A combined model for chain expansion including the possibility of locating a new facility and modification 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*

Abstract

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

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

*This research has been supported by grants from the Spanish Ministry of Economy and Competitiveness (MTM2015-70260-P and TIN2015-66680-C2-1-R), 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 of the Spanish Ministry of Science and Innovation and grant PID2019-105952GB-I00 funded by Ministerio de Ciencia e Innovación/ Agencia Estatal de Investigación /10.13039/501100011033.

*Corresponding author, e-mail: boglarka@inf.szte.hu

Email addresses: boglarka@inf.szte.hu (Boglárka G.-Tóth), l.anton@umh.es (Laura Anton-Sanchez), josefdez@um.es (José Fernández)

1. Introduction

Where to set up a facility so as to optimize a given objective is the topic Location Science is devoted to. When the facilities to be located can be set up anywhere in a given area of the plane, we have *continuous* location models, which are commonly modelled as NLP problems. If the potential set of locations is a discrete set of points, then we have a *discrete* location model, usually leading to MILP problems. If the new facilities can be placed anywhere in a network (at nodes or in the edges) then we are faced to *network* location models, which are usually transformed into MILP problems. The variety of models and techniques is so huge that there are two entries in the AMS 2010 Mathematics Subject Classification: 90B80 (Discrete location and assignment) and 90B85 (Continuous location). See Drezner and Hamacher (2002); Eiselt and Marianov (2015); Laporte et al. (2015); Mirchandani and Francis (1990) for an introduction to the topic.

In this research a continuous *competitive* facility location problem is investigated, which 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, 2014); Eiselt et al. (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 (so as to invest the budget allocated to them to the other ones). However, to our knowledge, in none of the models in the literature the existing facilities of the locating chain are allowed to vary their quality or to be closed. And this is exactly what it is researched in this paper. The most related papers with this topic are the following ones. In Saidani et al. (2012) a continuous competitive single facility location model with foresight is analysed, in which the facilities of the competitors may adjust their quality up or down; but the locating chain is new in the area (it does not own any existing facility) and an unlimited budget is assumed. A related leader-follower *discrete* model is considered in Küçükaydin et al. (2012). The leader is to open an unknown number of facilities, and the follower reacts by adjusting the quality of its existing facilities and/or closing some of them and/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 et al. (2012) a different approach is followed; it is assumed that customers patronize a facility only if they are within its 'radius of influence' (which can be interpreted as a surrogate of the quality) and if they are in the sphere of influence of more than one facility then 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) and/or upgrade (downgrading is not allowed) the quality (increase the radius of influence) of its existing ones. See Fernández

and Hendrix (2013) for a deeper review of those papers.

As it will be shown in Section 2 the continuous location model introduced in this paper is a non-convex mixed-integer nonlinear programming (MINLP) problem. Many researches have been working on the topic of designing and implementing practical algorithms for coping with MINLP problems during the last decades (see Belotti et al. (2013); Bonami et al. (2012); Burer and Letchford (2012); Trespalacios and Grossmann (2014) and the references therein). And some software packages are available, for both convex (AlphaECP Westerlund and Pörn (2002), BONMIN Bonami et al. (2005), Knitro Byrd et al. (2006) or MINOTAUR Leyffer et al.) and non-convex MINLP problems (ANTIGONE Misener and Floudas (2014), BARON Sahinidis (2017), COUENNE Belotti et al. or SCIP Vigerske and 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 the possibilities of 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 (an extension of the model in Fernández et al. (2007)) is introduced in Section 2; the *expanding* chain has a *limited budget*, and it may locate a new facility anywhere in *a given region of the plane* and/or *vary (up or down) the quality of its existing facilities* and/or *close some of them*; the customer choice rule is assumed to be *probabilistic*. (2) A branch-and-bound method based in interval analysis is proposed in Section 3; it is suitable for small-size instances and can also be applied to many other MINLP problems. (3) An ad-hoc heuristic is also proposed in Section 4 to tackle the problem, which has a good performance. And (4) in Section 5 a hybrid heuristic, which makes use of the ability of the interval B&B method to discard big areas of the searching region at early stages of the iterative process, is also 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. The location and design model for firm expansion

2.1. The continuous pure-location model

Since the new model introduced in this paper is an extension of the model in Fernández et al. (2007), we first briefly describe it and introduce the required notation. The model in Fernández et al. (2007) is a pure-location model, in the sense that the expanding chain only considers the opening of one new facility in the area. Some existing facilities may be owned by the expanding chain. The location and quality of all the existing facilities is known as well as the location and demand of customers (or demand points). The utility provided by a facility to a customer is measured as quality divided by (a function of the) distance.

And customers are assumed to follow the probabilistic choice rule. The notation used throughout the paper will be the following one:

Index sets

- i subscript for demand points, $i = 1, \ldots, i_{\max}$.
- j subscript for the existing facilities, $j = 1, \ldots, j_{\max}$ (the first k of them ($k < j_{\max}$) 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.

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.
- $\tilde{\alpha}_j$ quality of f_j.
- $g_i(\cdot)$ a non-decreasing (and non-negative) function.
- $\tilde{u}_{ij}$ utility that p_i perceives from f_j, given by $\tilde{u}_{ij} = \tilde{\alpha}_j / g_i(d_{ij})$.
- $d_i^{\min}$ radius of the circular area around p_i where the location is not allowed.
- $\alpha_{\min}$ minimum quality level for the new facility.
- $\alpha_{\max}$ maximum quality level for the new facility.
- S area where the new facility can be set up.
- G^{ex} annual fixed costs due to the existing chain-owned facilities.

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, $u_{i0}(f_0, \alpha_0) = \alpha_0 / g_i(d_i(f_0))$.

The annual market share captured by the expanding chain after the location of the new facility is given by

$$M_P(f_0, \alpha_0) = \sum_{i=1}^{i_{\max}} w_i \frac{u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{k} \tilde{u}_{ij}}{u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{j_{\max}} \tilde{u}_{ij}},$$

and the problem we are faced to is

$$\begin{cases} \max & \Pi_P(f_0, \alpha_0) = F(M_P(f_0, \alpha_0)) - G(f_0, \alpha_0) - G^{ex} \\ \text{s.t.} & f_0 \in S \subset \mathbb{R}^2 \\ & d_i(f_0) \geq d_i^{\min}, i = 1, \ldots, i_{\max} \\ & \alpha_0 \in [\alpha_{\min}, \alpha_{\max}] \end{cases}$$

The differentiable function $F(\cdot)$ converts the market share into sales, whereas the differentiable function $G(f_0, \alpha_0)$ accounts for the annualized cost of opening and running a facility located at f_0 with quality α_0. Hence, $\Pi_P(f_0, \alpha_0)$ measures the annual profit of the expanding chain.

