real data, in this subsection we have adjusted the new functions and parameters of the extended model in an ad hoc way to obtain 'reasonable' results. The modified and new parameter values can be found in Appendix A. In any case, notice that the aim of this subsection is to show how complex the problem is, and how the solution can change when varying the budget, and an accurate estimation of the papameters is not required for those purposes.

There are $i_{\max} = 21$ population centers (demand points) in the area. Figs. 1 and 2 show their location as a black circle (or dot) with a radius proportional to its buying power (which in turn is considered to be proportional to its number of inhabitants). The location of the new facility is forbidden in these circles. There are five hypermarkets in the area: two from the expanding chain (marked with a green cross) and three from the competitor (marked with a red diamond). In both cases, the original quality

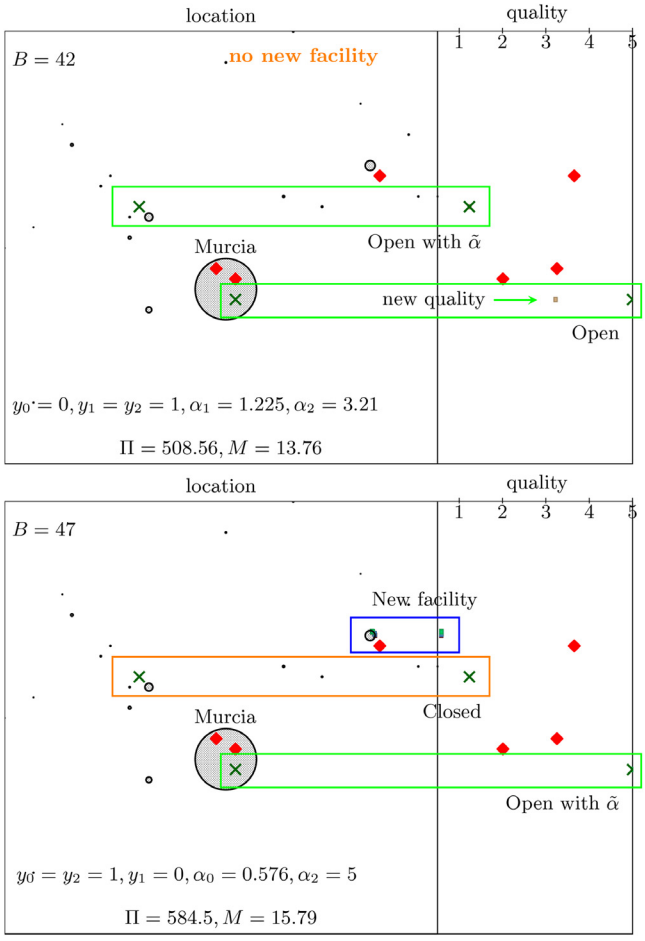

Fig. 1. Solution of the quasi-real problem of Murcia for budgets 42 (top) and 47 (bottom).

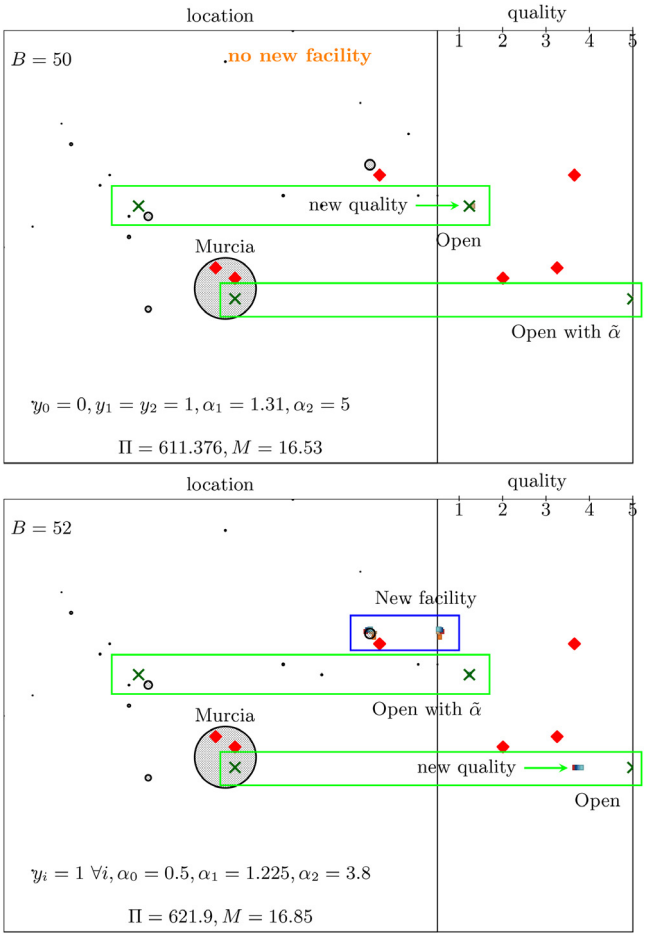

Fig. 2. Solution of the quasi-real problem of Murcia for budgets 50 (top) and 52 (bottom).

is marked on the right pane of the figure with the same symbol as the facility. The same problem has been solved with the exact method described in Section 3 for different budgets, and we can see in the pictures how the solution varies depending on the budget. The location and quality of the new facility, in case it is open, is shown inside a large blue box; 'New facility' is written on top of the box, and an interval hull for the solution boxes are shown both on the location and on the quality panes. The new qualities of the existing facilities are also shown on the quality pane by a box, and a green arrow points to it. When a facility is closed an orange box is drawn and "Closed" is written under the box.

When the budget lies in the interval [40,42), only the facility in Murcia (the most populated city) is kept open with the highest quality. The result using budget $B = 42$ is that both existing facilities are kept opened, although the quality of the facility in Murcia has to be reduced; no new facility is opened. From budget 42 to 46, the solution only changes in the quality of the facility of Murcia, increasing from 3.21 to 4.17. A major change happens when $B = 47$, see the bottom picture of Fig. 1, where the new facility is opened with minimal quality, directly competing against a competitor's facility; only the existing facility in Murcia is kept opened, with its original quality. Interestingly, this is the only budget where it is worth opening a new facility below the budget of 52. For budgets 48 to 51 the solution is not to open a new facility but to keep opened the two existing facilities with increasing qualities from the originals. From the budget $B = 52$, see the bottom picture of Fig. 2, it is again worth opening a new facility, and now all existing facilities remain open; however, the qualities of the existing facilities are decreased. When increasing the budget from 52 to 60, first the quality of the facility in Murcia is increased up to its original maximum quality of 5 (while the quality of the new facility is minimal), and from budget 56 only the quality of the new facility is increased.

We can see how tricky the problem is, where small changes in the budget may provoke completely new configurations of the solutions (as usual in MINLP problems).

3. An exact interval branch-and-bound algorithm

To exactly solve non-convex multimodal problems when also integer variables are present, global optimization methods are needed. Spatial Branch-and-Bound methods (Belotti, Lee, Liberti, Margot, & Wachter, 2009) are among the most used algorithms. They are similar to nonlinear B&B methods used for convex MINLP problems (see for instance Section 3.1 in Belotti et al., 2013). However, the relaxed problems in convex MINLP are convex NLP problems which can be solved with local techniques. Thus, branching in convex nonlinear B&B is only performed in the integer variables. On the contrary, the relaxation of a non-convex MINLP is a non-convex NLP, being themselves also hard-to-solve problems. Thus, branching is also required in the continuous variables as part of the B&B process to solve them.

In this study, we design a B&B method based on interval analysis tools. Unlike real analysis, which works with real numbers, interval analysis works with compact intervals (of real numbers). One of its main advantages is that, through a clever use of its properties, it allows to compute bounds automatically, which is specially useful in the design of B&B methods

(Hansen & Walster, 2004; Kearfott, 1996; Ratschek & Rokne, 1988). Interval B&B methods were successfully applied for solving many types of continuous NLP problems, including facility location problems (Fernández & Pelegrín, 2001; Redondo, Fernández, Álvarez, Arrondo, & Ortigosa, 2015; Tóth & Fernández, 2010; Tóth, Fernández, & Csendes, 2007). In this study, we design interval B&B methods to solve MINLP problems. The main idea is to work with the relaxed problem obtained when assuming that the integer variables are continuous too, and then, before rejecting some parts of the search region, we take care of the integer variables so as not to remove parts that may contain an optimum.

We will adopt the standard notation suggested in Kearfott et al. (2010) for interval analysis. Intervals will be denoted by boldface letters, and lower and upper bounds of intervals by 'underlines' and 'overlines', respectively. The width of an interval $\boldsymbol{z} = [\underline{z}, \overline{z}]$ will be denoted by $\operatorname{wid} \boldsymbol{z} = \overline{z} - \underline{z}$, and its relative width by $\operatorname{wid}_{rel} = \operatorname{wid} \boldsymbol{z} / \min\{|z| : z \in \boldsymbol{z}\}$ if $0 \notin \boldsymbol{z}$ and $\operatorname{wid} \boldsymbol{z}$, otherwise. The width of an interval vector $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)^T$ (also called a *'box'*) is given by $\operatorname{wid} \boldsymbol{z} = \max\{\operatorname{wid} \boldsymbol{z}_i : i = 1, \ldots, n\}$. The midpoint of an interval $\boldsymbol{z}$ will be denoted by $\operatorname{mid} \boldsymbol{z} = (\underline{z} + \overline{z})/2$. $\mathbb{IR}$ and $\mathbb{IR}^n$ will denote the set of intervals and n-dimensional boxes, respectively.

The main tool used in interval B&B algorithms is that of an inclusion function.

Definition 3.1. For a real function, $f : \mathbb{R}^n \to \mathbb{R}$, we call an interval function $\boldsymbol{f} : \mathbb{IR}^n \to \mathbb{IR}$ an inclusion function, if $\{f(z) : z \in \boldsymbol{z}\} \subseteq \boldsymbol{f}(\boldsymbol{z})$ holds for all boxes $\boldsymbol{z}$ within the domain of f.

The main benefit of an inclusion function $\boldsymbol{f}$ is that one can get bounds directly over any box in the domain of f. There are programming languages and libraries[1] that provide inclusion functions for the classical predeclared functions. From them, using interval arithmetic, inclusion functions for general functions can be built automatically (Tóth et al., 2007) using, for instance, the natural interval extension (Hansen & Walster, 2004). Automatic differentiation (Rall, 1981) also allows to obtain bounds for derivatives, gradients or Hessian matrices automatically.

The introduced MINLP location problem has the additional difficulty that functions $V_j(\alpha_j)$ are defined by an 'if' expression. Still, we can build an inclusion function for V_j and its derivative as follows:

$$\boldsymbol{V}_j(\boldsymbol{\alpha}_j) = \begin{cases} \frac{1}{\delta_j}(\boldsymbol{G}_2(2\tilde{\alpha}_j - \boldsymbol{\alpha}_j) - G_2(\tilde{\alpha}_j)) + v_j & \text{if } \overline{\alpha}_j < \tilde{\alpha}_j \\ \boldsymbol{G}_2(\boldsymbol{\alpha}_j) - G_2(\tilde{\alpha}_j) + v_j & \text{if } \underline{\alpha}_j > \tilde{\alpha}_j \\ \left[0, \max\left\{\frac{1}{\delta_j}(G_2(2\tilde{\alpha}_j - \underline{\alpha}_j) - G_2(\tilde{\alpha}_j)),\right.\right. & \\ \qquad \left.\left. G_2(\overline{\alpha}_j) - G_2(\tilde{\alpha}_j)\right\} + v_j\right] & \text{if } \underline{\alpha}_j < \tilde{\alpha}_j < \overline{\alpha}_j \end{cases}$$

and

$$\boldsymbol{V}'_j(\boldsymbol{\alpha}_j) = \begin{cases} -\frac{1}{\delta_j}\boldsymbol{G}'_2(2\tilde{\alpha}_j - \boldsymbol{\alpha}_j) & \text{if } \overline{\alpha}_j < \tilde{\alpha}_j \\ \boldsymbol{G}'_2(\boldsymbol{\alpha}_j) & \text{if } \underline{\alpha}_j > \tilde{\alpha}_j \\ (-\infty, +\infty) & \text{if } \underline{\alpha}_j < \tilde{\alpha}_j < \overline{\alpha}_j \end{cases}$$

where $\boldsymbol{G}_2$ and $\boldsymbol{G}'_2$ are inclusion functions for G_2 and G'_2, respectively (see also Kearfott, 1996; Pelegrín, Fernández, & Tóth, 2008).

