An Evolutionary Algorithm for the Leader-Follower Facility Location Problem with Proportional Customer Behavior[⋆]

Benjamin Biesinger, Bin Hu, and Günther Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9-11/1861, 1040 Vienna, Austria
{biesinger|hu|raidl}@ads.tuwien.ac.at

Abstract. The leader-follower facility location problem arises in the context of two non-cooperating companies, a leader and a follower, competing for market share from a given set of customers. In our work we assume that the firms place a given number of facilities on locations taken from a discrete set of possible points. The customers are assumed to split their demand inversely proportional to their distance to all opened facilities. In this work we present an evolutionary algorithm with an embedded tabu search to optimize the location selection for the leader. A complete solution archive is used to detect already visited candidate solutions and convert them into not yet considered ones. This avoids unnecessary time-consuming re-evaluations, reduces premature convergence and increases the population diversity at the same time. Results show significant advantages of our approach over an existing algorithm from the literature.

Keywords: competitive facility location, evolutionary algorithm, solution archive, bi-level optimization

1 Introduction

We consider a competitive facility location problem in which two decision makers, a leader and a follower, compete for market share. They choose given numbers of facility locations from a finite set of possible positions in order to satisfy clients, whereas the leader starts by placing all of his facilities. Each customer has a fixed demand which is assumed to be fulfilled by all opened facilities together inversely proportional to their distance. In this respect the considered model is for many real-world scenarios more precise than simpler leader-follower location problems where a customer's whole demand is assumed to be satisfied by its closest facility only. Demands correspond to the buying power of the customers, so the turnover of the competing firms increases with the amount of fulfilled demand.

⋆ This work is supported by the Austrian Science Fund (FWF) under grant P24660.

We propose an evolutionary algorithm (EA) that tries to find best possible facility locations for the leader so that his turnover is maximized with respect to a follower who is assumed to place his facilities optimally, i.e., aiming at lowering the leader's revenue. Therefore, for a given set of facility locations of the leader we have to find an optimal set of facility locations of the follower in order to obtain an accurate revenue value the leader can achieve. This makes the problem a bi-level optimization problem. Finding the optimal locations for the follower, which can be seen as evaluating a candidate leader solution, unfortunately is a time-consuming procedure so we want to avoid unnecessary computations. Consequently, we employ a complete solution archive which is a data structure that stores all generated candidate solutions and converts created duplicates into guaranteed not yet considered solutions. Using this archive together with a tabu-search for locally improving solutions within the EA, we are able to reduce premature convergence, loss of diversity and, as already mentioned before, costly re-evaluations of duplicates.

In Section 2 we define the problem more formally. Related work is presented in Section 3, which is followed by a description of a mathematical model for the leader-follower facility location problem with proportional customer behavior in Section 4. Section 5 introduces our evolutionary algorithm and its extensions. Section 6 discusses our computational results and compares our method to an approach from the literature. Finally, we draw conclusions in Section 7 and give an outlook on further promising research questions.

2 Problem Definition

In the following we will formally define the leader-follower facility location problem with proportional customer behavior. Given are the numbers $r \geq 1$ and $p \geq 1$ of facilities to be opened by the leader and follower, respectively, and a weighted complete bipartite graph $G = (I, J, E)$ where $I = \{1, \ldots, m\}$ represents the set of potential facility locations, $J = \{1, \ldots, n\}$ represents the set of customers, and $E = I \times J$, is the set of edges indicating corresponding assignments. Let $w_j > 0, \forall j \in J$, be the demand of each customer, which corresponds to the turnover to be earned by the serving facilities, and $d_{ij} \geq 0, \forall (i, j) \in E$, be the distances between customers and potential facility locations. The goal for the leader is to choose exactly p locations from I for opening facilities in order to maximize her turnover under the assumption that the follower in turn chooses r locations for his facilities optimally maximizing his turnover.

Each customer j splits her demand over all opened facilities. The amount of demand that a facility fulfills is inversely proportional to its distance to the customer. In the following we give a formal definition of a candidate solution and the turnover computation. Let (X, Y) be a candidate solution to our leader-follower facility location problem, where $X \subseteq I,\ |X| = r$, is the set of locations chosen by the leader and $Y \subseteq I,\ |Y| = p$, is the associated set of follower locations. Furthermore, let $x_i = 1$ if $i \in X$ and $x_i = 0$ otherwise, and $y_i = 1$ if

$i \in Y$ and $y_i = 0$ otherwise, $\forall i \in I$. Then, the turnover of the follower is

$$p^{\mathrm{f}} = \sum_{j \in J} w_j \frac{\sum_{i \in I} \frac{1}{d_{ij}+1} x_i}{\sum_{i \in I} \frac{1}{d_{ij}+1} x_i + \sum_{i \in I} \frac{1}{d_{ij}+1} y_i}$$

and the turnover of the leader is

$$p^{\mathrm{l}} = \sum_{j \in J} w_j - p^{\mathrm{f}}.$$

Note that one is added to the original distances d_{ij} just to avoid numerical problems with zero distances which might occur when considering the same locations for facilities and customers.

3 Related Work

Competitive facility location problems are an old type of problem introduced by Hotelling [8] in 1929. He considers two sellers placing one facility each on a line. In the last years many variations were considered that differ in the way the competitors can open their facilities and in the behavior of the customers. Kress and Pesch give an overview of competitive location problems in networks in [11].