The functional form of F and G should be tailored for each real problem. Possible expressions for both can be found in Fernández et al. (2007). In this paper we will use $F(M_P(f_0, \alpha_0)) = c \cdot M_p(f_0, \alpha_0)$, where c is the income 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.

2.2. Extended model

In the previous model, the expanding chain only considers one possibility to increase its profit: opening one new facility. However, in practice, the chain could have 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 following extended model takes into account all those possibilities. Some additional notation is required for its formulation. Note that in this model it is assumed that $k > 0$, i.e., the expanding chain already has some facilities operating in the area.

Variables

α_j	quality of the j-th existing facility owned by the expanding chain, $j = 1, \ldots, k$. At present, $\alpha_j = \tilde{\alpha}_j$.
y_j	binary variable which is 0 if facility j is closed ($j = 1, \ldots, k$) or not open ($j = 0$); and 1 otherwise.
ns	variables of the problem, $ns = (f_0, \alpha_0, \alpha_1, \ldots, \alpha_k, y_0, \ldots, y_k)$.

Parameters

$\tilde{\alpha}_j$	present quality of existing facility f_j, $j = 1, \ldots, j_{\max}$.
$\tilde{u}_{ij}$	utility that p_i perceives from f_j at present, $i = 1, \ldots, i_{\max}$, $j = 1, \ldots, j_{\max}$. In this paper, $\tilde{u}_{ij} = \tilde{\alpha}_j / g_i(d_{ij})$.
$\alpha^j_{\max}$	maximum quality level that f_j may have, $j = 1, \ldots, k$.
A_j	annualized cost corresponding to the opening of the j-th facility, $j = 1, \ldots, k$.
C_j	cost of closing facility f_j, $j = 1, \ldots, k$.
B	annual chain's budget (for opening a new facility or varying the quality of existing facilities or closing facilities as well as for the operational costs of the open facilities).

Computed values

$u_{ij}(\alpha_j)$	utility that p_i perceives from f_j, $i = 1, \ldots, i_{\max}, j = 1, \ldots, k$. In this paper, $u_{ij}(\alpha_j) = \alpha_j / g_i(d_{ij})$.
$V_j(\alpha_j)$	annualized cost of varying the quality of f_j to $\alpha_j, j = 1, \ldots, k$.
$M(ns)$	annual market share captured by the chain.
$F(M(ns))$	annual sales of the chain.
$R_j(\alpha_j)$	annual operating cost of the j-th facility, $j = 0, \ldots, k$.
$T(ns)$	annual total cost of the chain.
$\Pi(ns)$	annual profit obtained by the chain.

Before solving the problem, facility f_j, $j = 1, \ldots, k$, 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 $\alpha^j_{\max} \leq \alpha_{\max}$ from its present value $\tilde{\alpha}_j$. Hence, we will assume that $\alpha_j \in [\alpha_{\min}, \alpha^j_{\max}]$ if the facility remains open.

In addition to the annualized cost $G(f_0, \alpha_0)$ of opening the new facility (in case it is open) at a given location f_0 with a quality α_0 (note that now G does not include the annual operating cost), some additional costs must be taken now into account, namely, $A_j, C_j, V_j(\alpha_j)$, $j = 1, \ldots, k$, and $R_j(\alpha_j)$, $j = 0, \ldots, k$, as described above.

With the previous notation, the annual cost of the expanding chain is given by

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

$$+ y_0(G(f_0, \alpha_0) + R_0(\alpha_0)), \tag{2}$$

and the market share it captures is

$$M(ns) = \sum_{i=1}^{i_{\max}} w_i \frac{y_0 u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{k} y_j u_{ij}(\alpha_j)}{y_0 u_{i0}(f_0, \alpha_0) + \sum_{j=1}^{k} y_j u_{ij}(\alpha_j) + \sum_{j=k+1}^{j_{\max}} \tilde{u}_{ij}}. \tag{3}$$

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

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

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

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

$$\alpha_0 \geq y_0 \alpha_{\min} \tag{7}$$

$$\alpha_0 \leq y_0 \alpha_{\max} \tag{8}$$

$$\alpha_j \geq y_j \alpha_{\min}, \qquad j = 1, \ldots, k \tag{9}$$

$$\alpha_j \leq y_j \alpha^j_{\max}, \qquad j = 1, \ldots, k \tag{10}$$

$$T(ns) \leq B \tag{11}$$

$$y_j \in \{0, 1\}, \qquad j = 0, \ldots, k \tag{12}$$

Constraints (7)-(8) guarantee that if the new facility is opened ($y_0 = 1$) then $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$, and if it is not opened ($y_0 = 0$) then $\alpha_0 = 0$. Something similar is done in constraints (9)-(10) for the quality of the existing facilities.

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 $(\tilde{\alpha}_j, \alpha^j_{\max}]$, 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 a $\gamma > 0$ 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 $(\tilde{\alpha}_j, \alpha_{\max}]$ (also 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 (usually a variation in the quality requires a temporary closure of the facility). It is this fixed cost what 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:

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

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 non-decreasing, although its functional form may vary depending on the particular type of facility. 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 again that the particular choice of functions V_j and R_j does not affect the study of the aim of the paper.

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.

2.3. An example

The model has been applied to the case of the location of a hypermarket in an area around the city of Murcia, in south-eastern Spain Tóth et al. (2009). The aim of this subsection is to show how the model works.

There are $i_{\max} = 21$ population centers (demand points) in the area. Figure 1 shows their location as a gray 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 those 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 dot). In both cases, the original quality 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 on the location and on the quality pane as well. The new

quality of the existing facilities are also shown on the quality pane by a box; a green arrow points to the new quality. When a facility is closed an orange box is drawn, and 'Closed' is written under the box. When there are several global (or near-optimal) solutions different only in the quality values, like in the figure with a budget $B = 52$, we can see that the boxes for the inclusion of the qualities become larger. In such cases, a low quality on one facility is paired with a high quality for the other facility, but for the figure we show the inclusion of all the solutions.