Next, we describe the main steps of the interval B&B method we have implemented. For a generalized notation, we will denote by z the vector of the n variables, by f the objective function to be maximized, and by $g_j(z) \le 0$, $j = 1, \ldots, r$, the constraints of the problem. In our case, z is the vector ns (with dimension $n = 2k + 4$), and constraints (5)–(8) can be transformed into nonpositive form constraints to obtain functions $g_j(z)$.

[1] https://www.reliable-computing.org/intlang.html

3.1. Selection rule

The algorithm manages a list of boxes still to be processed, $\mathcal{L}_{\mathcal{W}}$, which initially consists of a single box containing the feasible region. In a given iteration k, the box to be processed is $\boldsymbol{z}^{(k)} = \arg\max\{\overline{\boldsymbol{f}}(\boldsymbol{z}) : \boldsymbol{z} \in \mathcal{L}_{\mathcal{W}}\}$, i.e., the so-called best-first strategy.

3.2. Subdivision rule

Let $\boldsymbol{z}$ be the box to be subdivided. Whenever a coordinate direction i is selected to be subdivided, we perpendicularly cut it as follows:

- If the corresponding variable z_i is continuous, then the ith component of the corresponding subboxes will be $[\underline{z}_i, \operatorname{mid} \boldsymbol{z}_i]$ and $[\operatorname{mid} \boldsymbol{z}_i, \overline{z}_i]$, respectively.
- If z_i is integer, then they will be $[\underline{z}_i, \lfloor \operatorname{mid} z_i \rfloor]$ and $[\lceil \operatorname{mid} z_i \rceil, \overline{z}_i]$.

The rest of the components of the subboxes do not change when performing this cut.

In our location problem all the integer variables are binary; thus, when splitting the box along a binary variable, the previous process just fixes the binary variable in the subboxes to 0 and 1, respectively.

Furthermore, in our location problem we also apply the following procedures:

(1) In the subbox with $y_j = 0$, we also set the variable α_j equal to 0 (the jth facility is closed or not opened). Analogously, when $y_0 = 0$ we also set $f_0^1 = 0$ and $f_0^2 = 0$.
(2) The *first time* a variable α_j, $j = 1, \ldots, k$, is selected to perform the subdivision, instead of bisecting the interval $\boldsymbol{\alpha}_j$ by its midpoint, as described above, we perform a trisection by $\tilde{\alpha}_j$, generating the three following subintervals: $[\underline{\alpha}_j, \tilde{\alpha}_j - \varepsilon]$, $[\tilde{\alpha}_j, \tilde{\alpha}_j]$ and $[\tilde{\alpha}_j + \varepsilon, \overline{\alpha}_j]$, where ε is the smallest machine number the computer can handle. This has proved to be useful because of the V_j functions. Notice this is only done the first time the variable is selected; bisection is used thereafter.
(3) A box is always subdivided perpendicularly to two coordinate directions. Hence, we always perform a tetrasection, instead of bisection, as it is commonly used in the literature. Whenever at least two integer variables can still be subdivided, we select the first two of them by their index (a measure of their "influence", e.g., cost or pseudo-cost, could be used instead). If only one integer variable is left to be subdivided, we select it, and also the widest continuous variable. If no more integer variables are to be divided, we select the two widest continuous variables for subdivision.
Because of this procedure, we have to take care of many special cases. For instance, when dividing by y_j and α_j, if α_j is selected for the first time, only 4 new subboxes are generated: for $y_j = 1$, α_j is subdivided into 3 pieces, as described above, but for $y_j = 0$ only $\alpha_j = 0$ is set.

3.3. Discarding tests

Most discarding tests have to be modified in order to deal with integer variables. Here, we will describe these tests with their modifications, if needed.

3.3.1. Feasibility test

We say that constraint $g_j(z) \le 0$ is *certainly continuously satisfied* for an interval box $\boldsymbol{z}$ if $\overline{\boldsymbol{g}}_j(\boldsymbol{z}) \le 0$, and that it is *certainly continuously unfulfilled* if $\underline{\boldsymbol{g}}_j(\boldsymbol{z}) > 0$. In case $0 \in \boldsymbol{g}_j(\boldsymbol{z})$, we cannot guarantee neither that it is certainly continuously satisfied nor it is

certainly continuously unfulfilled, thus the constraint is called *undetermined*. We call a box $\boldsymbol{z}$ to be *certainly continuously feasible* if all the constraints $g_j(z) \leq 0$, $j = 1, \ldots, r$, are certainly continuously satisfied, and *certainly continuously infeasible* when there exists at least one constraint that is certainly continuously unfulfilled. In any other case, the box $\boldsymbol{z}$ is called *continuously undetermined*.

Moreover, if all the continuous constraints are strictly fulfilled, i.e., $\overline{\boldsymbol{g}}_j(\boldsymbol{z}) < 0$, $j = 1, \ldots, r$, we call a box $\boldsymbol{z}$ *certainly continuously strictly feasible*.

We can only reject certainly continuously infeasible boxes. The feasibility test discards these boxes.

Note that for a *certainly continuously feasible* box $\boldsymbol{z}$, all the continuous constraints $g_j(z) \leq 0$, $j = 1, \ldots, r$, are satisfied for all points $z \in \boldsymbol{z}$. Therefore, all integer points in $\boldsymbol{z}$ are certainly feasible, such as the corner points of $\boldsymbol{z}$ that are integer points by the subdivision rule. This is specially useful for updating the best value found by the algorithm, and used in the cut-off test.

3.3.2. Cut-off test

Let $\tilde{f}$ be the best objective function value obtained so far by the algorithm. Any box $\boldsymbol{z}$ is discarded by the cut-off test provided $\overline{\boldsymbol{f}}(\boldsymbol{z}) < \tilde{f}$. Note that, in order to update $\tilde{f}$, (integer) feasible points are required, and they can be easily obtained from any integer assignment in certainly continuously feasible boxes.

The problem becomes more difficult when the optimal solution is on the boundary of several constraints. The budget constraint is usually binding and, when the new facility is to be opened, usually there is also at least one binding locational constraint. In these cases, it can be hard to find a feasible point, as most boxes near to the optimal solution are continuously undetermined.

3.3.3. Monotonicity test

Monotonicity can help to remove boxes where the global optimum cannot lie. It can be applied to both certainly continuously feasible and undetermined boxes. To derive these rules, let $\nabla \boldsymbol{f}(\boldsymbol{z}) = (\nabla_1 \boldsymbol{f}(\boldsymbol{z}), \ldots, \nabla_n \boldsymbol{f}(\boldsymbol{z}))^T$ be an inclusion of the gradient of the objective function f over a box $\boldsymbol{z}$. The monotonicity test can be applied when there exists a variable z_i for which the objective function is monotonous, i.e., $0 \notin \nabla_i \boldsymbol{f}(\boldsymbol{z})$. Let us denote by $\partial\mathcal{S}$ the boundary of the search region $\mathcal{S}$ (where the constraints $g_j(z) \leq 0$ are not taken into account). In our problem $\mathcal{S} = \left(S, [\alpha_{\min}, \alpha_{\max}], [\alpha_{\min}, \alpha^1_{\max}], \ldots, [\alpha_{\min}, \alpha^k_{\max}], [0,1], \ldots, [0,1]\right)$.

In continuous unconstrained problems, the facet $\boldsymbol{z}' = (\boldsymbol{z}_1, \ldots, \underline{z}_i, \ldots, \boldsymbol{z}_n)$ or $\boldsymbol{z}' = (\boldsymbol{z}_1, \ldots, \overline{z}_i, \ldots, \boldsymbol{z}_n)$, depending on whether the function is decreasing or increasing, of the box $\boldsymbol{z}$ contains the maximum. If $\boldsymbol{z}' \notin \partial\mathcal{S}$, there is another continuous box where $\boldsymbol{z}'$ is included, thus the box $\boldsymbol{z}$ can be discarded. If $\boldsymbol{z}' \in \partial\mathcal{S}$, $\boldsymbol{z}'$ is on the boundary, and the box $\boldsymbol{z}$ is narrowed to $\boldsymbol{z}'$.

However, for constrained problems with integer variables, $\boldsymbol{z}'$ has to be checked further. Having a certainly continuously feasible box $\boldsymbol{z}$ with $0 \notin \nabla_i \boldsymbol{f}(\boldsymbol{z})$ for a variable z_i, the following holds:

1. If z_i is continuous, we know that the maximum cannot be in the interior, thus the box $\boldsymbol{z}$ can either be discarded (when it is strictly continuously feasible), or narrowed to its facet $\boldsymbol{z}'$ when $\boldsymbol{z}' \in \partial\mathcal{S}$.
2. If z_i is integer, the box can be narrowed to the facet $\boldsymbol{z}'$.
   Notice that when the variable z_i is integer, the box cannot be removed since the gradient may change its sign in the open region $(\boldsymbol{z}_1, \ldots, (\underline{z}_i - 1, \underline{z}_i), \ldots, \boldsymbol{z}_n)$ or $(\boldsymbol{z}_1, \ldots, (\overline{z}_i, \overline{z}_i + 1), \ldots, \boldsymbol{z}_n)$, respectively. As in the division rule such regions with non-integer points are removed, we can interpret this as $\boldsymbol{z}'$ is on the boundary of the feasible set.

Now let us consider an undetermined box, $\boldsymbol{z}$. Depending on the constraints which are not certainly continuously satisfied by $\boldsymbol{z}$ the monotonicity test is different. In general, what is important in this case is whether the constraints in which the monotone variable z_i appears are certainly continuously satisfied in the facet $\boldsymbol{z}'$. We call such facet $\boldsymbol{z}'$ to be *feasible for* z_i. Now, if $0 \notin \nabla_i \boldsymbol{f}(\boldsymbol{z})$ for a variable z_i, we can state the following:

1. If z_i is a continuous variable, and the facet $\boldsymbol{z}'$ is feasible for z_i, then the box $\boldsymbol{z}$ can be either discarded (if $\boldsymbol{z}' \notin \partial\mathcal{S}$) or reduced to $\boldsymbol{z}'$ (if $\boldsymbol{z}' \in \partial\mathcal{S}$).
2. If z_i is an integer variable, and the facet $\boldsymbol{z}'$ is feasible for z_i, then the box can be narrowed to the facet $\boldsymbol{z}'$.

Specifically for our location problem, as the variables y_j, α_j, $j = 0, \ldots, k$ do not appear in the constraints delimiting the feasible area for the location of the new facility, it is enough to check whether the corresponding boxes certainly continuously satisfy the budget constraint. However, for the location variables f_0^1, f_0^2, all the constraints have to be checked as they appear in all the constraints.