The discrete $(r|p)$-centroid problem is a competitive facility location problem introduced by Hakimi [7]. In this problem two decision makers can place given numbers of facilities on specific locations and each customer's demand is always fulfilled by the closest facility. Alekseeva et al. [1–3] present several heuristic and exact solution approaches. Laporte and Benati [13] developed a tabu search and Roboredo and Pessoa [17] describe a branch-and-cut algorithm.

The leader-follower facility location problem with proportional customer behavior which we consider here differs only in the way how customer demands are satisfied. For this frequently more realistic problem variant not much previous work exists, and unfortunately it is not trivial to extend existing approaches for the $(r|p)$-centroid problem. Kochetov et al. [10] developed a matheuristic for a more general problem variant that contains our problem as a special case. They assume that for each location several so-called design scenarios are possible. All of a location's design scenarios have different fixed costs and different attractiveness for the customers. Both competitors have a fixed budget and must choose the facility locations and the design scenarios for these locations in order to maximize their profit. In their work the customers split their demand proportionally to the attractiveness of a facility and inversely proportional to the distance to each facility. The authors suggest an alternating heuristic to solve this problem which is derived from an alternating heuristic developed for the $(r|p)$-centroid problem with continuous facility locations in [4]. Based on a starting solution for the leader they find the optimal facility locations for the follower. This follower solution is subsequently chosen as leader solution and the optimal follower solution is found again. This procedure is repeated until a solution is obtained which has already been generated. In Section 6 we compare our approach to their algorithm.

Vega et al. [21] give an overview on the different customer choice rules of competitive multifacility location problems. They consider six different scenarios of customer behavior, including binary, partially binary, proportional as well as essential and unessential goods. The authors assume that the facilities can be placed anywhere on the plane and give discretization results for several customer choice rules.

Fernández and Hendrix [5] study recent insights in Huff-like competitive facility location and design problems. In their survey article they compared three different articles [12, 20, 19] describing all the same basic model. In all three papers, for each facilitiy a quality level has to be determined similar to the design scenarios used in Kochetov [10] and fixed costs for opening facilities incur. Küçükaydin et al. [12] and Saidani et al. [19] assume that the competitor is already in the market and in Sáiz et al. [20] focus on finding a nash equilibrium of two competitors entering a new market opening only one facility each.

4 Mathematical Model

We present a mathematical non-linear bi-level model for our problem which is derived from Kochetov et al. [10]. Let $v_{ij} = \frac{1}{d_{ij}+1}$ be the attractiveness of location i for customer j. The upper level problem (leader's problem) is:

$$\max \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} x_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i^*} \tag{1}$$

s.t.

$$\sum_{i \in I} x_i = p \tag{2}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{3}$$

where $(y_1^*, \ldots, y_m^*)$ is an optimal solution to the lower level problem (follower's problem):

$$\max \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} y_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i} \tag{4}$$

s.t.

$$\sum_{i \in I} y_i = r \tag{5}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{6}$$

The objective functions (1) and (4) maximize the sum of the fulfilled demand by the leader and the follower, respectively, considering the splitting over the facilities inversely proportional to their distances. Constraint (2) ensures that the leader opens exactly p facilities and, similarly, constraint (5) guarantees that the follower places exactly r facilities. Note that the follower in principle is allowed to open facilities at the same locations as the leader. All of the x_i variables are considered constants in the follower's problem.

In order to be able to solve the follower's problem more efficiently Kochetov et al. [10] suggest a linear transformation of this model, which is as follows. First, we introduce two new kinds of variables:

$$z_j = \frac{1}{\sum_{i \in I} v_{ij} x_i + \sum_{i \in I} v_{ij} y_i} \qquad \forall j \in J \tag{7}$$

and

$$y_{ij} = w_j z_j v_{ij} y_i \qquad \forall i \in I, j \in J. \tag{8}$$

Variables y_{ij} have the intuitive meaning that they are the demand of customer j that is supplied by the follower facility at location i. It is obvious that if we are able to model the non-linear equation (8) in a linear way such that equation (7) is valid we get a model that is equivalent to (4–6). This is realized by the following mixed integer linear program (MIP):

$$\max \sum_{j \in J} \sum_{i \in I} y_{ij} \tag{9}$$

$$\text{s.t.} \quad (5), (6) \text{ and}$$

$$\sum_{i \in I} y_{ij} + w_j z_j \sum_{i \in I} v_{ij} x_i \leq w_j \qquad \forall j \in J \tag{10}$$

$$y_{ij} \leq w_j y_i \qquad \forall i \in I, j \in J \tag{11}$$

$$y_{ij} \leq w_j v_{ij} z_j \leq y_{ij} + W(1 - y_i) \qquad \forall i \in I, j \in J \tag{12}$$

$$y_{ij} \geq 0, z_j \geq 0 \qquad \forall i \in I, j \in J \tag{13}$$

Objective function (9) maximizes the turnover obtained by the follower. Constraints (10) set the variables y_{ij} by restricting them not exceed the total demand of customer j minus the demand captured by the leader. The fact that a facility location i can only get some turnover from customer j when the follower opens a facility there is ensured by constraints (11). Finally, equation (8) is fulfilled because of constraints (12).