In the first figure, to fulfill the low budget $B = 40$, only the facility in Murcia (the most populated city) is kept open (with a high quality), the other shop has to be closed, and no new facility can be opened. The same happens for budget $B = 41$, but then for $B = 42$ 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 $B = 47$, where the new facility is opened with minimal quality, directly competing against one the competitor's facilities; and of the two existing facilities, only the one in Murcia is kept opened, with its original quality; the other one is closed. Interestingly, this is the only budget where it is worth opening a new facility below the budget of 52. For budgets 48 and 49 the solution is to keep opened the two existing facilities with the original qualities (the new facility is not opened), while for the budgets 50 and 51, we can also increase the quality of our low quality facility. From the budget $B = 52$, it is again worth to open a new facility, and now all existing facilities remain open; however, the quality of the existing facilities are decreased: the quality of the facility in Murcia to 3.7-3.8 and the quality of the other one to its original low quality. 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 (being the quality of the new facility 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.

2.4. Difficulty at solving the model

We implemented the model in AMPL Fourer et al. (2003) and tried first to solve two problems for different budgets, with all the solvers suitable for MINLP problems callable from AMPL, namely, BARON Sahinidis (2017), BONMIN Bonami et al. (2005), COUENNE Belotti et al., SCIP Vigerske and Gleixner (2018), Knitro Byrd et al. (2006), LGO Pintér (1997) and LocSol Benoist et al. (2011).

The first 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 more than one of the ten instances of our first problem. But changing the conditional definition of V_j using additional binary variables BARON, SCIP and LocSol were able to solve the problems as well. For LocSol, we had to set a reasonable time limit, because it is used as stopping criterion. Setting the time limit to 1 second for this toy problem (both BARON

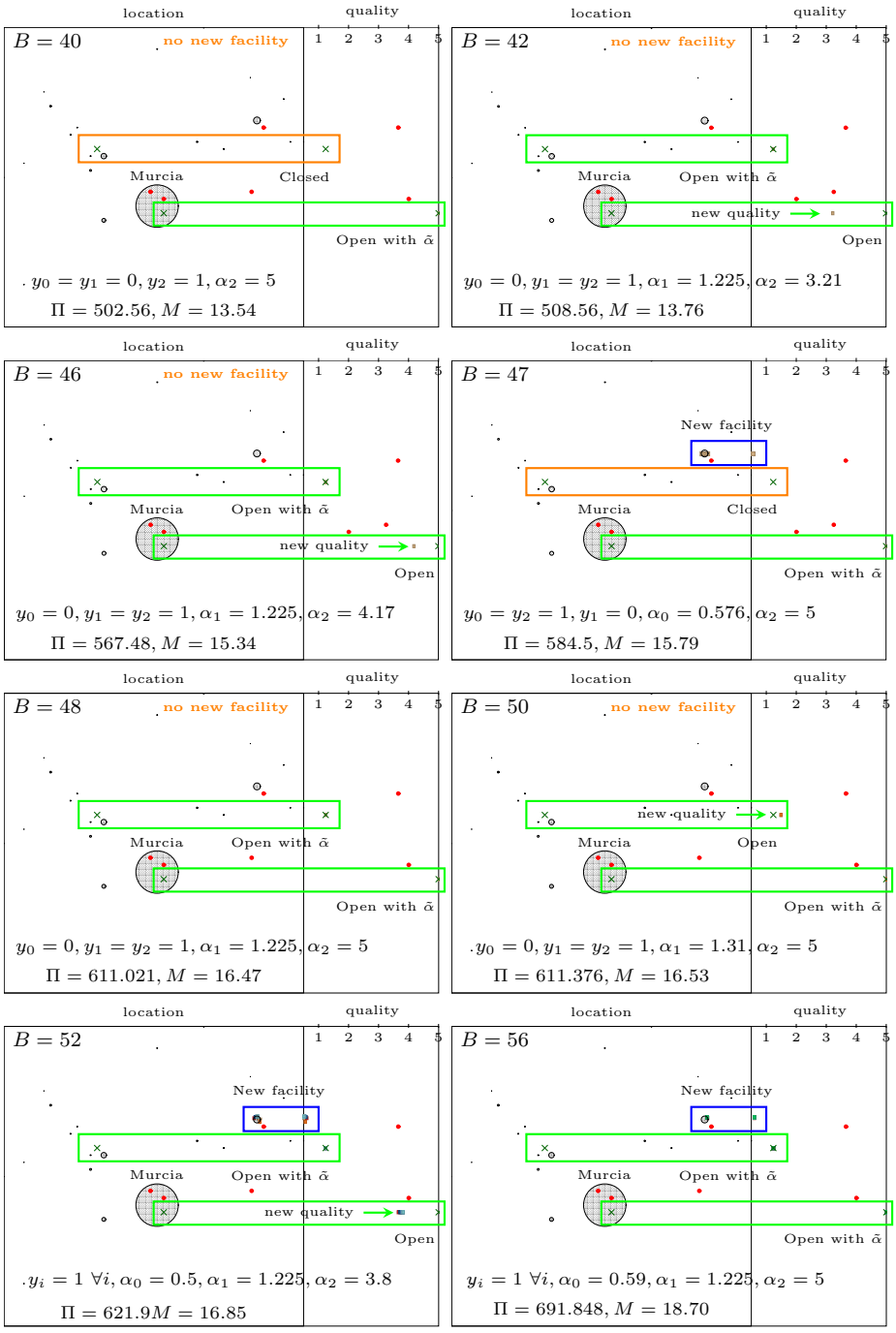

Figure 1: Solution of the small problem of Murcia with 21 demand points for different budgets

and our interval branch-and-bound method finished in less time) LocSol found near optimal solutions in 9 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.

The second test problem is the quasi-real example of the previous subsection. Concerning this problem, without any starting point, BONMIN found the optimal solution only for 1 out of the 15 instances; in the rest of problems it just found a local optimum. Similarly, LGO and Knitro found the optimal solution only once of the 15 cases. SCIP found the optimal solution 11 out of the 15 problems but, on average, it spent more than 5 hours on the problems it could solve. On the other hand, BARON was able to solve all the instances optimally. Comparing the results obtained by BARON and our method, we found that for the 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 leads to the opening of the new facility and also to keep opened 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 interval branch-and-bound 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 those instances BARON finds one global optimum point faster than our method finds (enclose) 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 is not evident, as both our interval branch-and-method and BARON took times in a wide range when solving these problems. Setting the time limit to 1 second, only 2 out of 15 problems were optimally solved by LocSol. Increasing the time limit to 10 seconds, the number of problems optimally solve increased to 6. As for the last 6 problems with the highest budget, both BARON and our method took longer CPU times, we run LocSol for these problems setting a time limit of 100 and 1000 seconds. LocSol found near optimal solutions in 4 out of the 6 problems in both cases.