In our location problem, the budget constraint is the one that provokes more continuously undetermined boxes, as it is usually binding in the optimal solution. When the budget is very small, many boxes can be discarded by the feasibility test, and when is high, it is the monotonicity test that becomes more efficient.

3.3.4. Projected one-dimensional Newton method

Fernández et al. (2007) introduce a one-dimensional Newton method that can be applied here on the continuous quality variables of both certainly continuously strictly feasible and undetermined boxes.

If the box is certainly continuously feasible, we can apply the method without any changes. The main idea is to run the interval Newton method considering only one variable at a time, fixing the other variables to their current interval value. Notice that in contrast to the classical interval Newton method, which is usually performed as a one-step (or one-iteration) method, the projected Newton method is run until it cannot remove any part of the box, or until it discards the box (as no local optimum is included in it) or until the width of the given variable is less than the tolerance in our stopping criterion.

Now let us consider an undetermined box, $\boldsymbol{z}$. We can also apply the projected one-dimensional Newton method to a variable z_i of $\boldsymbol{z}$ provided that all the constraints in which z_i appears are certainly continuously satisfied by $\boldsymbol{z}$.

Note that in all cases it is important to ensure that when the box $\boldsymbol{z}$ is on the boundary for variable z_i, the facet which is on the boundary cannot be removed by the Newton method and so has to be stored separately.

We have also employed the empirical rule of applying the method only to boxes whose width is less than one.

In the introduced location problem, if the box $\boldsymbol{z}$ is undetermined for some locational constraints, but strictly feasible for the budget constraint, the test can be applied to the quality variables (the objective function is concave in these variables Fernández et al., 2007).

3.3.5. Projected one-dimensional non-concavity test

Similarly to the one-dimensional Newton method, one can check the non-concavity of the objective function only for a given variable, considering the other variables fixed at their current interval values. The main advantage in both projected methods is that instead of computing the whole Hessian matrix, only one second derivative is required to be computed.

Again, we can only apply the non-concavity test to a continuous variable z_i provided that z_i does not appear in any of the constraints which are not certainly continuously satisfied by $\boldsymbol{z}$. Similarly, it is important to ensure that when the box $\boldsymbol{z}$ is on the

boundary for variable z_i, the facet which is on the boundary cannot be removed by the non-concavity test and so has to be stored separately.

3.4. Stopping rule

We have used two stopping criteria for any of the boxes. A box $\boldsymbol{z}$ is sent to the solution list $\mathcal{L}_S$ if either its size is smaller than a given tolerance, i.e., $\text{wid}(\boldsymbol{z}) < \epsilon$, or the relative width of the inclusion of its objective value is less than a given tolerance, i.e, $\text{wid}_{rel}(\boldsymbol{f}(\boldsymbol{z})) < \delta$.

4. A heuristic algorithm

The combinatorial nature of the problem, as given by the binary variables, and the fact that for each integer assignment a nonconvex NLP problem is to be solved, makes the problem extremely difficult.

In any case, if one should choose a facility to improve, the most *profitable* one could be a natural selection. Accordingly, if a facility should be chosen to be downgraded or closed, the least profitable one could be a good option. But a definition of profitability should then be given. In the heuristic introduced in this section, the profitability of an open facility is given by the ratio of the sales revenues it generates and the cost it incurs. More precisely,

$$profitab_0 = \frac{F\left(\sum_{i=1}^{i_{\max}} w_i \frac{y_0 u_{i0}(f_0,\alpha_0)}{y_0 u_{i0}(f_0,\alpha_0) + \sum_{\ell=1}^{k} y_\ell u_{i\ell}(\alpha_\ell) + \sum_{\ell=k+1}^{j_{\max}} \tilde{u}_{i\ell}}\right)}{G(f_0,\alpha_0) + R_0(\alpha_0)}$$

$$profitab_j = \frac{F\left(\sum_{i=1}^{i_{\max}} w_i \frac{y_j u_{ij}(\alpha_j)}{y_0 u_{i0}(f_0,\alpha_0) + \sum_{\ell=1}^{k} y_\ell u_{i\ell}(\alpha_\ell) + \sum_{\ell=k+1}^{j_{\max}} \tilde{u}_{i\ell}}\right)}{A_j + R_j(\alpha_j) + V_j(\alpha_j)}$$

if $j \neq 0$.

For given values of the variables, the open facilities could be ranked according to their profitability. However, notice that whenever a variable changes, the ranking may change too.

A pseudocode of the heuristic is given in Algorithm 1. Let $\check{B}$ be the available budget at any time. At the beginning, $\check{B}$ is equal to B (chain's initial annual budget) minus the total cost needed

Algorithm 1: Heuristic scheme

```
Create an empty tabu list.                                   // Tabu list of explored solution settings.
for iter := 1:numIniSol do
    if iter = 1 then
        Let ns be the solution with the current setting of chain-owned facilities.
    else
        Let ns be a feasible randomly generated solution. CheckTabuList(ns)
    if the profit Π improves with the closure of any chain-owned facility, even if B̌ > 0, then
        Close chain-owned facilities.
        CheckTabuList(ns)
    while B̌ < 0 and there are open chain-owned facilities   // If the budget is not enough for the open chain-owned facilities,
                                                             // some qualities must be reduced and/or some facilities must be closed.
    do
        GetMoreBudget
    CheckTabuList(ns)
    repeat
        if B̌ = 0                                             // If we have used all the available budget, we reduce the quality
                                                             // or close the facility with least profitability, to get some budget.
        then
            GetMoreBudget
            CheckTabuList(ns)
        Do the best option (I or II):
        // Option I:
        Improve the qualities of chain-owned facilities.
        Open a new facility (at least a budget B^min_new is required).
        Reopen facilities previously closed.
        // Option II:
        if B̌ < B^min_new and the new facility has not been opened then
            repeat
                GetMoreBudget
            until B̌ ≥ B^min_new or no chain-owned facilities are opened
        Open a new facility (at least a budget B^min_new is required).
        Improve the qualities of chain-owned facilities.
        Reopen facilities previously closed.
        CheckTabuList(ns)
    until Π does not increase or B̌ > 0
    if ns has not been discarded then
        Save ns and its previous configurations in the for-loop in the tabu list.
```

to maintain all the open chain-owned facilities with their current qualities.

The heuristic performs a multi-start search with *numIniSol* initial solutions. The first one is the current setting of the chain, that is, the new facility is not opened and the existing chain-owned facilities are opened with qualities $\tilde{\alpha}_j$, $j = 1, \ldots, k$. Other initial feasible solutions are randomly generated where the new facility is allowed to be opened and the chain-owned facilities are allowed to vary their quality or even be closed.

Let *ns* denote the current solution. In some steps of the heuristic, the function CheckTabuList checks whether *ns* is *similar* to any of the previous solutions already explored by the algorithm (which are stored in the tabu list). If so, *ns* is discarded and another random initial solution is generated (line 6), provided that the maximum number of initial solutions has not been reached. A solution is considered to be similar to another one when the value of their binary variables is the same and the difference in value in each of the continuous variables is smaller than ε. We have set $\varepsilon = 0.1$ in our implementation.

When more budget is required, the procedure GetMoreBudget (see Algorithm 2) decides whether to Reduce (Algorithm 3) the quality of the facility with least profitability or to Close (Algorithm 4) it. It is important to note that every time that a facility's quality changes or a facility is closed, we should recompute the profitability ranking, because the market share for the open facilities may change. Additionally, each time that a facility is closed, the heuristic includes a forbidden area around that facility to prevent the new facility from being located where the other one was just closed.

In each main iteration of the heuristic (lines 15–30 in Algorithm 1), the best of two options is implemented: (I) first Improve (Algorithm 5) the qualities of the chain-owned facilities distributing the available budget via a greedy strategy, and then Open (Algorithm 6) the new facility provided that the chain's profit improves and that there is enough budget for the opening, or (II) do it the other way around: first Open the new facility and then, if there is enough budget and the profit increases, Improve the qualities of the chain-owned facilities. In fact, these options are run three times each, assuming an available budget for the first procedure performed (Improve or Open, respectively) equal to $\lambda\check{B}$: firstly with $\lambda = 1$ (hence, using all the available budget), secondly choos-

Algorithm 2: Procedure: GetMoreBudget

1 Using Algorithm 3, tentatively Reduce the quality of the facility with the lowest profitability whose quality can be reduced, j_1.
2 Using a few iterations of a Weiszfeld-like method, get an estimate $\Pi^{est1}_{j_1}$ on the profit that can be achieved with the previous quality reduction.
3 Reset the solution to the one before doing Step 1.
4 Using Algorithm 4, tentatively Close the least profitable facility, j_1.
5 Using a few iterations of a Weiszfeld-like method, get an estimate $\Pi^{est2}_{j_1}$ on the profit that can be achieved with the previous closure.
6 **if** $\Pi^{est1}_{j_1} \geq \Pi^{est2}_{j_1}$ **then**
7 Choose to Reduce the quality of facility j_1.
8 **else**
9 Choose to Close down facility j_1.

Algorithm 3: Procedure: Reduce

1 Rank the open chain-owned facilities according to its profitability.
2 Let j_1 be the facility with the lowest profitability whose quality can be reduced, reduce its quality, $\alpha^{old}_{j_1}$, down to $\alpha_{j_1} := \max\{\alpha_{\min}, \alpha^{lp}_{j_1}\}$ where $\alpha^{lp}_{j_1}$ is the solution of the equation
$$profitab_{j_1} = profitab_{j_2}$$
and j_2 is the facility with the second lowest profitability.
3 **if** $\alpha^{lp}_{j_1} > \alpha^{old}_{j_1}$ **then** $\alpha_{j_1} := \alpha_{\min}$
4 **if** $j_1 = 0$ **then**
5 $\check{B} := \check{B} + G(f^{old}_0, \alpha^{old}_0) + R_0(\alpha^{old}_0) - G(f^{old}_0, \alpha_0) - R_0(\alpha_0)$
6 **else**
7 $\check{B} := \check{B} + R_{j_1}(\alpha^{old}_{j_1}) + V_{j_1}(\alpha^{old}_{j_1}) - R_{j_1}(\alpha_{j_1}) - V_{j_1}(\alpha_{j_1})$

Algorithm 4: Procedure: Close

1 Rank the open chain-owned facilities according to its profitability.
2 Close down the least profitable facility, j_1.
3 **if** $j_1 = 0$ **then**
4 $\check{B} := \check{B} + G(f^{old}_0, \alpha^{old}_0) + R_0(\alpha^{old}_0)$
5 **else**
6 $\check{B} := \check{B} + A_{j_1} + R_{j_1}(\alpha^{old}_{j_1}) + V_{j_1}(\alpha^{old}_{j_1}) - C_{j_1}$

Algorithm 5: Procedure: Improve

1 **while** $\check{B} > 0$ *and* $\alpha_j < \alpha^j_{\max}$ *for some* $j \in \{0, \ldots, k\}$ *and* Π *improves* **do**
2 Rank the open chain-owned facilities according to its profitability.
3 Select the most profitable facility whose quality can be improved, j_1.
4 **if** $j_1 = 0$ **then**
5 Improve the location f^{old}_0 and the quality α^{old}_0 of the new facility using a Weiszfeld-like method whenever the new facility is still the most profitable one. Let (f_0, α_0) be the new solution.
6 $\check{B} := \check{B} + G(f^{old}_0, \alpha^{old}_0) + R_0(\alpha^{old}_0) - G(f_0, \alpha_0) - R_0(\alpha_0)$
7 **else**
8 Determine the maximum quality that the facility j_1 can have by solving the equation
$$R_{j_1}(\alpha_{j_1}) + V_{j_1}(\alpha_{j_1}) - (\check{B} + R_{j_1}(\alpha^{old}_{j_1}) + V_{j_1}(\alpha^{old}_{j_1})) = 0$$
Improve the quality of facility j_1 using a Weiszfeld-like method whenever facility j_1 is still the most profitable one.
9 Let α_{j_1} be its new quality and $\alpha^{old}_{j_1}$ its quality before applying the procedure.
10 $\check{B} := \check{B} + R_{j_1}(\alpha^{old}_{j_1}) + V_{j_1}(\alpha^{old}_{j_1}) - R_{j_1}(\alpha_{j_1}) - V_{j_1}(\alpha_{j_1})$