Constant W is chosen large enough, so that an optimal solution to this model satisfies equations (7), i.e., $W = \max_{j \in J}(w_j) \cdot \max_{i \in I, j \in J}(v_{ij}) \cdot \max_{j \in J}(z_j)$, where $\max_{j \in J}(z_j) \leq \max_{j \in J}(1/\sum_{i \in I} v_{ij} x_i)$ because of constraints (10). Due to constraint (12) with its W, the linear programming (LP) relaxation of this model unfortunately is relatively weak, therefore finding an optimal solution to this model using a general purpose mixed integer programming solver like CPLEX is time-consuming even for small instances. Nevertheless, this model is still easier to solve than (4–6) directly.

5 Evolutionary Algorithm

In this section we present an EA that aims to find the optimal solution to the leader's problem. We use an incomplete solution representation only storing the facilities of the leader indicated by the binary vector $x = (x_1, \ldots, x_m)$. For augmenting the incomplete leader solution, which can also be seen as the evaluation of a candidate leader solution, the follower's problem has to be solved. As solving this problem exactly is time-consuming, a greedy evaluation procedure is used for approximating the quality of intermediate leader solution candidates, which is described in the next section. Only at the end of the EA the best solution found is evaluated using the MIP of Section 4 to get an exact objective value. After explaining the greedy solution evaluation we will introduce the EA with its variation operators, the complete solution archive, and finally the embedded tabu-search-based local improvement method.

5.1 Greedy Solution Evaluation

The greedy evaluation procedure tries to find a near-optimal solution to the follower's problem in short time. It performs by iteratively selecting a locally best possible position for opening a facility, until all r follower facilities are placed. A currently best possible location is determined by computing the turnover of the follower for all possible locations using a function similar to the objective function of the leader's problem (4):

$$p^{\mathrm{f}}(y) = \sum_{j \in J} w_j \frac{\sum\limits_{i \in I} v_{ij} y_i}{\sum\limits_{i \in I} v_{ij} x_i + \sum\limits_{i \in I} v_{ij} y_i}, \tag{14}$$

where $y = (y_1, \ldots y_m)$ is the partial solution vector of the follower containing all so far opened facilities and additionally the candidate location. Then, a location with the highest turnover is chosen; ties are broken randomly. The value obtained from this procedure is a lower bound to the follower's problem and therefore $\sum\limits_{j \in J} w_j - p^{\mathrm{f}}(y)$ is an upper bound to the objective value of the leader's solution.

5.2 Initial Population/EA Framework and Variation Operators

The EA's initial population is created by choosing p different facility locations uniformly at random to ensure a high diversity at the beginning. We employ a steady-state genetic algorithm in which exactly one new candidate solution is derived in each iteration. It always replaces the worst individual of the population. Binary tournament selection with replacement is used to choose two candidate solutions for recombination. Offsprings further undergo mutation.

Recombination works as follows. Suppose that we have two candidate solutions $X^1 \subset I$ and $X^2 \subset I$. Then an offspring X' of X^1 and X^2 is derived by adopting all locations from $S = X^1 \cap X^2$ and adding $p - |X^1 \cap X^2|$ further locations from $(X^1 \cup X^2) \setminus S$ chosen uniformly at random.

Mutation is based on the swap neighborhood structure, which is also known from the p-Median problem [22]. A swap move closes a facility and re-opens it at a different, so far unoccupied position. Our mutation applies μ random swap moves, where μ is determined anew at each EA-iteration by a random sample from a Poisson distribution with mean value one.

5.3 Solution Archive

Several methods for duplicate detection in genetic algorithms have been proposed in the literature [18, 15, 14]. In contrast to simple hashing-based approaches, there exist a few works where the archive is not just used to recognize duplicates, but more importantly to also efficiently convert them into similar not yet considered solutions. Such an operation can also be considered as "intelligent mutation". Yuen and Chow [23] present such an approach for continuous optimization problems. For the application to our problem a variation of the trie-based complete solution we proposed in [16] is most suitable. Tests on benchmark problems with binary solution representations, including NK landscapes and Royal Road functions as well as the Generalized Minimum Spanning Tree Problem [9] proved that such an archive is able to boost an EAs performance substantially, especially when the solution evaluation is costly.

A complete solution archive is a data structure that stores all generated candidate solutions in a compact way. An evolutionary algorithm can benefit from such an archive because an on-the-fly conversion of already visited solutions increases diversity in the population, reduces the danger of premature convergence and re-evaluations of already visited solutions are avoided completely. Another rather theoretical property of such an archive-enhanced EA is that in principle it is a complete optimization approach yielding a guaranteed optimal solution in bounded time after considering all solutions of the search space. In practice, however, such an EA usually will be terminated earlier, still yielding only a heuristic solution.

For the underlying data structure we use an indexed trie, which is a tree data structure often applied in dictionary applications [6]. For the performance of a solution archive it is important that inserting, searching and converting a solution can be performed efficiently. A trie is exceptionally good for this purpose because all these operations can be implemented in $\mathcal{O}(m)$ time, where m is the length of the solution representation, i.e., independent of the number of solutions it contains. In general each trie node consists of $|\mathcal{A}|$ pointers to successor nodes, indexed by the elements of $\mathcal{A}$, where $\mathcal{A}$ is the domain of a solution vectors elements, i.e., $\mathcal{A} = \{0, 1\}$ in our case. The maximum height of the trie is determined by the length of the solution vector m.

We combine the EA and the solution archive as follows: Each time a candidate solution is created, we check if this solution is already contained in the archive. In case it is a duplicate it is converted on-the-fly into a not yet considered solution.