In order to check the reliability and performance of BARON, we then solved a larger instance, generated randomly. It has 100 demand points, 3 existing facilities, 2 belonging to the expanding chain. Using our interval method for 5 different budgets, the enclosure of the global optima is reached in 106 seconds on average, while BARON did not find a near-optimal solution within an hour. For one problem, we let BARON to run until one day, but it could only find a good local solution, not the global optimum.

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 (see Belotti et al. (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 sBB 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 (see for instance Hansen and Walster (2004); Kearfott (1996); Ratschek and Rokne (1988)). Interval B&B methods were successfully applied for solving many types of continuous NLP problems, including facility location problems Fernández and Pelegrín (2001); Redondo et al. (2015); Tóth and Fernández (2010); Tóth et al. (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 searching region, we take care of the integer variables so as not to remove parts which 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 $\operatorname{wid} \boldsymbol{z} = \overline{z} - \underline{z}$, whereas 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 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 intervals $\boldsymbol{z}$ 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 Languages for interval analysis (Accessed 17-03-2022) which provide inclusion functions for the classical predeclared functions. And from them, and using the interval arithmetic, inclusion functions for general functions can be built automatically Tóth et al. (2007) (using, for instance, the natural interval extension Hansen and 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:

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

and

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

where $\boldsymbol{G}_2$ and $\boldsymbol{G}'_2$ are inclusion functions for G_2 and G'_2, respectively (see also Kearfott (1996); Pelegrín et al. (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 $g_j(z) \leq 0, j = 1, \ldots, r$, the constraints of the problem. In our case, z is the vector ns (with dimension $n = 2k + 4$), and constraints (5)-(11) can be transformed into non-positive form constraints to obtain functions $g_j(z)$.

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.

In the hybrid algorithm described in Section 5, the so-called breath-first strategy, which selects $\boldsymbol{z}^{(k)} = \arg\max\{\operatorname{wid} \boldsymbol{z} : \boldsymbol{z} \in \mathcal{L}_{\mathcal{W}}\}$, provides better results, as we will see.

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 i-th 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.

- However, if z_i is integer, then they will be $[\underline{z}_i, \lfloor \text{mid}\, z_i \rfloor]$ and $[\lceil \text{mid}\, z_i \rceil, \overline{z}_i]$; notice that if mid $\boldsymbol{z}_i$ is not integer, the points in the open interval $(\lfloor \text{mid}\, z_i \rfloor, \lceil \text{mid}\, z_i \rceil)$ are not integer, and therefore, are not feasible. Hence, in this way, it is assured that the bounds of the integer variables are always integers in all boxes.

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 fix the binary variable in the subboxes to 0 and 1, respectively.

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

(1) In the subbox with $y_j = 0$, we also set the variable α_j equal to 0 (the j-th facility is closed or not opened), so its width is 0 and it is no longer selected to be bisected in the subdivision rule; 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: $[\underline{\alpha}_j, \tilde{\alpha}_j - EPS]$, $[\tilde{\alpha}_j, \tilde{\alpha}_j]$ and $[\tilde{\alpha}_j + EPS, \overline{\alpha}_j]$, where EPS 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 after that.

(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. 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. As we have said, we denote by $g_j(z) \leq 0, j = 1, \ldots, r$, the constraints of the problem.

3.3.1. Feasibility test

We say that constraint $g_j(z) \leq 0$ is *certainly continuously satisfied* for an interval box $\boldsymbol{z}$ if $\overline{\boldsymbol{g}}_j(\boldsymbol{z}) \leq 0$, 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, when at least one constraint is undetermined, but none of the constraints are certainly continuously unfulfilled, 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, thus the feasibility test discards those 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 $\boldsymbol{z}$ which are integer points by the subdivision rule. This is specially useful for updating the best value found by the algorithm, 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 $\tilde{\boldsymbol{f}}$, (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 those 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 $\boldsymbol{\nabla f}(\boldsymbol{z}) = (\boldsymbol{\nabla}_1 \boldsymbol{f}(\boldsymbol{z}), \ldots, \boldsymbol{\nabla}_n \boldsymbol{f}(\boldsymbol{z}))^T$ 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., $0 \notin \boldsymbol{\nabla}_i \boldsymbol{f}(\boldsymbol{z})$. Let us denote by $\partial\mathcal{S}$ the boundary of the searching region $\mathcal{S}$ (where the constraints $g_j(z) \leq 0$ are not taken into account). In our problem $\mathcal{S} = (S, [0, \alpha_{\max}], [0, 1], \ldots, [0, \alpha_{\max}], [0, 1])$.

In continuous unconstrained problems, the facet $\boldsymbol{z}'$ ($\boldsymbol{z}' = (\boldsymbol{z}_1, \ldots, \underline{z}_i, \ldots, \boldsymbol{z}_n)$ or $\boldsymbol{z}' = (\boldsymbol{z}_1, \ldots, \overline{z}_i, \ldots, \boldsymbol{z}_n)$, depending on whether the function is decreasing of increasing, respectively) 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 $\boldsymbol{z}' \in \partial\mathcal{S}$, so $\boldsymbol{z}'$ is on the boundary, 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 $0 \notin \boldsymbol{\nabla}_i \boldsymbol{f}(\boldsymbol{z})$ 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 $0 \notin \boldsymbol{\nabla}_i \boldsymbol{f}(\boldsymbol{z})$ 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 $y_j, \alpha_j, j = 0, \ldots, k$ 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 the one that becomes more efficient.

3.3.4. Projected one-dimensional Newton method

In Fernández et al. (2007) a one-dimensional Newton method was introduced, which can be applied here on both certainly continuously strictly feasible or undetermined boxes, on their continuous quality variables.

If the box is 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 those 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 $\mathcal{L}_{\mathcal{S}}$ if either its size is smaller than a given tolerance, i.e., $\mathrm{wid}\,(\boldsymbol{z}) < \epsilon$, or the relative width of the inclusion of its objective value is less than a given tolerance, i.e, $\mathrm{wid}_{rel}(\boldsymbol{f}(\boldsymbol{z})) < \delta$.

4. A heuristic algorithm

How to invest the annual chain's budget B to maximize the revenue of the expanding chain is not an easy task. Should the new facility be opened? If so, where should it be located and which quality should it have? Should some of the existing facilities be upgraded or downgraded? If so, which ones and how much? Or is it better to close some of the existing facilities? Any individual decision we make about a facility has an influence in the others ones, as they are interrelated. The best global decision taking into account all the possible combinations is to be found. The combinatorial nature of the problem, as given by the binary variables, and the fact that for each combinatorial problem (for each integer assignment) we have a nonconvex NLP problem, as given by the continuous variables, make the problem extremely difficult to solve.