Algorithm 6: Procedure: Open

1 **if** $\check{B} \geq B^{\min}_{new}$ **then**
2 Apply a Weiszfeld-like method to locate a new facility using budget $\check{B}$.
3 $\check{B} := \check{B} - G(f_0, \alpha_0) - R_0(\alpha_0)$

Algorithm 7: Procedure: REOPEN

```
                                  // B^min_any is a lower bound of the minimum budget needed to reopen any facility. We
                                  // use the annual operating cost with minimum quality of the cheapest existing facility.
while B̌ > B^min_any and not all the chained-owned facilities are opened and Π improves do
    for all closed chained-owned facilities j do
        Determine the maximum quality that facility j can have by solving the equation
                                  R_j(α_j) + V_j(α_j) + A_j − (B̌ + C_j) = 0
        Obtain the quality of facility j using a Weiszfeld-like method.
    Reopen the facility j_1 that improves Π the most.
    if Π does not improve compared to the setting before reopening facility j_1 then
        Reset the solution to its previous setting.
    else
        B̌ := B̌ + C_{j_1} − R_{j_1}(α_{j_1}) − V_{j_1}(α_{j_1}) − A_{j_1}
```

ing λ randomly within the interval [0.75,1), and thirdly within [0.5,0.75). The use of the last two values of λ ensures that there is leftover budget for the second performed procedure (OPEN and IMPROVE, respectively). In both options, if the profit increases and there is enough budget, it is possible to REOPEN (Algorithm 7) previously closed facilities.

Procedure OPEN requires at least a budget $B^{\min}_{\text{new}}$ to locate the new facility. This minimum budget should be, at least, equal to the budget required to build a facility with minimum quality at the cheapest place. Since we do not know the cheapest place, we set $B^{\min}_{\text{new}}$ equal to the budget required to build the new facility with minimum quality, ignoring the location cost. If this procedure is called when the new facility is already located, it tries to locate it randomly in a different place to get a better solution.

In Algorithms 2, 5, 6 and 7 we use a modification of the *multi-start* Weiszfeld-like algorithm presented in Redondo, Fernández, García, & Ortigosa (2009a). Weiszfeld algorithm is a steepest descent type method which takes discrete steps along the search paths. In the method, the derivatives of the objective function are equated to zero and the next iteration is obtained by implicitly solving these equations. The modification used in this paper checks that the budget constraint is satisfied. Specifically, if in an iteration i the new facility results in an infeasible solution $(f_0^{(i)}, \alpha_0^{(i)})$, then (1) we first move its location to a point in the segment $[f_0^{(i-1)}, f_0^{(i)}] \subset \mathbb{R}^2$ which is on the border of the feasible region, in case the location is infeasible, then (2) we reduce its quality, in case the cost of the facility exceeds the budget, and finally (3) if the solution is still infeasible then we move the facility along the segment $[(f_0^{(i-1)}, \alpha_0^{(i-1)}), (f_0^{(i)}, \alpha_0^{(i)})] \subset \mathbb{R}^3$ until it is feasible. Note that only the new facility can provide infeasible solutions, since we calculate the quality of existing facilities in Algorithm 5, where the available budget, $\check{B}$, is taken into account.

5. A hybrid heuristic

The heuristic algorithm described in the previous section explores all the feasible set when looking for good solutions. However, many subregions of the feasible set do not contain good solutions. In fact, the interval B&B method described in Section 3 usually discards large subregions at the early stages of the algorithm and usually spends more time trying to prove near-optimality of the non-discarded areas covered by the boxes after that moment. Hence, it could be a good idea to execute first some iterations of the interval B&B so as to discard these non-promising areas, and then to execute the heuristic algorithm only in promising areas.

This is exactly what our hybrid algorithm does. First, some iterations of the B&B method are performed. The larger the number of iterations, the greater the volume of non-promising areas discarded, but also the more CPU time is needed. A balance is needed between the volume discarded and the CPU time required, especially taking into account that after that the heuristic algorithm will be applied in each box offered as output by the B&B algorithm. For smaller instances, running the interval B&B method at most 10% of its total running time proved to give good results. However, for larger problems, when $k > 2$, the method could only remove a tiny part of the search region, even if the running time was increased. The reason is the curse of dimensionality. Thus, we have applied a non-reliable step in order to make this special pre-solve procedure useful for higher-dimensional problems: we have removed boxes that *most likely* do not contain an optimal solution. This has been done using the *pup* index (see Markót, Fernández, Casado, & Csendes, 2006 for more details), which offers an estimation of how promising a box is, taking into account both the bounds on the objective and the constraints. When the B&B method was stopped, we have calculated the *pup* index for each box, together with the minimum $pup_{\min}$ and maximum $pup_{\max}$ values. For a given $\lambda \in [0,1]$ all boxes having an index $pup < \lambda pup_{\max} + (1-\lambda) pup_{\min}$ are removed. Both the running time that the B&B should be allowed to run, as well as the value of the parameter λ, must be tuned for each type of problem. In particular, in our computational studies, after some preliminary experiments, we let the algorithm run for 10 seconds and set $\lambda = 0.5$ for the instances with 100 demand points. For larger instances (500 and 1000 demand points), the parameters depended on the number of chain-owned facilities. For $k = 1, 3$ and 5, the interval B&B was ran for 15, 50 and 300 seconds with $\lambda = 0.5, 0.6$ and 0.7, respectively.

The result of that execution of the B&B algorithm is a long list of boxes. Since executing the heuristic algorithm in each of these boxes will be extremely time-consuming, the number of boxes is reduced by joining boxes which are close enough. Specifically, for each box in the solution list, we check whether there is another box in the final list of joint boxes (initially empty) close to it. Namely, we join a box $\boldsymbol{x}$ from the solution list with a box $\boldsymbol{y}$ in the final list provided that $\sum_{i=1}^{2k+4} \max\{|\underline{x}_i - \underline{y}_i|, |\overline{x}_i - \overline{y}_i|\} < \rho \sum_{i=1}^{2k+4} (\text{wid}\, \boldsymbol{x}_i + \text{wid}\, \boldsymbol{y}_i)$. Here $\rho \in [1,2]$ is a given parameter (the higher the number of variables, the higher the value of ρ). If two boxes are close enough, then we replace the box $\boldsymbol{y}$ in the final list with the interval hull of the boxes $\boldsymbol{x}$ and $\boldsymbol{y}$ (the small-

est box containing them). If a box $\boldsymbol{x}$ is not close enough to any of the boxes of the final list, then we add it at the end of the final list. The process is repeated with the next box of the solution list. Once we finish the checking with all the boxes of the solution list, the same procedure is applied once more to the final list.

Then, the hybrid heuristic executes Algorithm 1 for each of the boxes in that reduced list, and returns the best solution found among them all. We performed some preliminary tests using (i) the same number of starting solutions, *numIniSol*/*NB*, in each box (*NB* denotes the number of boxes in the reduced list), and (ii) making a proportional distribution of the total number of initial solutions depending on the size of the box and rounding up to the nearest integer. In many of the test problems there was a very large box compared to the rest to which were assigned almost all the starting solutions; and most of the other boxes were usually very small. In order to increase the exploration on these boxes, at least one initial solution was assigned to any box. The size of a box $\boldsymbol{z}$ is obtained as follows:

$$\text{size}(\boldsymbol{z}) = \prod_{i\in\{1,\dots,2k+4\,:\,\bar{z}_i\neq\underline{z}_i\}} (\bar{z}_i - \underline{z}_i)$$

Since the boxes are usually of very different sizes, the second strategy proved to be the best, and we adopted it in the hybrid heuristic.

Some of the boxes in the reduced list comprise fixed binary variables. Since in these boxes these facilities have to remain either closed or opened, it is necessary to adapt some of the procedures of the heuristic introduced in the previous section to accommodate this fact.

For each box provided in the reduced list, the randomly generated solutions produced by the heuristic always lie within the box: if some binary variables are fixed, they keep the same value, whereas the rest of binary variables randomly take the value 0 or 1, and the continuous variables take a random value within the corresponding interval. All the random solutions are generated in this way, except the first one, which is pseudo-randomly generated as follows. For each chain-owned facility j, if y_j can be either 0 or 1, we set it to 1 (open facility). If the facility is opened and its present quality $\tilde{\alpha}_j$ is within the range of possible α_j values, we assign $\tilde{\alpha}_j$ as the quality of facililty j, otherwise we use the midpoint of the corresponding α_j interval. For the new facility, if y_0 can be either 0 or 1, we set it to 0 (the new facility is not open). In case it must be open, we assign as location and quality the midpoint of the corresponding intervals in which these variables must lie. Note that this location of the new facility could be infeasible due to the proximity of demand points or previously closed facilities.

It is worth noting that the existence of feasible points in the boxes provided by the interval B&B algorithm is not guaranteed. Due to the overestimation of the inclusions provided by the inclusion functions, it may happen that an infeasible box is declared as continuously undetermined. That is why we include in the hybrid algorithm the possibility of not applying Algorithm 1 in some boxes. Specifically, if the location of the new facility in the initial solution is infeasible, another location is randomly generated within the corresponding intervals. If after 100 attempts no feasible solution is found, Algorithm 1 is not applied. It may also happen that there exist feasible solutions in terms of location, but no feasible solution in terms of budget can be found within the intervals of the facilities' qualities. In this case we allow a maximum of 1000 attempts (randomly generating all the variables within their corresponding intervals) before not applying Algorithm 1.

6. Computational studies

6.1. Difficulty of solving the model

We implemented the model described in Section 2 in AMPL (Fourer, Gay, & Kernighan, 2003) and tried first to solve a few problems for different budgets, with all the available solvers suitable for MINLP problems callable from AMPL. Namely, we have used BARON (Sahinidis, 2017), BONMIN (Bonami et al., 2005), COUENNE (Belotti et al., 2020), IPOPT (Wächter & Biegler, 2005), Knitro (Byrd et al., 2006), LGO (Pintér, 1997), LocSol (Benoist, Estellon, Gardi, Megel, & Nouioua, 2011), LOQO (Benson & Vanderbei, 2003), SCIP (Vigerske & Gleixner, 2018) and SNOPT (Gill, Murray, & Saunders, 2005). We also solved the problems with the differential evolution (DE) heuristic (Storn & Price, 1997), as implemented in the Python package *scipy*. In DE we used all the parameters with their default value except for the maximum number of iterations: we increased it so that in practice DE only stops when it considers that it has found an optimal solution. We compared the results obtained with those provided by the interval B&B method, IBB. All methods were run on an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz with 500 GB RAM.