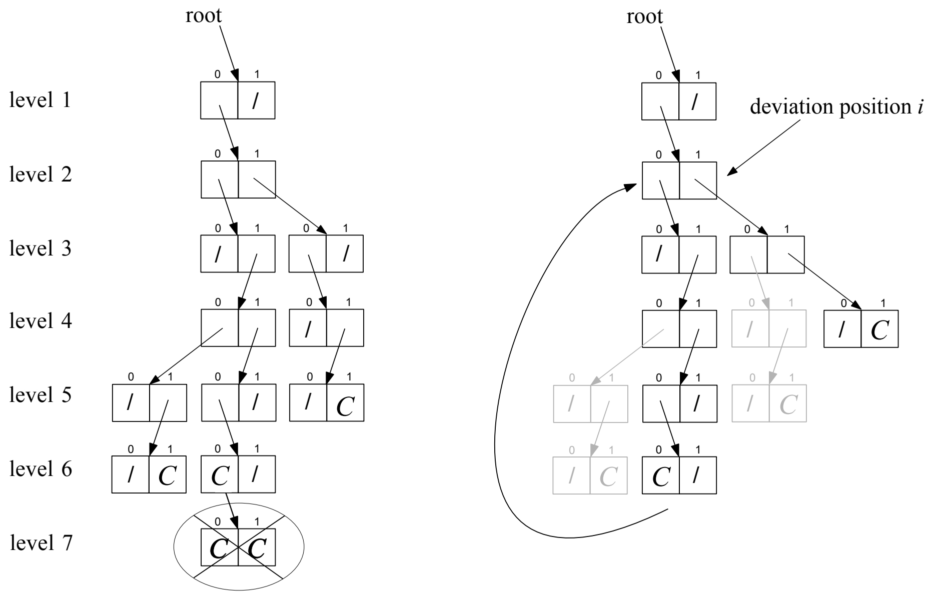

Fig. 1. Solution archive with some inserted solutions on the lefthand side and a sample conversion of $(0, 0, 1, 1, 0, 0, 1)$ into the new solution $(0, 1, 1, 1, 0, 0, 0)$ on the righthand side.

Then the new solution is inserted into the archive and transferred back to the EA, where it is integrated into the population.

Trie Operations
We now describe the problem-specific trie operations which are based on the general methods described in [16]. For inserting a solution into the trie we start at the root node of the trie with the first element x_1 of the solution vector. On each level $i = 1, \ldots, m-1$ of the trie we follow the pointer indexed by x_i. At the lowest level $m-1$, a special constant pointer "C", also called *complete*, is stored to finally represent the solution. Intermediate nodes are always only created when needed and *null*-pointers ("/") indicate empty subtries. Note that such a trie also has a strong relationship to an explicitly stored branch-and-bound tree, as each node divides the search space into two subspaces. Additionally, a subtrie can be pruned if it contains only solutions that have already been visited, i.e., if both of its children are *complete* then this node is deleted and the corresponding entry in the parent node is set to *complete*. On the lefthand side of Figure 1 a sample trie for a small instance with $m = 7$ and $p = 3$ is shown. This trie contains the solutions $(0, 0, 1, 1, 0, 0, 1)$, $(0, 1, 0, 1, 1, 0, 0)$, and $(0, 0, 1, 0, 1, 1, 0)$. The crossed out trie node is pruned by *invalidity*, which is explained in the next paragraph.

Apart from this basic insertion procedure we use some modifications for our type of problem, exploiting the fact that exactly p variables must be set to one in any feasible solution. First, we can stop the insertion procedure already when encountering the p-th one by storing a "C". All remaining elements of the solution must be set to zero. This explains the different depths of the branches in Figure 1. The second adjustment is that we prune the trie by cutting off subtries containing only invalid solutions. Whenever a one is considered for a solution to be inserted, we check if enough facilities would still fit if instead a zero would be chosen. If this is not the case, a corresponding pointer indexed by zero is set to *complete* to indicate that there are no valid solutions in that subtrie. In Figure 1 this is done at the crossed out trie node. These modifications ensure that the trie always is as compact as possible.

The search procedure is similar to the insertion described above because we also start at the root and follow the child nodes corresponding to the solution vector but we are not modifying any trie node. Instead, we conclude that the solution is contained in the trie when we encounter a *complete* pointer and that the solution is new if we reach a *null*-pointer, respectively.

For converting a contained solution $(x_1, \ldots, x_m)$ into a similar but not yet stored one we first choose a position where we will alter the solution. This is done by first determining all feasible deviation positions $i \in I$, for which the corresponding trie nodes at the search path do not contain *complete* for $1 - x_i$. From these possibilities, one deviation position is then selected uniformly at random. Should no feasible deviation position exist anymore, we know that the whole search space has been covered and we can stop the whole optimization with the so far best solution being an optimum. In this case the whole trie has been reduced to a single *complete* pointer. Otherwise, we change the element at the deviation position from one to zero or the other way around which corresponds to closing or opening a facility at location i, respectively. In contrast to previous trie-based solution archives, we have to make another change at a later position for ensuring that p variables are set to one again. There are two possible cases depending on the pointer at the deviation position.