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 less 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 opened facility is given by the ratio of the income that it generates and the cost it incurres. More precisely,

$$profitab_j = \begin{cases} \dfrac{F\left(\sum_{i=1}^{i_{\max}} w_i \dfrac{y_0 u_{i0}(f_0,\alpha_0)}{y_0 u_{i0}(f_0,\alpha_0) + \sum_{j=1}^{k} y_j u_{ij}(\alpha_j) + \sum_{j=k+1}^{j_{\max}} \tilde{u}_{ij}}\right)}{G(f_0,\alpha_0) + R_0(\alpha_0)} & \text{if } j = 0 \\ \dfrac{F\left(\sum_{i=1}^{i_{\max}} w_i \dfrac{y_j u_{ij}(\alpha_j)}{y_0 u_{i0}(f_0,\alpha_0) + y_j \sum_{j=1}^{k} u_{ij}(\alpha_j) + \sum_{j=k+1}^{j_{\max}} \tilde{u}_{ij}}\right)}{A_j + R_j(\alpha_j) + V_j(\alpha_j)} & \text{if } j \neq 0 \end{cases}$$

For a given value 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 $\breve{B}$ be the available budget at any time. At the beginning, $\breve{B}$ is equal to B (chain's initial annual budget) minus the total cost needed to maintain all the opened 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 open and the existing chain-owned facilities are open with qualities $\tilde{\alpha}_j$, $j = 1, \ldots, k$. Other initial feasible solutions are randomly generated where the new facility is allowed to be open 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 **check_tabu_list** 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 has set $\varepsilon = 0.1$ in our implementation.

When more budget is required, the procedure **get_more_budget** (see Algorithm 2) decides whether to **reduce** (Algorithm 3) the quality of the facility with less 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 13-29 in Algorithm 1), the best of two options is implemented: (I) first **improve** (Algorithm 5) the qualities of the chain-owned facilities 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, those options are run three times each, assuming an available budget for the first procedure performed (**improve** or **open**, respectively) equal to $\lambda \breve{B}$: firstly with $\lambda = 1$ (hence, using all the available budget), secondly choosing 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 **improve** distributes the available budget through a greedy strategy. The quality of the facility with the highest profitability can vary as much as needed (without surpassing the budget and if Π improves). After the most profitable facility, as long as there is still some budget left, the same is done with the facility with the second highest profitability, and so on.

Procedure **open** requires at least a budget $B^{\min}$ 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 $B^{\min}$ 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 the procedure **reopen**, $minB$ is a lower bound of the minimum budget needed to reopen any facility. We have set it equal to the annual operating cost with minimum quality of the cheapest existing facility.

In Algorithms 2, 5, 6 and 7 we use a modification of the *multi-start* Weiszfeld-like algorithm presented in Redondo et al. (2009) (the modification checks that the budget constraint is satisfied).

Algorithm 1: Heuristic scheme

```
   // Tabu list of explored solution settings.
1  Create an empty tabu list.
2  for iter := 1:numIniSol do
3      if iter = 1 then
4          Let ns be the solution with the current setting of chain-owned
            facilities.
5      else
6          Let ns be a feasible randomly generated solution.
           check_tabu_list(ns)
7      In case Π improves, even if B̆ > 0, close chain-owned facilities.
       check_tabu_list(ns)
       /* If the budget is not enough for the opened chain-owned
          facilities, some qualities must be reduced and/or some
          facilities must be closed.                                  */
8      while B̆ < 0 and there are opened chain-owned facilities do
9          get_more_budget
10     check_tabu_list(ns)
11     repeat
           /* If we have used all the available budget, we reduce
              the quality or close the facility with less
              profitability, to get some budget.                      */
12         if B̆ = 0 then
13             get_more_budget
14             check_tabu_list(ns)
15         Do the best option (I or II):
           // I
16         improve the qualities of chain-owned facilities.
17         open a new facility (at less a budget B^min is required).
18         reopen facilities previously closed.
           // II
19         if B̆ < B^min and the new facility has not been opened then
20             repeat
21                 get_more_budget
22             until B̆ ≥ B^min or no chain-owned facilities are opened
23         open a new facility (at less a budget B^min is required).
24         improve the qualities of chain-owned facilities.
25         reopen facilities previously closed.
26         check_tabu_list(ns)
27     until Π does not increase or B̆ > 0
28     if ns has not been discarded then
29         Save ns and its previous configurations in the for-loop in the tabu
            list.
```

Algorithm 2: Procedure: **get_more_budget**

Tentatively **reduce** the quality of the facility with the lowest profitability whose quality can be reduced, j_1.
Using a few iterations of a Weiszfeld-like method, estimate the profit that can be achieved with the previous quality reduction, $\Pi_{j_1}^{est1}$.
Restart the problem to its previous setting.
Tentatively **close** down the least profitable facility, j_1.
Using a few iterations of a Weiszfeld-like method, estimate the profit that can be achieved with the previous closure, $\Pi_{j_1}^{est2}$.
if $\Pi_{j_1}^{est1} \geq \Pi_{j_1}^{est2}$ **then**
    Choose to **reduce** the quality of facility j_1.
else
    Choose to **close** down facility j_1.

Algorithm 3: Procedure: **reduce**

Rank the opened chain-owned facilities according to its profitability.
Let j_1 be the facility with the lowest profitability whose quality can be reduced, reduce its quality, $\alpha_{j_1}^{old}$, down to $\alpha_{j_1} = \max\{\alpha_{\min}, \alpha_{j_1}^{lp}\}$ where $\alpha_{j_1}^{lp}$ is the solution of the equation

$$profitab_{j_1} = profitab_{j_2}$$

and j_2 is the facility with the second lowest profitability.
if $j_1 = 0$ **then**
    $\breve{B} := \breve{B} + G(f_0^{old}, \alpha_0^{old}) + R_0(\alpha_0^{old}) - G(f_0^{old}, \alpha_0) - R_0(\alpha_0)$
else
    $\breve{B} := \breve{B} + R_{j_1}(\alpha_{j_1}^{old}) + V_{j_1}(\alpha_{j_1}^{old}) - R_{j_1}(\alpha_{j_1}) - V_{j_1}(\alpha_{j_1})$

Algorithm 4: Procedure: **close**

Rank the opened chain-owned facilities according to its profitability.
Close down the least profitable facility, j_1.
if $j_1 = 0$ **then**
    $\breve{B} := \breve{B} + G(f_0^{old}, \alpha_0^{old}) + R_0(\alpha_0^{old})$
else
    $\breve{B} := \breve{B} + A_{j_1} + R_{j_1}(\alpha_{j_1}^{old}) + V_{j_1}(\alpha_{j_1}^{old}) - C_{j_1}$