The first base problem is a toy one, with just $i_{\max} = 4$ demand points, $j_{\max} = 4$ existing facilities, one of them belonging to the expanding chain ($k = 1$). Only Knitro succeeded at solving some of the 10 problems with different budgets with the conditional definition of V_j. However, changing the definition of V_j using additional binary variables, BARON, SCIP and LocSol were able to solve some of the problems as well. LOQO and LGO failed in all cases by reporting infeasible solutions. For LocSol, we had to set a reasonable time limit because it is used as the stopping criterion. Setting the time limit to 1 second for this toy problem (both BARON and our interval B&B method finished in less time) LocSol found near optimal solutions in 8 out of the 10 problems. BONMIN without any good starting point, failed at solving the toy problem regardless of the budget. SCIP could solve all the toy problems optimally but, on average, it was more than 50 times slower as compared to any of the other successful methods. These results are summarized in Tables 1 and 2. We reported in percentages if the solution was found within the relative accuracy of 0.01, the average relative accuracy in objective terms, the average computational time in seconds, and also when the method reported an infeasible point as solution in percentages. In Table 2 we have collected the methods that failed to solve the problems and were left out from further consideration.

Table 1
Comparision of existing solvers for a toy problem: promising methods.

Solver	BARON	Knitro	LocSol	SCIP	DE	IBB
Found %	100	50	80	100	40	100
Avg.Rel.Diff	0.0	67.5	0.1	0.0	84.0	0
Avg.Time (s)	0.56	0.20	0.90	31.58	114.24	0.61
Not feas %	0	0	10	0	30	0

Table 2
Comparision of existing solvers for a toy problem: methods unable to solve it.

Solver	SNOPT	BONMIN	COUENNE	IPOPT
Found %	10	10	10	0
Avg.Rel.Diff	110.9	110.9	110.9	106.5
Avg.Time (s)	0.00	0.03	40.18	0.03
Not feas %	0	0	0	10

Table 3
Comparison of existing solvers for a quasi-real problem.

Solver	BARON	Knitro	LocSol	SCIP	DE	IBB
Found %	93.3	6.7	40.0	86.7	6.7	100.0
Avg.Rel.Diff	0.3	12.8	6.9	1.1	9.9	0.0
Avg.Time (s)	1324.9	0.4	271.0	19517.2	2484.3	2371.1
Not feas %	0	0	0	0	13.3	0

Table 4
Comparison of BARON with IBB for a random problem with 100 demand point.

Solver	BARON	IBB
Found %	0	100
Avg.Rel.Diff	10.0	0.0
Avg.Time (s)	3601.0	110.3
Not feas %	0	0

The second base test problem is the quasi-real example of Section 2.1, for which 15 different budgets were used. Comparing the results obtained by BARON and our method, we found that for lower budget settings, our method was able to find the enclosing interval of the global optima in 25 seconds on average, while BARON took 800 seconds on average. For the problems whose budget allows to open the new facility and also to keep open the existing facilities, the number of near-optimal solutions is very high, due to the fact that increasing the quality of one facility can be compensated (both in profit and budget) by decreasing the quality of another facility. Our IBB method aims to find all global optimal points (in fact, to enclose all near-optimal points within a given accuracy). Opposed to this, BARON aims to find just one global optimum point within the given tolerance and stops immediately. Thus, in these instances BARON finds one global optimum point faster than our method encloses all the near-optimal points. For these 4 out of 15 cases our method took 3 times more CPU time on average than BARON. When using LocSol we had to set a reasonable time limit, which was not evident, as both our IBB method and BARON took times in a wide range when solving these problems. In order to have a fair comparison, we set the time limit to 1 second for the first 5 problems, 10 seconds for the next 6, and to 1000 for the last 4. The results are summarized in Table 3. We can see that only BARON and SCIP were able to solve more than 80% of the problems, but SCIP takes almost 10 times more than BARON or IBB. The other solvers are clearly outperformed. Thus, only BARON could be a competitor for IBB.

In order to check the reliability and performance of BARON, we solved a larger instance, generated randomly. It has 100 demand points, 3 existing facilities, 2 belonging to the expanding chain (see Section 6.3 for details on the generation). Using our interval method for 5 different budgets, the enclosure of the global optima is reached in 110 seconds on average, while BARON did not find a near-optimal solution within an hour (see Table 4). For one of the budgets we let BARON run one day, but it could only find a good local solution, not the global optimum.

6.2. Hardware and software

All the computational results in the following sections have been obtained under Linux on an AMD EPYC 7642 X2 with 3200 MHz CPU and 512GB memory. The algorithms have been implemented in C++. For the interval B&B method, we used the interval arithmetic in the PROFIL/BIAS library (Knüppel, 1993), and the automatic differentiation of the C++ Toolbox library (Hammer, Hocks, Kulisch, & Ratz, 1995). We have used a time limit of 12 hours for each run.

6.3. Test problems

In order to check the performance of the methods and their variants, we have generated different test problems varying some parameters. We have initially fixed the number of demand points ($i_{\max} = 100$), but varied the number of existing facilities ($j_{\max} = 3$, 5) and the number of facilities belonging to the expanding chain ($k = 1$, 2). For each setting 10 problems were generated, by choosing the input data of the problems randomly from the following intervals using uniform distributions:

- location of demand points and existing facilities: $p_i, f_j \in ([0, 10], [0, 10])$. This was also considered as the search region for f_0
- demand: $w_i \in (0, 10]$
- quality of existing facilities: $\tilde{\alpha}_j \in [0.5, 5]$. This was also the non-zero search region for α_0
- distances are calculated by the Euclidean distance, and g_i is the square function, thus $g_i(d_i(x))$ is the squared distance
- sales revenues per unit of goods sold in $F(\cdot) = c \cdot M(\cdot)$: $c \in [12, 14]$
- parameters of function Φ_i: $\varphi_{i0} = 2, \varphi_{i1} \in [0.5, 1.5]$
- parameters of function G_2: $\beta_0 \in [7, 9]$, $\beta_1 \in [5, 5.5]$
- radius of forbidden regions: $d_i^{\min} = w_i/\rho$, where $\rho = 3\sqrt{i_{\max} + 50} - 20$
- parameter of function R_j: $o_j = 20$
- annualized cost for keeping a facility opened: $A_j \in [8, 11]$
- cost for closing a facility: $C_j = A_j/2$
- parameters of function V_j: $v_j = G_2(1)$ and $\delta_j \in [3, 5]$

After generating the instances, we calculated the budget needed to run the facilities of the expanding chain with their current setting, $\tilde{B} = T(ns)$ (i.e., when no new facility was taken into account ($y_0 = 0$) and when the existing facilities were open with their original qualities, i.e., $y_j = 1$ and $\alpha_j = \tilde{\alpha}_j$, $\forall j = 1, \ldots, k$). In order to generate problems which are closer to reality, we performed a pre-optimization for the qualities of the existing facilities, i.e., we applied our B&B method to the problem obtained by setting the budget to $\tilde{B}$ and by fixing $y_0 = 0$ and $y_j = 1$, $\forall j = 1, \ldots, k$. Finally, we set $\tilde{\alpha}_j$ equal to the optimal value of α_j of the aforementioned problem.

Furthermore, we also used $\tilde{B}$ to set reasonable budgets for the generated instances. In particular, we solved each problem instance for 80%, 100%, 120%, 150% and 200% of the budget $\tilde{B}$, that is, from a scenario in which the chain does not have enough budget to keep all its facilities with their current qualities, to a scenario where the budget of the chain doubles the one required for the current setting. Hence, in all $2 \cdot 2 \cdot 10 \cdot 5 = 200$ instances with $i_{\max} = 100$ demand points were generated. They have been used to analyze the performance of the interval B&B method and the quality of the solutions of the heuristic and hybrid algorithms.

In order to check the performance of the heuristic and the hybrid methods on large-scale instances, we also generated larger instances with $i_{\max} = 500$ and $i_{\max} = 1000$ demand points with $j_{\max} = 7$ or $j_{\max} = 10$ existing facilities varying the number of chain-owned facilities using k= 1, 3 or 5. As before, for each setting 10 problems were generated, i.e., we used $2 \cdot 2 \cdot 3 \cdot 10 \cdot 5 = 600$ large instances. In these cases we could not obtain the optimal solution with the interval B&B, so we only compared the two approximate algorithms.

6.4. About the interval branch-and-bound method

In order to choose the most efficient version of the interval branch-and-bound method, we solved the test problems with $i_{\max} = 100$ demand points using the following versions of the interval B&B algorithm:

Table 5
Average CPU time in seconds for the different versions of the interval B&B method.

j_{max}	k	Basic	Mono	$\frac{Mono}{Basic}$	Newton	$\frac{Newt}{Mono}$	NonConc	$\frac{NonConc}{Newt}$
3	1	11613	1997	17%	1987	100%	1989	100%
	2	13725	3939	29%	1695	43%	1713	101%
5	1	9360	214	2%	201	94%	203	101%
	2	14203	6911	49%	4103	59%	4128	101%

Basic: Only feasibility and cut-off tests are applied as discarding tests, and only the natural inclusion is evaluated for bounding.

Mono: The discarding tests of the Basic version together with the monotonicity test are applied. For bounding, the intersection of the centered form and natural inclusion (Hansen & Walster, 2004) is used.

Newton: This is the Mono version together with the projected one-dimensional Newton method.

NonConc: This is the Newton version together with the projected one-dimensional non-concavity test.

In Table 5 we can compare the different versions of the interval B&B method for the different problem settings. As the main indicator of efficiency, the table gives the average computational time in seconds for the 10 problems and 5 budgets in each setting. A more detailed version of the results is discussed in Appendix B. For those settings, where the given method could not finish before its time limit, the 12 hours were taken into account.

In the first two columns of Table 5, the type of problems is shown: first the number of existing facilities and then the number of facilities belonging to the expanding chain is given. In the following columns, for each version of the interval method (Basic, Mono, Newton and NonConc) the average computational time in seconds is reported. Moreover, we report in percentages the ratio between each version with its preceding variant, in the columns named $\frac{Mono}{Basic}$, $\frac{Newt}{Mono}$ and $\frac{NonConc}{Newt}$.

The use of the monotonicity test with the centered form always makes the interval B&B method faster. When only one chain-owned facility exists ($k = 1$), the monotonicity test is extremely efficient, while in case of two chain-owned facilities, there is less improvement, but still, Mono is very efficient, two times faster than Basic. The projected one-dimensional Newton method is also efficient when two chain-owned facilities exist (when the monotonicity test is less efficient). On the other hand, the projected one-dimensional non-concavity test is not efficient for our location problem.

Comparing the problem settings, the easiest problems are, in general, the ones with one chain-owned facility. The reason is clear: these problems have one variable less than the others. Interestingly, among these problems, the ones with 5 existing facilities are easier to solve; however, it is not clear if there is a reason for that. On the contrary, when there are two chain-owned facilities, it turns out that, in general, the problems with 3 existing facilities are easier to solve.

In Table 6 we have reported the average results of all the problems for each budget scenario comparing the different versions of the interval B&B method. As we can see, the budget needed to run the facilities of the expanding chain with their current setting, $\tilde{B}$, gives in general the hardest problem for each variant. For the Basic and Mono versions, the easiest problems on average are those whose budget is more restrictive; on the contrary, the Newton and NonConc variants are more effective as the budget constraint becomes looser. On average, the Mono version needs just one third of the running time of Basic. The Newton version also reduces the computational time, specially when the budget constraint is looser.

Table 6
Average computational times for each budget settings using the different versions of the interval B&B method.

Budget	Basic	Mono	$\frac{Mono}{Basic}$	Newton	$\frac{Newt}{Mono}$	NonConc	$\frac{NonConc}{Newt}$
80%	7488	2398	32%	2389	100%	2395	100%
100%	14559	4065	28%	3553	87%	3551	100%
120%	13841	3403	25%	2760	81%	2765	100%
150%	11447	3217	28%	1067	33%	1092	102%
200%	13793	3242	24%	214	7%	239	112%
Avg.	**12225**	**3265**	**27%**	**1997**	**61%**	**2008**	**101%**

In general, compared to the Mono version, it needs 61% of its CPU time, and compared to the Basic method, only 16% on average.