– If it is a *null*-pointer, we know that the corresponding subspace has not been explored yet, which means that any feasible solution from this point on is a new one. Therefore, if we have to close a facility, we choose randomly from the set of open facilities with an index greater than i, set the corresponding variable x_i to zero, and insert the remaining solution as usual into the new trie branch. The case when we have to open a facility is handled analogously.
– If the pointer at the deviation position points towards a successive trie node, we go to this node and consider its pointers. If one of them is *complete*, we have no choice but to follow the other one. Otherwise, we prefer the pointer corresponding to the original solution's variable value, i.e., we follow at level j the pointer indexed by x_j, and repeat the process until we end up in a *null*-pointer. From there we proceed analogously as in the first case and apply the remaining necessary modification(s) randomly to the remaining solution elements. This procedure is guaranteed to terminate with a feasible solution because there must be at least one *null*-pointer in each subtrie.

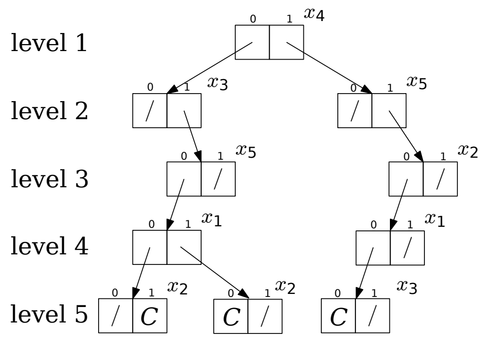

Fig. 2. A randomized trie

On the righthand side of Figure 1 an example of a conversion is illustrated. Suppose that the already existing solution $x = (0, 0, 1, 1, 0, 0, 1)$ shall be converted and inserted. Upon reaching the *complete* pointer, a deviation point is chosen randomly – in this case $i = 2$. Since the alternative entry at $1 - x_2$ points to another trie node, we follow it to the corresponding branch. There we replace the *null* pointer at position one by inserting a new subtrie branch because the element of the original solution $x_3 = 1$. Then we close a random facility with an index greater than 3 – in this case facility at location 7 is chosen – which results in the new solution $(0, 1, 1, 1, 0, 0, 0)$.

Since the conversion procedure can only change solution elements from the deviation position on, it might induce an undesirable bias, i.e., positions with higher indices tend to be changed more often than elements with lower indices. In order to handle this problem, a technique called trie randomization is employed, which was already used in [16] and is described in detail there. Instead of dividing the search space at level $i \in \{1, \ldots, m\}$ according to the value of element x_i, we decide randomly for each trie-node which remaining element is used for this purpose. The elements' index is then stored along with the trie node. Figure 2 shows an example of a randomized trie. Although this technique does not avoid biasing completely, it is substantially reduced.

5.4 Local Improvement

Each new candidate solution derived in the EA via recombination and mutation whose objective value lies within a certain distance from the so far best solution value further undergoes a local improvement step. It is based on a local search applying the swap neighborhood structure already used for mutation. The best improvement step function is used, so all neighbors of a solution that are reachable via one swap move are considered and evaluated and the best one is selected for the next iteration. This procedure terminates when no superior neighbor can be found.

In cooperation with the solution archive this basic local improvement procedure is extended to a tabu search variant where the solution archive acts as tabu list. When enumerating the swap neighborhood of a candidate solution, we check for each neighbor solution if it has already been visited before, i.e., is contained in the solution archive. Only so far unvisited solutions are evaluated and the best one is selected for the next iteration, even if it is worse than the original solution; ties are broken randomly. This process is repeated for α iterations without improving the objective value or until there is no more unvisited neighbor solution. Note that our approach differs from classical tabu search implementations since we do not consider move attributes to be black-listed in a tabu list of limited length but are using the solution archive instead.

6 Computational Results

In this section we present computational results of our approach and compare them to results from the literature. We consider instances from the *Discrete Location Problems* library[1] which are also used by Kochetov et al. [10]. In these instances each customer location corresponds to a possible facility location, i.e., $I = J$. There are 50 such locations and they are chosen randomly on an Euclidean plane of size 100×100. The demand of each customer is randomly drawn from $\{1, \ldots, 10\}$ and the number of facilities to be opened is taken from $\{2, \ldots, 5\}$ for the follower and $\{2, \ldots, 10\}$ for the leader. We further generated larger instances[2] with same properties but 100 locations, i.e., $m = n = 100$. In total we considered 72 test instances.

The EA has a population size of 100 and has been terminated after 3000 iterations without improvement or after 300 seconds. The termination parameter α for the tabu-search-based local search is set to five. Local search/tabu search is called for each candidate solution whose objective value lies within 1% of the best solution found so far. After the EA finishes, the final best solution is evaluated exactly by solving the MIP from Section 4 and using the best greedy solution as starting solution with CPLEX 12.5. All tests are performed on a single core of an Intel Xeon Quadcore with 2.54 GHz.

First we evaluate the impact of the solution archive on the results in Table 1. We compare following algorithms:

- The EA variant where the final best solution is not evaluated with the MIP. This means that the corresponding objective values are not exact, but only approximate values from the greedy evaluation method.
- The Alternating Heuristic (AH) by Kochetov et al. [10].
- The EA variant (EA+MIP) that does not employ the archive and utilizes the basic local search only; the final best solution is evaluated with MIP.
- The EA variant (EA+SA+MIP) that uses the solution archive and the tabu search as local improvement method; the final best solution is evaluated with MIP.