Algorithm 5: Procedure: **improve**

while $\breve{B} > 0$ *and* $\alpha_j < \alpha_{\max}^j$ *for some* $j \in \{0, \ldots, k\}$ *and* Π *improves* **do**
  Rank the opened chain-owned facilities according to its profitability.
  Select the most profitable facility whose quality can be improved, j_1.
  if $j_1 = 0$ **then**
    Improve the location f_0^{old} and the quality α_0^{old} 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.
    $\breve{B} := \breve{B} + G(f_0^{old}, \alpha_0^{old}) + R_0(\alpha_0^{old}) - G(f_0, \alpha_0) - R_0(\alpha_0)$
  else
    Determine the maximum quality that the facility j_1 can have solving the equation

$$R_{j_1}(\alpha_{j_1}) + V_{j_1}(\alpha_{j_1}) - (\breve{B} + R_{j_1}(\alpha_{j_1}^{old}) + V_{j_1}(\alpha_{j_1}^{old})) = 0$$

    Improve the quality of facility j_1 using a Weiszfeld-like method whenever facility j_1 is still the most profitable one.
    Let α_{j_1} be its new quality and $\alpha_{j_1}^{old}$ its quality before applying the procedure.
    $\breve{B} := \breve{B} + R_{j_1}(\alpha_{j_1}^{old}) + V_{j_1}(\alpha_{j_1}^{old}) - R_{j_1}(\alpha_{j_1}) - V_{j_1}(\alpha_{j_1})$

Algorithm 6: Procedure: **open**

if $\breve{B} \geq B^{\min}$ **then**
  Locate a new facility using the budget $\breve{B}$ using a Weiszfeld-like method.
  $\breve{B} := \breve{B} - G(f_0, \alpha_0) - R_0(\alpha_0)$

Algorithm 7: Procedure: **reopen**

while $\breve{B} > minB$ *and not all the chained-owned facilities are open and* Π *improves* **do**
  for *all closed chained-owned facilities* j **do**
    Determine the maximum quality that facility j can have solving the equation

$$R_j(\alpha_j) + V_j(\alpha_j) + A_j - (\breve{B} + C_j) = 0$$

    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**
    Restart the problem to its previous setting.
  else
    $\breve{B} := \breve{B} + C_{j_1} - R_{j_1}(\alpha_{j_1}) - V_{j_1}(\alpha_{j_1}) - A_{j_1}$

5. A hybrid heuristic

The heuristic algorithm described in the previous section explores all the feasible set 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 the near-optimality of the non-discarded areas covered by the boxes after that moment. Hence, it would be a good idea to execute first some iterations of the interval B&B so as to discard those non-promising areas, and then to execute the heuristic algorithm only in those 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 volume discarded and the CPU time required, specially taking into account that after that the heuristic algorithm will be applied in each box. After some preliminary computational studies, we found that the interval B&B method should be running at most 10% of its total running time. In particular, in our computational studies, we let the algorithm run for 10 seconds.

The result of that execution is a long list of boxes. Since executing the heuristic algorithm in each of those boxes will be extremely time-consuming, the number of boxes is reduced by joining boxes which are closer than ϵ (the maximum width allowed in the stopping rule for a box to be sent to the solution list). Specifically, for each box in the list (not discarded by the B&B method), we check its distance to the rest of the boxes (if any) in the list, and if one of them is closer enough, then those two boxes are replaced by their interval hull (the smallest box containing them). If no close enough box exists, the new box is placed at the end of the list, and the process is repeated with the next box in the list, until no more boxes can be joined.

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 those boxes, at least one initial solution was assigned to any box. The size of a box $\boldsymbol{z}$ is obtained as follows:

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

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.

We also carried out a preliminary study to investigate whether the rule employed to select the next box to be subdivided in the interval B&B algorithm has an influence in the efficiency of the hybrid algorithm. Two rules were investigated: the best-first and the breath-first strategies (see Section 3.1). Interestingly, the breath-first strategy provided better results. Remember that the interval B&B algorithm is stopped prematurely, and this rule allows to discard more volume of the search space at early stages than the best-first strategy.

In some of the boxes in the reduced list some of the binary variables are fixed, which means that the corresponding facilities are either closed or opened. Since in those boxes those 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: when closing a facility or checking whether they all are closed, it must be taken into account if the facility can be closed; when trying to reopen an existing facility of the chain, we must consider whether it can be opened; and when locating the new facility, we must check whether it can be opened.

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 that 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 open 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 those variables must lie. Note that this location of the new facility may 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 qualities of the facilities. 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. Hardware and software

All the computational results have been obtained under Linux on an AMD Athlon(tm) 64 X2 with 2.2GHz CPU and 2GB 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 et al. (1995). We have used a time limit of 12 hours for each run.

6.2. Test problems

In order to check the performance of the methods and their variants, we have generated different test problems varying some parameters. Namely, we have fixed the number of demand points ($n = 100$), but varied the number of existing facilities ($m = 3,\ 5$) 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: $p_i, f_j \in ([0, 10], [0, 10])$
- demand: $w_i \in (0, 10]$
- quality of existing facilities: $\tilde{\alpha}_j \in [0.5, 5]$
- income per unit of goods sold ($F(\cdot) = c \cdot M(\cdot)$): $c \in [12, 14]$
- 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 $\rho = 3\sqrt{n + 50} - 20$
- 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 found that in none of them the original qualities remain unchanged when solving the problem. This is because the qualities associated to the facilities were far from being the optimal ones, due to their randomness. Thus, in order to generate problems which are closer to reality, we performed a pre-optimization for the qualities of the existing facilities, as follows. First, 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 $\alpha_j = \tilde{\alpha}_j,\ \forall j = 1, \dots k$). Second, we

run our B&B method for the budget $\tilde{B}$ in order to optimize the qualities of the existing facilities, by fixing $y_0 = 0$ and $y_j = 1,\ \forall j = 1, \dots k$. Finally, we set $\tilde{\alpha}_j$ equal to the optimal value of α_j of the previous 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 were generated.

6.3. About the interval branch-and-bound method

In order to analyse the efficiency of each of the discarding tests, we solved the test problems using the following versions of the interval B&B algorithm:

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 and 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 1 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 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 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.

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

m	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%
		Aver.	**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%
		Aver.	**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%
		Aver.	**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%
		Aver.	**14203**	**6911**	**49%**	**4103**	**59%**	**4128**	**101%**

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.