6.5. About the heuristics

In the heuristic (see Algorithm 1) we used $numIniSol = 10$ in all instances. In the hybrid heuristic (see Section 5) 10 initial solutions were distributed among the boxes provided by the B&B, taking into account that at least one initial solution was always used in each box. Both the heuristic and the hybrid heuristic algorithms were run 10 times for each problem, that is, for each of the 10 base instances generated for each combination of i_{max}, j_{max}, and k, considering 5 different budgets in all cases.

Table 7 summarizes the results for the instances with $i_{max} = 100$, which could be solved with the exact B&B method. It shows the averages for each setting, together with the standard deviation in brackets. The budget is shown in relative terms to $\tilde{B}$. For each problem, we calculated how many times the solution obtained by the interval B&B was found in the 10 runs by the heuristic and the hybrid heuristic methods (columns 4 and 5). We also computed the relative difference in terms of objective value for the solution obtained by the interval B&B, and the average value of the 10 solutions found by the heuristic and the hybrid method (columns 6 and 7), and also with the best solution of both algorithms in the 10 runs (columns 8 and 9). The last three columns show, for each setting, the average computational time, in seconds, spent by the Newton variant of the interval B&B, the heuristic and the hybrid heuristic, respectively. The average of each column is also reported.

As it is shown, both heuristics can find near-optimal solutions. The average number of times that the heuristic found the best solution in the 10 runs is near 5, while in the hybrid method it reaches 6.34. On average, the differences between the global optimum and the average value of the solutions found by the heuristic and the hybrid method are 0.17% and 0.11%, respectively and, in the worst case, it is only 1.07% for the heuristic and 0.51% for the hybrid method. On average, the differences between the global optimum and the best solution found decrease to 0.09% and 0.06% for the heuristic and the hybrid method, respectively. The elapsed time for both heuristic algorithms is, on average, very similar and much smaller than for the interval B&B. It must be taken into account that the computational times of the hybrid method include the time that the B&B was ran to obtain the list of boxes that it needs. Specifically, for $i_{max} = 100$ this time was 10 seconds.

Table 7
Results for the problems with 100 demand points using the heuristic and the hybrid heuristic algorithms.

j_{max}	k	Budget	Times found		Difference in obj (%) with average solution		Difference in obj (%) with the best solution		Computational time (sec.)		
			heur.	hybr.	heur.	hybr.	heur.	hybr.	B&B	heur.	hybr.
3	1	80%	4.7 (4.1)	6.7 (4.2)	0.26 (0.8)	0.07 (0.2)	0.11 (0.3)	0.02 (0.1)	3293.1 (6178.5)	8.1 (3.8)	17.7 (8.4)
		100%	5.2 (3.9)	6.2 (4.2)	0.00 (0.0)	0.01 (0.1)	0.00 (0.0)	0.00 (0.0)	4388.3 (5627.4)	8.4 (4.1)	16.5 (5.4)
		120%	4.9 (4.7)	7.6 (3.3)	0.02 (0.1)	0.03 (0.2)	0.00 (0.0)	0.00 (0.0)	1927.5 (3417.7)	10.1 (4.0)	17.4 (7.3)
		150%	5.4 (5.0)	6.1 (4.3)	0.05 (0.2)	0.10 (0.3)	0.01 (0.0)	0.10 (0.3)	174.4 (233.8)	13.0 (6.6)	20.5 (10.2)
		200%	4.2 (5.0)	6.9 (4.0)	0.01 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	152.5 (200.5)	18.9 (22.1)	19.8 (8.9)
	2	80%	5.2 (4.3)	5.6 (4.6)	0.30 (0.8)	0.13 (0.3)	0.08 (0.2)	0.08 (0.2)	714.2 (810.4)	11.0 (2.3)	20.6 (3.5)
		100%	3.6 (4.5)	4.3 (4.5)	0.53 (1.0)	0.11 (0.2)	0.12 (0.3)	0.07 (0.2)	2313.7 (5060.9)	13.8 (3.2)	22.0 (5.0)
		120%	5.4 (4.4)	4.8 (4.4)	0.08 (0.2)	0.07 (0.2)	0.05 (0.1)	0.01 (0.0)	4754.9 (13591.9)	30.4 (19.2)	28.2 (7.9)
		150%	7.2 (4.5)	7.5 (3.8)	0.08 (0.2)	0.05 (0.2)	0.08 (0.2)	0.00 (0.0)	385.0 (418.6)	40.8 (29.0)	37.0 (14.5)
		200%	8.1 (4.0)	7.9 (4.2)	0.02 (0.0)	0.08 (0.2)	0.02 (0.0)	0.08 (0.2)	307.0 (401.2)	75.1 (92.2)	69.7 (79.5)
5	1	80%	3.0 (3.8)	7.9 (3.1)	0.18 (0.5)	0.00 (0.0)	0.04 (0.1)	0.00 (0.0)	339.3 (384.0)	9.7 (2.9)	16.3 (4.0)
		100%	4.5 (3.6)	7.1 (3.4)	0.10 (0.5)	0.06 (0.5)	0.00 (0.0)	0.00 (0.0)	332.7 (589.4)	10.8 (3.6)	18.2 (4.4)
		120%	5.5 (3.7)	8.4 (1.8)	0.03 (0.1)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	195.8 (320.0)	18.2 (12.9)	18.4 (5.2)
		150%	4.8 (4.6)	5.9 (4.4)	0.21 (0.4)	0.51 (0.8)	0.14 (0.4)	0.28 (0.7)	92.2 (143.7)	21.4 (16.4)	20.5 (5.5)
		200%	6.1 (4.1)	8.9 (2.5)	0.11 (0.3)	0.00 (0.0)	0.07 (0.2)	0.00 (0.0)	47.2 (32.3)	27.6 (21.7)	21.0 (4.4)
	2	80%	2.9 (2.8)	4.0 (3.7)	0.09 (0.3)	0.07 (0.3)	0.00 (0.0)	0.00 (0.0)	5211.2(5512.2)	15.9 (6.4)	22.7 (4.6)
		100%	3.8 (3.4)	4.7 (3.6)	0.17 (0.4)	0.09 (0.2)	0.07 (0.2)	0.07 (0.2)	7178.1(13283.0)	20.9 (9.6)	32.4 (18.6)
		120%	3.0 (4.8)	4.4 (4.5)	1.07 (1.4)	0.49 (0.8)	1.01 (1.5)	0.13 (0.3)	4162.4(9710.7)	37.3 (36.5)	37.7 (25.6)
		150%	5.4 (4.4)	5.2 (4.8)	0.11 (0.2)	0.27 (0.5)	0.04 (0.1)	0.20 (0.5)	3615.5(10319.5)	52.1 (62.1)	54.0 (62.6)
		200%	6.5 (4.6)	6.6 (3.8)	0.06 (0.1)	0.10 (0.2)	0.04 (0.1)	0.07 (0.2)	347.6(133.7)	97.8 (99.2)	96.4 (103.0)
Avg.			**4.97**	**6.34**	**0.17**	**0.11**	**0.09**	**0.06**	**1996.62**	**27.06**	**30.35**

We performed statistical tests to check if there were significant differences between the results obtained with both heuristic algorithms for the instances with $i_{max} = 100$. Since the data did not fit a normal distribution, we used the Mann-Whitney-Wilcoxon test and found, with a significance level equal to 0.05, that the heuristic and the hybrid method were statistically different in terms of their average objective values. However, we found no differences between the best solutions found by both methods. Regarding the computational time, we found significant differences between both algorithms. The above results indicate that, if both algorithms are run enough times, in some of those runs the two methods find the same solution (in many cases, the optimal one). However, the hybrid method is more robust than the heuristic because its results, taking the average objective value of all runs, are significantly better, and this using a computational time very similar to that of the heuristic.

Table 8 shows the results for larger instances ($i_{max} = 500$ and 1000). As before, all problems were run 10 times and the computational times of the hybrid method include the time spent by the B&B algorithm to generate the list of boxes (15, 50 and 300 seconds for k=1, 3 and 5, respectively). Although the average of the objective function value seems to suggest that the heuristic algorithm performs slightly better that the hybrid algorithm, there are no statistical differences according to the statistical tests performed. The difference in the computational time is not significant either.

7. Conclusions and future research

When a chain wants to expand in a given geographical area, it can choose among opening a new facility, varying the quality of the existing chain-owned facilities, closing some of these facilities or a combination of all these possibilities. In this paper, the continuous competitive facility location and design model introduced in Fernández et al. (2007) has been extended to accommodate all these possibilities. The resulting model is a hard-to-solve MINLP problem, for which existing solvers fail.

Three different approaches are proposed in this paper. First, an exact spatial interval branch-and-bound method is proposed. It is a modification of the classical interval B&B methods proposed for continuous NLP problems to handle integer variables. Second, an ad-hoc heuristic has been designed. It uses a *profitability index* to select the facilities to be upgraded, downgraded or closed, and a Weiszfeld-like algorithm to update the location of the new facility and the qualities of the chain-owned facilities (new and existing ones). Third, a hybrid heuristic is also proposed, which makes use of boxes provided by the interval B&B (using a non-reliable discarding test based on the *pup* index) obtained at early stages of its iterative process, and applies to them the ad-hoc heuristic.

The exact interval B&B algorithm can only solve medium size problems. As for the heuristic and hybrid algorithms, they both are quite effective and find solutions of objective value very close to the optimum. Furthermore, the computational time required by both algorithms is much smaller than the interval B&B. Note, however, that they cannot guarantee that the optimal solution has been found, and in the best case they just find *one* optimal solution, whereas the interval B&B method finds *all* the optimal solutions with guarantee. It is worth noting that the hybridizing idea used in this work could also be applied to other heuristics.