[1] `math.nsc.ru/AP/benchmarks/Design/design_en.html`
[2] `www.ads.tuwien.ac.at/w/Research/Problem_Instances#Competitive_Facility_Location_Problems`

Table 1. Results on small instances with $m = n = 50$ locations. We compare the EA before the exact evaluation (EA), the Alternating Heuristic (AH), the EA with exact evaluation (EA+MIP) and the EA with exact evaluation and solution archive (EA+SA+MIP).

r	p	EA			AH		EA + MIP			EA + SA + MIP		
		$\overline{obj}'$	sd	$t[s]$	obj	$t[s]$	$\overline{obj}$	sd	$t[s]$	$\overline{obj}$	sd	$t[s]$
2	2	127,000	0,00	8	**127,000**	62	**127,000**	0,00	27	**127,000**	0,00	22
2	3	153,000	0,00	10	**153,000**	395	**153,000**	0,00	18	**153,000**	0,00	18
2	4	170,471	0,00	10	**170,471**	3172	**170,471**	0,00	18	**170,471**	0,00	18
2	5	183,338	0,00	15	(182,665)	>36000	**182,665**	0,00	21	**182,665**	0,00	21
3	2	101,000	0,00	12	**101,000**	734	**101,000**	0,00	353	**101,000**	0,00	337
3	3	127,000	0,00	13	**127,000**	246	**127,000**	0,00	103	**127,000**	0,00	101
3	4	145,478	0,07	21	**145,508**	1458	145,478	0,07	54	**145,508**	0,00	49
3	5	159,717	0,00	21	**159,112**	9144	**159,112**	0,00	66	**159,112**	0,00	65
4	2	83,529	0,00	16	**83,529**	6830	**83,529**	0,00	3022	**83,529**	0,00	3018
4	3	108,492	0,00	18	**108,492**	2468	**108,492**	0,00	1265	**108,492**	0,00	1264
4	4	126,962	0,14	29	**127,000**	1004	126,962	0,14	809	**127,000**	0,00	795
4	5	140,850	0,03	33	**140,891**	5490	140,850	0,03	399	**140,891**	0,00	316
5	2	71,177	0,00	19	(71,177)	>36000	**71,177**	0,00	20296	**71,177**	0,00	20388
5	3	95,140	0,00	22	**94,888**	19337	**94,888**	0,00	9621	**94,888**	0,00	9860
5	4	113,092	0,09	34	**113,109**	11060	113,092	0,09	6969	**113,109**	0,00	7022
5	5	126,983	0,04	55	**127,000**	9015	**126,983**	0,04	3020	**127,000**	0,00	2878

In this table we use small instances with 50 locations and customers where r and p are chosen from $\{2, \ldots, 5\}$. Mean objective values of 30 independent runs are given in columns $\overline{obj}$ and corresponding standard deviations in columns sd. Times until termination are listed under $t[s]$ in seconds. For the variant where no MIP is used, $\overline{obj}'$ denotes approximate objective values.

We observe that although the run-time of the EA without archive is in many cases slightly higher, on all instances the EA with archive performs better or as good as the EA without archive. Furthermore, AH produces the same results as our EA with solution archive but requires much more time. The low standard deviations of the EA indicate that our approach is robust at least for small instances. Apart from the short run-times we notice that the objective values obtained by evaluating the best solution found by the EA using the greedy evaluation are very close to those obtained by the exact evaluation. The run-time of all configurations that incorporate the exact evaluation increases steadily with r because of the quickly growing complexity of the MIP. For larger r this evaluation is the dominant part of the algorithm, which takes more than five hours in case of $r = 5$ and $p = 2$, while the actual EA usually terminates within one minute. On this instance and on the instance with $r = 2$ and $p = 5$, AH is not even able to terminate after 10 hours, therefore we show the objective value obtained so far in parentheses. Note that the solution space of instances with $p = 2$ is relatively small, so by using the solution archive we are able to enumerate all possible solutions in the archive.

In order to get a more meaningful comparison between AH and our EA with solution archive, we compare results on the full instance set with up to $n = 100$ locations in Table 2. For these tests, we use a modified AH algorithm (MAH) which solves the follower's problem with the greedy solution evaluation procedure

Table 2. Results on the full set of instances. We compare the modified Alternating Heuristic (MAH) with our EA with solution archive (EA+SA).