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 those problems it does not reduce the CPU time. The higher the budget, the

Table 2: Average computational times for each budget settings using the different versions of the interval B&B method

Budget	Basic	Mono	$\frac{\text{Mono}}{\text{Basic}}$	Newton	$\frac{\text{Newt}}{\text{Mono}}$	NonConc	$\frac{\text{NonConc}}{\text{Newt}}$
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%
Aver.	**12225**	**3265**	**27%**	**1997**	**61%**	**2008**	**101%**

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 those 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 those 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.

Comparing the problem settings, the easiest problems are, in general, the ones with one chain-owned facility. The reason is clear: those problems have one variable less than the others. Interestingly, among those 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 2 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. This is because of the way we generated the random problems: remember that we performed a pre-optimization of the qualities of the existing facilities, so it is more difficult to find a better solution. 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. In general, compared to the Mono version, it needs 61% of its CPU time, and compared to the Basic method, only 16% on average.

6.4. About the heuristics

In the heuristic (see Algorithm 1) we used $numIniSol = 10$ in all instances and 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, so the overall number of initial solutions for a problem could be much greater than 10. In the hybrid algorithm, in all instances, we used the boxes generated by the B&B with the breath-first strategy.

Both the heuristic and the hybrid heuristic algorithms were run 10 times for each instance, and we calculated the relative difference in terms of objective value for the solution obtained by the interval B&B, and the best solution of the heuristic and the hybrid heuristic in the 10 runs, but also how many times the best solution was found in the 10 runs for the heuristic and the hybrid heuristic methods. The averages for each (m, k) settings are shown in Table 3, together with the standard deviation in brackets. As in Table 1, the budget is shown in relative terms to $\tilde{B}$. 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 in the objective value. On average, the difference from the global optimum is 0.09% in both cases and, in the worst case, it is only 0.94% for the heuristic and 0.44% for the hybrid method. For both algorithms, the average number of times that they found the best solution they could in the 10 runs is greater than 5, being the results obtained by the hybrid heuristic slightly better. The elapsed time for the heuristic algorithms is much smaller than for the interval B&B, the hybrid being, on average, the fastest one, especially when $B > \tilde{B}$.

7. Conclusions and future research

When a chain wants to invest in order to expand itself in a given geographical area, it can choose among opening a new facility, varying the quality of the existing chain-owned facilities up or down, closing some of those facilities in order to allocate the money formerly devoted to them to other of its facilities or to the new one (if the chain opens it), or a combination of all those possibilities. In this paper, the continuous competitive facility location and design model introduced in Fernández et al. (2007) has been extended to accommodate all those possibilities.

The resulting model is a hard-to-solve MINLP problem, for which the existing solvers fail: only BARON Sahinidis (2017) and LocSol Benoist et al. (2011) seem to be able to solve some small instances, although they can fail at solving

Table 3: Results for the problems with 100 demand points using the heuristic and the hybrid heuristic algorithms

			Difference in obj (%)		Times found		Computational time (sec.)		
m	k	Budget	heur.	hybr.	heur.	hybr.	B&B	heur.	hybr.
3	1	80%	0.11 (0.4)	0.04 (0.1)	5.2 (4.3)	5.3 (3.8)	3293.1 (6178.5)	26.6 (11.7)	35.7 (24.8)
		100%	0.00 (0.0)	0.09 (0.3)	4.8 (4.0)	4.9 (4.3)	4388.3 (5627.4)	29.0 (14.8)	30.4 (22.9)
		120%	0.01 (0.0)	0.00 (0.0)	5.5 (4.0)	6.1 (4.2)	1927.5 (3417.7)	33.8 (14.3)	29.5 (17.6)
		150%	0.03 (0.1)	0.19 (0.4)	5.7 (5.0)	3.6 (4.6)	174.4 (233.8)	43.6 (21.7)	43.2 (28.1)
		200%	0.00 (0.0)	0.07 (0.2)	4.4 (4.9)	4.4 (5.0)	152.5 (200.5)	60.4 (56.5)	41.9 (23.0)
	2	80%	0.08 (0.3)	0.00 (0.0)	6.0 (4.2)	5.5 (4.3)	714.2 (810.4)	44.0 (8.5)	45.0 (10.9)
		100%	0.14 (0.3)	0.44 (1.2)	3.4 (4.2)	4.6 (4.9)	2313.7 (5060.9)	57.0 (10.5)	52.2 (10.5)
		120%	0.05 (0.1)	0.08 (0.1)	5.2 (4.2)	3.5 (4.5)	4754.9 (13591.9)	112.8 (54.4)	82.4 (32.8)
		150%	0.08 (0.2)	0.00 (0.0)	6.9 (4.6)	8.0 (3.8)	385.0 (418.6)	142.0 (79.3)	112.2 (59.0)
		200%	0.02 (0.0)	0.00 (0.0)	8.4 (3.5)	6.9 (4.4)	307.0 (401.2)	216.5 (217.3)	179.5 (199.9)
5	1	80%	0.05 (0.2)	0.00 (0.0)	2.6 (4.0)	5.6 (3.5)	339.3 (384.0)	34.9 (14.4)	35.6 (21.2)
		100%	0.00 (0.0)	0.00 (0.0)	4.6 (4.1)	5.7 (4.2)	332.7 (589.4)	40.3 (13.6)	39.7 (21.7)
		120%	0.00 (0.0)	0.00 (0.0)	6.0 (3.5)	6.3 (3.4)	195.8 (320.0)	62.1 (49.8)	39.0 (19.9)
		150%	0.14 (0.4)	0.27 (0.7)	4.8 (4.5)	5.3 (4.5)	92.2 (143.7)	80.1 (59.4)	53.3 (27.8)
		200%	0.07 (0.2)	0.04 (0.1)	6.1 (4.1)	6.5 (4.3)	47.2 (32.3)	90.2 (57.5)	64.8 (32.2)
	2	80%	0.00 (0.0)	0.00 (0.0)	2.9 (3.0)	3.6 (3.4)	5211.2(5512.2)	59.7 (26.8)	59.9 (18.1)
		100%	0.07 (0.2)	0.07 (0.2)	4.5 (3.1)	4.0 (3.6)	7178.1(13283.0)	74.8 (28.3)	78.5 (28.8)
		120%	0.94 (1.6)	0.29 (0.5)	2.8 (4.4)	4.3 (4.4)	4162.4(9710.7)	137.5 (132.2)	95.8 (57.7)
		150%	0.04 (0.1)	0.20 (0.4)	5.3 (4.2)	3.2 (4.6)	3615.5(10319.5)	180.3 (200.3)	155.3 (169.9)
		200%	0.04 (0.1)	0.07 (0.2)	6.4 (4.4)	5.6 (4.4)	347.6(133.7)	278.0 (254.0)	245.3 (244.5)
Aver.			**0.09**	**0.09**	**5.08**	**5.15**	**1996.62**	**90.18**	**75.96**

other instances, too. Thus, new algorithms are required to cope with the new problem.