As future research, we plan to adapt more discarding tests (Fernández & Pelegrín, 2001; Tóth & Fernández, 2010) and to design new ones to make the exact interval B&B algorithm faster. As for the hybrid method, new strategies to compute suitable parameters of the *pup* index and to determine the suitable time that the B&B process should be employed will be investigated. We point out that the algorithms introduced in this paper could be parallelized to make use of high-performance computing devices (Redondo, Fernández, García, & Ortigosa, 2008; 2011). A reformulation of the model following the lines of Aros-Vera, Marianov, & Mitchell (2013) to get a near convex MINLP model with bilinear terms should also be investigated. The model could be extended in several ways, such as to take into account the possibility of locating more than one new facility (Redondo, Fernández, García, & Ortigosa, 2011), or to take the possible reaction of the competitors

Table 8
Results for the problems with 500 and 1000 demand points using the heuristic and the hybrid heuristic algorithms.

i_{max}	j_{max}	k	Budget	Average solution		Computational time (sec.)	
				heur.	hybr.	heur.	hybr.
500	7	1	80%	1601.6	1494.6	34.9 (7.4)	37.1 (15.2)
			100%	1935.9	1936.5	43.7 (16.3)	62.0 (34.8)
			120%	2178.1	2172.8	43.8 (15.1)	48.7 (17.2)
			150%	2475.1	2474.8	47.4 (15.3)	52.5 (17.4)
			200%	2835.9	2827.0	51.6 (15.4)	54.4 (18.3)
		3	80%	4728.7	4751.9	83.7 (15.8)	119.1 (31.4)
			100%	5196.3	5214.4	88.0 (25.2)	127.9 (21.6)
			120%	5519.2	5452.0	150.7 (133.3)	196.3 (166.7)
			150%	5765.8	5740.6	250.7 (328.9)	318.4 (403.4)
			200%	5882.4	5886.7	777.0 (1010.6)	864.1 (1144.6)
		5	80%	8477.4	8489.3	128.5 (81.5)	367.9 (40.4)
			100%	8740.3	8744.8	392.8 (377.1)	400.7 (30.0)
			120%	8890.6	8889.0	2408.6 (1352.2)	1327.5 (852.5)
			150%	8914.4	8910.8	2019.0 (1272.2)	1519.4 (972.0)
			200%	8922.6	8911.2	2531.5 (1216.1)	2067.7 (1044.3)
	10	1	80%	1199.1	1198.6	42.5 (17.3)	43.5 (18.8)
			100%	1412.3	1411.5	44.0 (17.0)	56.7 (29.2)
			120%	1570.0	1570.0	45.5 (18.8)	56.1 (21.6)
			150%	1769.9	1769.4	52.5 (21.0)	60.5 (21.9)
			200%	2022.5	2020.1	61.3 (19.3)	68.0 (17.0)
		3	80%	2819.2	2843.9	99.8 (29.1)	140.5 (22.2)
			100%	3168.1	3163.5	107.8 (34.0)	147.6 (47.3)
			120%	3406.0	3394.2	137.8 (64.2)	174.3 (54.9)
			150%	3599.5	3599.5	267.5 (195.0)	274.3 (149.7)
			200%	3649.7	3651.2	934.8 (739.4)	1033.2 (836.6)
		5	80%	5029.7	5014.3	145.9 (40.7)	414.5 (49.7)
			100%	5316.2	5307.9	451.7 (309.2)	434.3 (95.1)
			120%	5480.5	5482.4	1723.5 (1143.1)	1060.4 (745.5)
			150%	5502.3	5506.4	1926.3 (1037.9)	1551.9 (920.3)
			200%	5502.3	5506.2	2121.9 (1262.0)	2373.1 (1515.0)
1000	7	1	80%	1958.3	1958.0	78.0 (35.9)	51.8 (41.8)
			100%	2573.9	2574.0	87.3 (43.1)	74.6 (55.0)
			120%	2863.1	2863.0	122.4 (44.0)	84.4 (67.9)
			150%	3326.3	3328.4	121.8 (34.8)	93.9 (57.9)
			200%	3994.3	3991.1	128.4 (34.3)	104.3 (52.3)
		3	80%	7172.1	7216.1	198.8 (32.7)	218.7 (54.6)
			100%	7983.2	7911.6	185.6 (34.9)	226.3 (45.0)
			120%	8435.7	8419.0	205.8 (52.6)	236.2 (43.3)
			150%	8953.1	8950.4	198.8 (50.0)	264.3 (65.4)
			200%	9238.7	9238.7	475.9 (435.9)	602.1 (534.8)
		5	80%	12864.2	12794.7	208.1 (49.3)	425.8 (50.2)
			100%	13319.6	13298.7	453.0 (439.9)	547.3 (185.9)
			120%	13503.9	13518.1	4664.2 (4451.3)	2717.4 (2588.0)
			150%	13569.5	13565.6	4430.7 (3358.3)	4755.5 (3530.7)
			200%	13570.3	13567.0	4782.4 (3003.5)	5832.4 (3831.0)
	10	1	80%	1175.4	1175.7	88.2 (26.9)	48.6 (46.7)
			100%	1554.1	1554.2	95.3 (25.9)	111.2 (102.4)
			120%	1705.2	1705.3	88.9 (32.4)	79.4 (66.0)
			150%	1984.6	1979.8	101.8 (37.2)	75.6 (56.7)
			200%	2383.5	2383.6	103.4 (38.5)	121.5 (58.8)
		3	80%	4592.6	4642.6	172.1 (35.0)	235.0 (63.8)
			100%	5194.0	5163.1	185.6 (45.5)	240.7 (70.8)
			120%	5559.1	5588.7	208.8 (45.1)	245.9 (45.3)
			150%	6031.5	6033.6	212.8 (53.3)	252.9 (65.4)
			200%	6292.1	6290.1	504.0 (719.6)	531.3 (685.5)
		5	80%	7594.4	7553.0	258.2 (55.5)	523.2 (78.8)
			100%	8229.7	8224.1	420.4 (229.4)	585.0 (110.8)
			120%	8526.6	8511.0	1844.9 (2316.9)	1935.0 (2330.2)
			150%	8665.5	8663.9	3260.4 (3001.6)	3410.6 (2947.4)
			200%	8672.1	8672.1	2855.0 (2147.4)	3497.5 (2715.1)
Avg.				**5583.30**	**5577.85**	**732.59**	**726.35**

into account, leading to a Stackelberg (or leader-follower) problem (Redondo, Fernández, Arrondo, García, & Ortigosa, 2013). Similar location models, using other customer choice rules (Fernández, Tóth, Redondo, Ortigosa, & Arrondo, 2017; Fernández et al., 2019) should also be investigated.

Acknowledgments

We would like to thank Juana L. Redondo and Pilar M. Ortigosa for their collaboration in an early version of the paper. We also want to thank the referees, whose comments have helped to improve the paper.

Appendix A. Data of the quasi-real example in Section 2.1

The data of the quasi-real example in Section 2.1 are basically the same as in the aggregated problem in Fernández & Tóth (2009). However, in order to fit the new model in the present paper, we have modified a few parameters, namely, we have set $c = 80, \beta_0 = 8,$ and $\beta_1 = 2.5$. Besides of that, the following choices for the new parameters were made:

- $\alpha^j_{\max} = 5$
- $A_1 = 8, A_2 = 10$
- $C_j = A_j/2$
- $B = 100$
- $\delta_j = 4, j = 1, 2$
- $v_j = \exp(1/\beta_0 + \beta_1) - \exp(\beta_1) = 1.622$
- $o_j = 20A_j$

Appendix B. Detailed comparison of the discarding tests in the interval branch-and-bound method

In order to analyse the efficiency of each of the discarding tests, we compare the Basic, Mono, Newton and NonConc versions of the interval B&B algorithm, as defined in Section 6.4.

In Table B.1 we compare these versions for the different problem settings with $i_{\max} = 100$ demand points. As the main indicator of efficiency, the table gives the average computational time in seconds for the 10 problems in each setting. For those settings, where the given method could not finish before its time limit, the 12 hours were taken into account (there were 39 such cases for Basic, 4 for Mono, and only 2 for Newton and Nonconc out of the 200 scenarios).

In the first three columns of Table B.1, the type of the problems is shown: first the number of existing facilities and the number of facilities belonging to the expanding chain is given, and then, the budget in relative terms to $\tilde{B}$, the budget needed to run the facilities of the expanding chain with their current setting. In the following columns, for each version of the interval method (Basic, Mono, Newton and NonConc) the average computational time in seconds is reported. Moreover, we report in percentages the ratio between each version with its preceding variant, in the columns named $\frac{\text{Mono}}{\text{Basic}}$, $\frac{\text{Newt}}{\text{Mono}}$ and $\frac{\text{NonConc}}{\text{Newt}}$. Additionally, for each problem setting, we report the averages for the different budget scenarios taken by the versions of the interval B&B method.

As we can see, for the Basic version of the interval method, the smallest budget always leads to easier-to-solve problems, as the feasibility test can efficiently discard large regions from the infeasible region. Increasing the budget usually increases the time, although for some (m, k) settings we can see a decrease of time after the 120% budget.

The use of the monotonicity test with the centered form always makes the interval B&B method faster. For some settings, the improvement of Mono over Basic is dramatic, being more than 10 times faster. When only one chain-owned facility exists ($k = 1$), the tendency is clear: the more budget we have, the more efficient the monotonicity test is, probably because there are more boxes which certainly satisfy the budget constraint, and to which the monotonicity test can be applied. In case two chain-owned facilities exist, there is no such tendency, but still, Mono is very efficient, two times faster than Basic.

Table B.1
Average computational time in seconds for the different versions of the interval B&B method.

$j_{\max}$	k	Budget	Basic	Mono	$\frac{\text{Mono}}{\text{Basic}}$	Newton	$\frac{\text{Newt}}{\text{Mono}}$	NonConc	$\frac{\text{NonConc}}{\text{Newt}}$
3	1	80%	8437	3307	39%	3293	100%	3296	100%
		100%	17732	4384	25%	4388	100%	4389	100%
		120%	17570	1928	11%	1928	100%	1922	100%
		150%	7210	177	2%	174	98%	176	101%
		200%	7117	187	3%	153	81%	159	104%
		Avg.	**11613**	**1997**	**17%**	**1987**	**100%**	**1989**	**100%**
	2	80%	7105	716	10%	714	100%	716	100%
		100%	14124	2331	17%	2314	99%	2314	100%
		120%	13804	4799	35%	4755	99%	4760	100%
		150%	14569	5925	41%	385	6%	424	110%
		200%	19022	5926	31%	307	5%	354	115%
		Avg.	**13725**	**3939**	**29%**	**1695**	**43%**	**1713**	**101%**
5	1	80%	3320	339	10%	339	100%	338	100%
		100%	8544	335	4%	333	99%	333	100%
		120%	11523	199	2%	196	98%	198	101%
		150%	10131	100	1%	92	92%	94	102%
		200%	13284	99	1%	47	48%	54	115%
		Avg.	**9360**	**214**	**2%**	**201**	**94%**	**203**	**101%**
	2	80%	11091	5230	47%	5211	100%	5230	100%
		100%	17834	9212	52%	7178	78%	7167	100%
		120%	12466	6687	54%	4162	62%	4181	100%
		150%	13877	6665	48%	3616	54%	3675	102%
		200%	15749	6758	43%	348	5%	388	112%
		Avg.	**14203**	**6911**	**49%**	**4103**	**59%**	**4128**	**101%**

The projected one-dimensional Newton method is also efficient in general, although not always. For the lowest budget, it cannot usually be applied, as most of the boxes are undetermined for the budget constraint; that is why for these problems it does not reduce the CPU time. The higher the budget, the more efficient the Newton method is. And it is particularly efficient when two chain-owned facilities exist (when the monotonicity test is less efficient): in these cases, when 150% and 200% of the budget $\tilde{B}$ is available, the Newton version is 20 times faster than Mono and it employs just 5% of the running time of the Mono version. Notice that in these settings there are three quality variables for which the projected one-dimensional Newton method can be applied.

On the other hand, the projected one-dimensional non-concavity test is not efficient for our location problem (only for three out of the 20 settings we can see a small improvement). For smaller budgets, it cannot usually be applied, as most of the boxes are undetermined for the budget constraint. For higher budgets, its use provoked an increase in the CPU time, mainly because both the monotonicity and the projected one-dimensional Newton are very efficient, and there is not much room for improvement.

References

Aros-Vera, F., Marianov, V., & Mitchell, J. E. (2013). p-Hub approach for the optimal park-and-ride facility location problem. *European Journal of Operational Research, 226*(2), 277–285.

Ashtiani, M. (2016). Competitive location: A state-of-art review. *International Journal of Industrial Engineering Computations, 7*, 1–18.

Belotti, P., Berthold, T., Bonami, P., Cafieri, S., Margot, F., Megaw, C., Vigerske, S., & Wachter, A. (2020). COUENNE (Convex Over and Under ENvelopes for Nonlinear Estimation). (Accessed 17 March 2022) https://projects.coin-or.org/Couenne.

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., & Mahajan, A. (2013). Mixed-integer nonlinear optimization. *Acta Numerica, 22*, 1–131.