n	r	p	MAH		EA + SA			n	r	p	MAH		EA + SA		
			obj	$t[s]$	$\overline{obj}$	sd	$t[s]$				obj	$t[s]$	$\overline{obj}$	sd	$t[s]$
50	2	2	**127,000**	19	**127,000**	0,00	22	100	2	2	277,942	667	**278,736**	0,00	600
50	2	3	**153,000**	9	**153,000**	0,00	18	100	2	3	334,233	535	**337,228**	0,00	625
50	2	4	**170,471**	8	**170,471**	0,00	18	100	2	4	373,665	503	**374,425**	0,00	674
50	2	5	**182,665**	7	**182,665**	0,00	21	100	2	5	399,208	260	**401,781**	0,00	505
50	2	6	**191,771**	7	**191,771**	0,00	23	100	2	6	419,920	275	**421,091**	0,15	586
50	2	7	**198,074**	5	198,073	0,00	63	100	2	7	431,803	272	**436,123**	0,00	440
50	2	8	203,655	5	**204,277**	0,00	80	100	2	8	446,474	158	**448,192**	0,18	440
50	2	9	207,761	5	**208,698**	0,00	190	100	2	9	455,788	166	**458,905**	0,37	529
50	2	10	211,942	4	**212,743**	0,00	305	100	2	10	463,211	173	**467,055**	0,16	416
50	3	2	**101,000**	322	**101,000**	0,00	337	100	3	2	(223,153)	<1	**(223,194)**	0,00	27
50	3	3	**127,000**	87	**127,000**	0,00	101	100	3	3	276,818	5959	**279,000**	0,00	6397
50	3	4	**145,508**	31	**145,508**	0,00	49	100	3	4	319,427	4128	**319,819**	0,00	3956
50	3	5	158,68	49	**159,112**	0,00	65	100	3	5	349,471	3867	**349,793**	0,00	2703
50	3	6	**169,767**	22	**169,767**	0,00	58	100	3	6	372,760	3453	**373,836**	0,12	2777
50	3	7	**178,835**	20	**178,835**	0,00	76	100	3	7	391,314	2086	**391,894**	0,39	2658
50	3	8	**185,516**	14	185,419	0,00	139	100	3	8	407,623	1721	**407,765**	0,08	3148
50	3	9	**191,456**	11	191,371	0,00	150	100	3	9	419,985	1709	**420,305**	0,18	2424
50	3	10	196,442	12	**196,659**	0,00	198	100	3	10	430,465	1299	**431,578**	0,33	2670
50	4	2	**83,529**	2839	**83,529**	0,00	3018	100	4	2	**(183,223)**	<1	**(183,223)**	0,00	38
50	4	3	**108,492**	1163	**108,492**	0,00	1264	100	4	3	(239,527)	<1	**(239,628)**	0,00	83
50	4	4	**127,000**	716	**127,000**	0,00	795	100	4	4	(280,336)	<1	**(280,549)**	0,08	126
50	4	5	**140,891**	256	**140,891**	0,00	316	100	4	5	**(313,041)**	<1	**(313,041)**	0,00	157
50	4	6	152,390	199	**152,660**	0,00	324	100	4	6	(337,158)	<1	**(337,540)**	0,12	242
50	4	7	162,238	138	**162,443**	0,00	255	100	4	7	(356,575)	<1	**(358,233)**	0,18	267
50	4	8	**170,230**	99	**170,230**	0,00	184	100	4	8	(374,436)	<1	**(375,031)**	0,04	300
50	4	9	**176,866**	80	176,735	0,00	165	100	4	9	(387,975)	<1	**(389,837)**	0,12	300
50	4	10	182,363	54	**182,458**	0,00	211	100	4	10	(400,421)	1	**(401,428)**	0,13	300
50	5	2	**71,177**	19288	**71,177**	0,00	20388	100	5	2	**(156,538)**	<1	**(156,538)**	0,00	44
50	5	3	**94,888**	8985	**94,888**	0,00	9860	100	5	3	(207,682)	<1	**(208,025)**	0,00	112
50	5	4	**113,109**	6356	**113,109**	0,00	7022	100	5	4	(244,959)	<1	**(248,663)**	0,06	212
50	5	5	**127,000**	2715	**127,000**	0,00	2880	100	5	5	(279,889)	<1	**(281,522)**	0,00	194
50	5	6	**138,819**	1674	**138,819**	0,00	1875	100	5	6	(305,488)	<1	**(307,129)**	0,13	300
50	5	7	**148,715**	986	147,928	0,00	1884	100	5	7	(327,357)	<1	**(328,314)**	0,05	300
50	5	8	**157,348**	835	**157,348**	0,00	1010	100	5	8	(345,947)	<1	**(346,254)**	0,12	300
50	5	9	**164,347**	612	**164,347**	0,00	800	100	5	9	(360,572)	<1	**(362,159)**	0,31	300
50	5	10	170,215	322	**170,515**	0,00	723	100	5	10	**(374,737)**	1	(374,556)	0,24	300

and only evaluates the final best solution exactly via MIP in the end. This speeds up the algorithm significantly so that it is applicable for larger instances and the run-times become comparable. On small instances with $n = 50$ and $p \leq 5$ we observe that this modification has no negative effects on the results of the algorithm at all, therefore we assume that this is a viable approach. On instances with $n = 100, r \geq 4$, even this simplification is not enough since a single exact solution evaluation using MIP does not terminate within 10 hours. Therefore we rely on approximations again by using the greedy evaluation method for MAH and EA and put the objective values in parentheses. In these cases MAH runs faster than the EA but produces worse results on almost all instances. We also observe that for small n, r and p values the standard deviations of the EA are zero, which confirms that our approach is very robust even for larger instances.

7 Conclusions and Future Work

In this work we developed an evolutionary algorithm for the leader-follower facility location problem with proportional customer behavior incorporating a complete solution archive. We used an incomplete solution representation based on the leader facilities only and described a MIP and a greedy procedure to evaluate a candidate solution. Both of the methods are used in our algorithm. The solution archive is able to significantly improve the results of the otherwise rather simple EA. Furthermore, we observed the alternating heuristic of Kochetov et al. is very time-consuming when the follower's problem is solved exactly. The run-time can be decreased by using the greedy procedure instead which does not have a significant negative impact on the results. However, our EA is able to find solutions that are equally good or even better than those of the Alternating Heuristic for most of the instances.