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. In particular, the subdivision rule has been modified to take into account the integer variables, as well as the monotonicity test, the projected one-dimensional Newton method and the projected one-dimensional non-concavity test. The idea for the modification of those tests is to relax the integrality of the variables and, when it is the moment to rule out subregions, to take once again the integrality into account so as not to discard promising parts. 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 Weisfeld-like algorithm to update the location of the new facility and the qualities of the chain-owned facilities (new and existing ones). It follows different strategies in a greedy way, but also keeps a tabu list to avoid unnecessary searches. Third, a hybrid heuristic is also proposed, which makes use of boxes provided by the interval B&B obtained at early stages of its iterative process, and applies to them the ad-hoc heuristic.

The so-called Newton variant of the exact interval B&B algorithm can solve medium size problems (with up to 100 demand points and 5 existing facilities) without difficulty in around 30 minutes of CPU time on average. As for the heuristic and hybrid algorithms, they both are quite effective, finding solutions whose objective value is very close to the optimum. On average, more than half of the times they were run for each instance, both algorithms found a (near)optimal solution. Furthermore, the computational time required by both algorithms is much smaller than the interval B&B. Specifically, the hybrid heuristic is, on average, the fastest one (it can solve the problems with 100 demand points in just 75 seconds). Note, however, that it is a heuristic algorithm and it cannot guarantee that the optimal solution has been found, and in the best case it just finds *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 and Pelegrín (2001); Tóth and Fernández (2010) and to design new ones. We point out that the algorithms introduced in this paper could be parallelized to make use of high-performance computing devices Redondo et al. (2008, 2011). The model could be extended in several ways, such as to take into account the possibility of locating more than one new facility Redondo et al. (2011), or to take the possible reaction of the competitors into account, leading to a Stackelberg (or leader-follower) problem Redondo et al. (2013). Similar location models, using other customer choice rules Fernández et al. (2017); Fernández et al. (2019) should also be investigated.

Acknowledgements

We would like to thank Juana L. Redondo and Pilar M. Ortigosa for their collaboration in an early version of the paper.

References

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., . COUENNE (Convex Over and Under ENvelopes for Nonlinear Estimation). URL: `https://projects.coin-or.org/Couenne`. [Accessed 17-03-2022].

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, 597–634.

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, 299.

Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wachter, A., 2005. An Algorithmic Framework for Convex Mixed Integer Nonlinear Programs. Technical Report. IBM Research Report. URL: `https://www.coin-or.org/Bonmin/`. [Accessed 17-03-2022].

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

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

Byrd, R.H., Nocedal, J., Waltz, R., 2006. KNITRO: An integrated package for nonlinear optimization. Springer. p. 35–59. URL: `https://www.artelys.com/solvers/knitro/`. [Accessed 17-03-2022].

Drezner, T., 1995. Competitive facility location in the plane, in: Drezner, Z. (Ed.), Facility location: a survey of applications and methods. Springer, Berlin. Springer Series in Operations Research and Financial Engineering, pp. 285–300.

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, 1720–1730.

Drezner, Z., Hamacher, H.W. (Eds.), 2002. Facility location: applications and theory. Springer, Berlin.

Eiselt, H., Marianov, V. (Eds.), 2015. Applications of location analysis. Springer.

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

Fernández, J., Hendrix, E., 2013. Recent insights in Huff-like competitive facility location and design. European Journal of Operational Research 227, 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, 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, 1274–1287.

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

Fernández, J., G.- Tóth, B., L. Redondo, J., M. Ortigosa, P., 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. doi:`https://doi.org/10.1016/j.cor.2018.08.001`.

Fourer, R., Gay, D., Kernighan, B., 2003. AMPL. A Modeling Language for Mathematical Programming (second edition). Duxbury-Thomson. URL: `https://ampl.com/`. [Accessed 17-03-2022].

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

Hammer, R., Hocks, M., Kulisch, U., Ratz, D., 1995. C++ Toolbox for verified computing I: basic numerical problems: theory, algorithms and programs. Springer-Verlag, Berlin.

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

Kearfott, R., 1996. Rigorous global search: continuous problems. Kluwer, Dordrecht.

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

Knüppel, O., 1993. PROFIL/BIAS - A Fast Interval Library. Computing 53, 277–287.

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

Languages for interval analysis, Accessed 17-03-2022. Available at `https://www.cs.utep.edu/interval-comp/intlang.html`.

Laporte, G., Nickel, S., Saldanha da Gama, F. (Eds.), 2015. Location Science. Springer.

Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A., Munson, T., Palkar, P., Sharma, M., . MINOTAUR: Mixed-Integer Optimization Toolbox-Algorithms, Underestimators, Relaxations. URL: `https://wiki.mcs.anl.gov/minotaur/index.php/MINOTAUR`. [Accessed 17-03-2022].

Mirchandani, P., Francis, R. (Eds.), 1990. Discrete location theory. Wiley-Interscience, New York.

Misener, R., Floudas, C., 2014. ANTIGONE: Algorithms fon coNTinuous / Integer Global Optimization of Nonlinear Equations. Journal of Global Optimization 59, 503–526.

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

Pintér, J.D., 1997. Lgo—a program system for continuous and lipschitz global optimization, in: Developments in global optimization. Springer, pp. 183–197.

Plastria, F., 2001. Static competitive facility location: an overview of optimisation approaches. European Journal of Operational Research 129, 461–470.

Rall, L., 1981. Automatic differentiation, techniques and applications. Lecture Notes in Computer Science, Springer, Berlin.

Ratschek, H., Rokne, J., 1988. New computer methods for global optimization. Ellis Horwood, Chichester.

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, 983–1005.

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

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

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

Sahinidis, N.V., 2017. BARON 17.8.9: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual*. URL: `http://www.minlp.com/downloads/docs/baron%20manual.pdf`. [Accessed 17-03-2022].

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, 9–17.

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

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

Trespalacios, F., Grossmann, I., 2014. Review of mixed-integer nonlinear and generalized disjunctive programming methods. Chemie Ingenieur Technik 86, 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, 563–593. URL: `https://scip.zib.de/`. [Accessed 17-03-2022].

Westerlund, T., Pörn, R., 2002. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optimization and Engineering 3, 253–280. URL: `https://www.gams.com/latest/docs/S_ALPHAECP.html`. [Accessed 17-03-2022].