Belotti, P., Lee, J., Liberti, L., Margot, F., & Wachter, A. (2009). Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software, 24*(4-5), 597–634.

Benati, S., & Hansen, P. (2002). The maximum capture problem with random utilities: Problem formulation and algorithms. *European Journal of operational research, 143*(3), 518–530.

Benoist, T., Estellon, B., Gardi, F., Megel, R., & Nouioua, K. (2011). LocalSolver 1.x: A black-box local-search solver for 0–1 programming. *4OR, 9*(3), 299.

Benson, H. Y., & Vanderbei, R. J. (2003). Solving problems with semidefinite and related constraints using interior-point methods for nonlinear programming. *Mathematical Programming, 95*(2), 279–302. https://doi.org/10.1007/s10107-002-0350-x. https://vanderbei.princeton.edu/loqo/LOQO.html.

Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., … Wachter, A. (2005). An algorithmic framework for convex mixed integer nonlinear programs. *Technical Report*. IBM Research Report. (Accessed 17 March 2022) https://www.coin-or.org/Bonmin/.

Bonami, P., Klinç, M., & Linderoth, J. (2012). Algorithms and software for convex mixed integer nonlinear programs. In *Mixed integer nonlinear programming*. In *The IMA volumes in mathematics and applications: vol. 154* (pp. 1–39). Berlin: Springer Science+Business Media, The IMA volumes in mathematics and applications.

Burer, S., & Letchford, A. (2012). Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science, 17*(1), 97–106.

Byrd, R. H., Nocedal, J., & Waltz, R. (2006). *KNITRO: An integrated package for nonlinear optimization*. In G. di Pillo, & M. Roma (Eds.) (pp. 35–59)). Springer. (Accessed 17 March 2022) https://www.artelys.com/solvers/knitro/.

Cooper, L. G., & Nakanishi, M. (1989). *Market-share analysis: Evaluating competitive marketing effectiveness*: vol. 1. Springer Science & Business Media.

Correia, I., & Captivo, M. E. (2003). A Lagrangean heuristic for a modular capacitated location problem. *Annals of Operations Research, 122*(1), 141–161.

Drezner, T. (1995). Competitive facility location in the plane. In Z. Drezner (Ed.), *Facility location: A survey of applications and methods*. In *Springer series in operations research and financial engineering* (pp. 285–300). Berlin: Springer.

Drezner, T. (2014). A review of competitive facility location in the plane. *Logistics Research, 7*. Article ID: 114.

Drezner, T., Drezner, Z., & Kalczynski, P. (2012). Strategic competitive location: Improving existing and establishing new facilities. *Journal of the Operational Research Society, 63*(12), 1720–1730.

Eiselt, H., Marianov, V., & Drezner, T. (2015). Competitive location models. In G. Laporte, S. Nickel, & F. Saldanha-da Gama (Eds.), *Location science* (pp. 365–398). Springer. chapter 14.

Fernández, J., Fernández, P., & Pelegrín, B. (2002). Estimating actual distances by norm functions: A comparison between the $l_{k,p,\theta}$-norm and the $l_{b_1,b_2,\theta}$-norm and a study about the selection of the data set. *Computers & Operations Research, 29*(6), 609–623.

Fernández, J., & Hendrix, E. (2013). Recent insights in Huff-like competitive facility location and design. *European Journal of Operational Research, 227*(3), 581–584.

Fernández, J., & Pelegrín, B. (2001). Using interval analysis for solving planar single–facility location problems: New discarding tests. *Journal of Global Optimization, 19*(1), 61–81.

Fernández, J., Pelegrín, B., Plastria, F., & Tóth, B. (2007). Solving a Huff-like competitive location and design model for profit maximization in the plane. *European Journal of Operational Research, 179*(3), 1274–1287.

Fernández, J., & Tóth, B. (2009). Obtaining the efficient set of nonlinear biobjective optimization problems via interval branch-and-bound methods. *Computational Optimization and Applications, 42*(3), 393–419.

Fernández, J., Tóth, B. G., Redondo, J., Ortigosa, P., & Arrondo, A. (2017). A planar single-facility competitive location and design problem under the multi-deterministic choice rule. *Computers & Operations Research, 78*, 305–315.

Fernández, J., G.-Tóth, B., Redondo, J. L., & Ortigosa, P. M. (2019). The probabilistic customer's choice rule with a threshold attraction value: Effect on the location of competitive facilities in the plane. *Computers & Operations Research, 101*, 234–249.

Fourer, R., Gay, D., & Kernighan, B. (2003). *AMPL. A modeling language for mathematical programming (second edition)*. Duxbury-Thomson. (Accessed 17 March 2022) https://ampl.com/.

Freire, A. S., Moreno, E., & Yushimito, W. F. (2016). A branch-and-bound algorithm for the maximum capture problem with random utilities. *European Journal of Operational Research, 252*(1), 204–212.

Gill, P. E., Murray, W., & Saunders, M. A. (2005). SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review, 47*(1), 99–131. https://doi.org/10.1137/S0036144504446096. https://ccom.ucsd.edu/~optimizers/solvers/snopt/.

Haase, K., & Müller, S. (2014). A comparison of linear reformulations for multinomial logit choice probabilities in facility location models. *European Journal of Operational Research, 232*(1), 689–691.

Hakimi, S. (1990). Locations with spatial interactions: Competitive location and games. In R. Francis, & P. Mirchandani (Eds.), *Discrete location theory* (pp. 439–478). New York: Wiley/Interscience.

Hammer, R., Hocks, M., Kulisch, U., & Ratz, D. (1995). *C++ toolbox for verified computing I: Basic numerical problems: Theory, algorithms and programs*. Berlin: Springer-Verlag.

Hansen, E., & Walster, G. W. (2004). *Global optimization using interval analysis - Second edition, revised and expanded*. New York: Marcel Dekker.

Huff, D. (1964). Defining and estimating a trading area. *Journal of Marketing, 28*(3), 34–38.

Jain, A., & Mahajan, V. (1979). Evaluating the competitive environment in retailing using multiplicative competitive interactive models. *Research in Marketing, 1*, 217–235.

Kearfott, R. (1996). *Rigorous global search: Continuous problems*. Dordrecht: Kluwer.

Kearfott, R., Nakao, M., Neumaier, A., Rump, S. M., Shary, S. P., & van Hentenryck, P. (2010). Standardized notation in interval analysis. *TOM, 15*(1), 7–13.

Knüppel, O. (1993). PROFIL/BIAS - A fast interval library. *Computing, 53*(1), 277–287.

Küçükaydin, H., Aras, N., & Altinel, I. K. (2012). A leader-follower game in competitive facility location. *Computers & Operations Research, 39*(2), 437–448.

Ljubić, I., & Moreno, E. (2018). Outer approximation and submodular cuts for maximum capture facility location problems with random utilities. *European Journal of Operational Research, 266*(1), 46–56.

Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., & Munson, T. (2020). Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation, 13*(2), 301–338. https://doi.org/10.1007/s12532-020-00196-1.

Mai, T., & Lodi, A. (2020). A multicut outer-approximation approach for competitive facility location under random utilities. *European Journal of Operational Research, 284*(3), 874–881.

Markót, M., Fernández, J., Casado, L., & Csendes, T. (2006). New interval methods for constrained global optimization. *Mathematical Programming Series A, 106*, 287–318.

Misener, R., & Floudas, C. (2014). ANTIGONE: Algorithms fon coNTinuous/integer global optimization of nonlinear equations. *Journal of Global Optimization, 59*(2), 503–526.

Nakanishi, M., & Cooper, L. (1974). Parameter estimate for multiplicative interactive choice model: Least square approach. *Journal of Marketing Research, 11*, 303–311.

Nakanishi, M., & Cooper, L. (1982). Simplified estimation procedures for MCI models. *Marketing Science, 1*(3), 314–322.

Pelegrín, B., Fernández, J., & Tóth, B. (2008). The 1-center problem in the plane with independent random weights. *Computers & Operations Research, 35*(3), 737–749.

Pintér, J. D. (1997). LGO-A program system for continuous and Lipschitz global optimization. In *Developments in global optimization* (pp. 183–197). Springer.

Plastria, F. (2001). Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research, 129*(3), 461–470.

Rall, L. (1981). Automatic differentiation, techniques and applications. *Lecture notes in computer science*. Berlin: Springer.

Ratschek, H., & Rokne, J. (1988). *New computer methods for global optimization*. Chichester: Ellis Horwood.

Redondo, J., Fernández, J., Álvarez, J., Arrondo, A., & Ortigosa, P. (2015). Approximating the Pareto-front of a planar bi-objective competitive facility location and design problem. *Computers & Operations Research, 62*, 337–349.

Redondo, J., Fernández, J., Arrondo, A., García, I., & Ortigosa, P. (2013). A two-level evolutionary algorithm for solving the facility location and design (1|1)-centroid problem on the plane with variable demand. *Journal of Global Optimization, 56*(3), 983–1005.

Redondo, J., Fernández, J., García, I., & Ortigosa, P. (2008). Parallel algorithms for continuous competitive location problems. *Optimization Methods & Software, 23*(5), 779–791.

Redondo, J., Fernández, J., García, I., & Ortigosa, P. (2009a). A robust and efficient global optimization algorithm for planar competitive location problems. *Annals of Operations Research, 167*(1), 87–106.

Redondo, J., Fernández, J., García, I., & Ortigosa, P. (2009b). Solving the multiple competitive facilities location and design problem on the plane. *Evolutionary Computation, 17*(1), 21–53.

Redondo, J., Fernández, J., García, I., & Ortigosa, P. (2011). Parallel algorithms for continuous multifacility competitive location problems. *Journal of Global Optimization, 50*(4), 557–573.

Sahinidis, N. V. (2017). BARON 17.8.9: Global optimization of mixed-integer nonlinear programs, user's manual. (Accessed 17 March 2022) http://www.minlp.com/downloads/docs/baron%20manual.pdf.

Saidani, N., Chu, F., & Chen, H. (2012). Competitive facility location and design with reactions of competitors already in the market. *European Journal of Operational Research, 219*(1), 9–17.

Storn, R., & Price, K. V. (1997). Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

Tóth, B., & Fernández, J. (2010). *Interval methods for single and bi-objective optimization problems - applied to competitive facility location problems*. Saarbrücken: Lambert Academic Publishing.

Tóth, B., Fernández, J., & Csendes, T. (2007). Empirical convergence speed of inclusion functions for facility location problems. *Journal of Computational and Applied Mathematics, 199*, 384–389.

Tóth, B., Fernández, J., Pelegrín, B., & Plastria, F. (2009a). Sequential versus simultaneous approach in the location and design of two new facilities using planar Huff-like models. *Computers & Operations Research, 36*(5), 1393–1405.

Tóth, B., Plastria, F., Fernández, J., & Pelegrín, B. (2009b). On the impact of spatial pattern, aggregation, and model parameters in planar Huff-like competitive location and design problems. *OR Spectrum, 31*(1), 601–627.

Trespalacios, F., & Grossmann, I. (2014). Review of mixed-integer nonlinear and generalized disjunctive programming methods. *Chemie Ingenieur Technik, 86*(7), 991–1012.

Vigerske, S., & Gleixner, A. (2018). SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software, 33*(3), 563–593. (Accessed 17 March 2022) https://scip.zib.de/.

Wächter, A., & Biegler, L. T. (2005). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming, 106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y. https://coin-or.github.io/Ipopt/.

Westerlund, T., & Pörn, R. (2002). Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering, 3*(3), 253–280. (Accessed 17 March 2022) https://www.gams.com/latest/docs/S_ALPHAECP.html.