Here we considered only the variant where customers split their demand proportionally among all facilities. There exist several other variants with respect to customer behaviors in the literature including binary and partially binary choice. It would be interesting to examine the performance of our approach when applied to different customer behavior. Another approach for such discrete competitive facility location problems is to only solve the linear programming (LP) relaxation of the follower's problem, which results in a lower bound on the turnover for a leader solution. When combined with a greedy evaluation, which yields an upper bound to a leader solution, it is possible to omit some exact or LP evaluations if the greedy value is lower than the exact or LP solution value of the best solution found so far.

References

1. Alekseeva, E., Kochetov, Y.: Matheuristics and Exact Methods for the Discrete $(r|p)$-Centroid Problem. In: Talbi, E.G. (ed.) Metaheuristics for Bi-level Optimization, Studies in Computational Intelligence, vol. 482, pp. 189–219. Springer Berlin Heidelberg (2013)
2. Alekseeva, E., Kochetova, N., Kochetov, Y., Plyasunov, A.: A Hybrid Memetic Algorithm for the Competitive P-Median Problem. In: Bakhtadze, N., Dolgui, A. (eds.) Information Control Problems in Manufacturing, vol. 13, pp. 1533–1537. International Federation of Automatic Control (2009)
3. Alekseeva, E., Kochetova, N., Kochetov, Y., Plyasunov, A.: Heuristic and Exact Methods for the Discrete $(r|p)$-Centroid Problem. In: Cowling, P., Merz, P. (eds.) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol. 6022, pp. 11–22. Springer Berlin Heidelberg (2010)
4. Bhadury, J., Eiselt, H., Jaramillo, J.: An alternating heuristic for medianoid and centroid problems in the plane. Computers & Operations Research 30(4), 553–565 (2003)
5. Fernández, J., Hendrix, E.M.: Recent insights in huff-like competitive facility location and design. European Journal of Operational Research 227(3), 581–584 (2013)
6. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York, NY, USA (1997)

7. Hakimi, S.: On locating new facilities in a competitive environment. European Journal of Operational Research 12(1), 29–35 (1983)
8. Hotelling, H.: Stability in competition. The Economic Journal 39(153), 41–57 (1929)
9. Hu, B., Raidl, G.: An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In: Soule, T. (ed.) Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO 2012). pp. 393–400. ACM Press, Philadelphia, PA, USA (2012)
10. Kochetov, Y., Kochetova, N., Plyasunov, A.: A matheuristic for the leader-follower facility location and design problem. In: Lau, H., Van Hentenryck, P., Raidl, G. (eds.) Proceedings of the 10th Metaheuristics International Conference (MIC 2013). pp. 32/1–32/3. Singapore (2013)
11. Kress, D., Pesch, E.: $(r|p)$-centroid problems on networks with vertex and edge demand. Computers & Operations Research 39, 2954–2967 (2012)
12. Küçükaydin, H., Aras, N., Altınel, I.K.: Competitive facility location problem with attractiveness adjustment of the follower: A bilevel programming model and its solution. European Journal of Operational Research 208(3), 206–220 (2011)
13. Laporte, G., Benati, S.: Tabu Search Algorithms for the $(r|X_p)$-medianoid and $(r|p)$-centroid Problems. Location Science 2, 193–204 (1994)
14. Louis, S., Li, G.: Combining robot control strategies using genetic algorithms with memory. In: Angeline, P., Reynolds, R., McDonnell, J., Eberhart, R. (eds.) Evolutionary Programming VI, Lecture Notes in Computer Science, vol. 1213, pp. 431–441. Springer Berlin Heidelberg (1997)
15. Mauldin, M.: Maintaining Diversity in Genetic Search. In: Brachman, R.J. (ed.) Proceedings of the National Conference on Artificial Intelligence (AAAI-84). pp. 247–250. Austin, Texas, USA (1984)
16. Raidl, G., Hu, B.: Enhancing genetic algorithms by a trie-based complete solution archive. In: Cowling, P., Merz, P. (eds.) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol. 6022, pp. 239–251. Springer Berlin Heidelberg (2010)
17. Roboredo, M., Pessoa, A.: A branch-and-cut algorithm for the discrete $(r|p)$-centroid problem. European Journal of Operational Research 224(1), 101–109 (2013)
18. Ronald, S.: Preventing diversity loss in a routing genetic algorithm with hash tagging. Complexity International 2, 548–553 (1995)
19. Saidani, N., Chu, F., Chen, H.: Competitive facility location and design with reactions of competitors already in the market. European Journal of Operational Research 219(1), 9–17 (2012)
20. Sáiz, M.E., Hendrix, E.M., Pelegrín, B.: On nash equilibria of a competitive location-design problem. European Journal of Operational Research 210(3), 588–593 (2011)
21. Suárez-Vega, R., Santos-Peñate, D., Pablo, D.G.: Competitive Multifacility Location on Networks: the $(r|X_p)$-Medianoid Problem. Journal of Regional Science 44(3), 569–588 (2004)
22. Teitz, M.B., Bart, P.: Heuristic methods for estimating the generalized vertex median of a weighted graph. Operations research 16(5), 955–961 (1968)
23. Yuen, S.Y., Chow, C.K.: A non-revisiting genetic algorithm. In: Proceedings of the IEEE Congress on Evolutionary Computation, 2007 (CEC 2007). pp. 4583–4590. IEEE Press, Singapore (2007)