# Optimization algorithms for the price clustering problem

Gabriele Favazzi Mollica[1] and Giovanni Righini[1][0000−0001−9830−7454]

University of Milan, Department of Computer Science
Via Celoria 18, 20133, Milan, Italy
gabriele.favazzimollica@studenti.unimi.it, giovanni.righini@unimi.it

**Abstract.** The price clustering problem requires to group items into clusters so that each item is sold at the price associated with its cluster instead of its original price. A lower bound is imposed on the weighted sum of the total profit obtained by selling the items with the objective of minimizing the difference between the original price and the cluster price for each item. The problem has been investigated and solved with dynamic programming. We present more effective algorithms based on bisection search and iterative improvement allowing to obtain either optimal solutions or $\epsilon$-optimal solutions for any given $\epsilon$.

**Keywords:** Clustering · Bisection search · Dynamic programming.

## 1 The price clustering problem

The price clustering problem is a combinatorial optimization problem that has been recently introduced (see [1]) and it is motivated by the following application. A company that sells a huge number of items with different prices wants to group them into a given number $m$ of clusters, such that all items assigned to the same cluster are sold at the same price. The price assigned to each cluster is a decision variable and it must be chosen so that the expected annual profit, estimated on the basis of the expected demand for each type of item, is not smaller than a given target value. The goal is to define the cluster prices and the assignment of items to clusters so that the price of each item is as close as possible to its original price; therefore, the objective is to minimize the maximum difference (in absolute value) between the price of an item and the price assigned to its cluster.

Compared to facility location problems on a line (see for instance [3]) the price clustering problem is harder because of the constraint on the minimum required annual profit. An attempt to provide exact optimization algorithms has been done with dynamic programming: in [1] mono-directional and bi-directional dynamic programming algorithms were described and compared. They could solve instances with 256 items in less than one second to proven optimality. However, in spite of their very good performance with few hundreds of items, when they are run on instances with thousands items their computing time becomes several orders of magnitude larger and some instances cannot be solved due to memory

overflow, i.e. to the excessive number of states generated. In real settings, the price clustering problem istances may involve several thousands items. For this reason we designed and tested more effective optimization algorithms, namely a bisection search algorithm and an iterative improvement algorithm, which are described in Section 3 and 4, respectively.

### 1.1 Problem definition

To make this paper self-contained, we recall the definition of the price clustering problem given in [1].

*Data.* The following data are given:

- a set $N$ of $n$ items;
- a weight $w_i \in \Re_+$ for each item $i \in N$ (its annual demand);
- a value $p_i \in \Re_+$ for each item $i \in N$ (its original price); without loss of generality, we assume items in $N$ to be numbered by non-decreasing values of price. i.e. $p_i \leq p_j \; \forall i < j$;
- a target value $V$ (the required total annual profit);
- a number $m$ of clusters to be defined; we indicate with $M = \{1, \ldots, m\}$ the indexed set of clusters.

*Variables.* A solution is a partition of $N$ into $m$ clusters $C_1, \ldots, C_m$. For each cluster $C_k$, its *centroid* is a point on the line in position $q_k \in \Re_+$, representing the price assigned to all items in the cluster.

Hence, a solution is represented by the following variables: a binary assignment variable $x_{ik}$ represents the assignment of item $i \in N$ to cluster $k \in M$; a continuous non-negative variable $q_k \in \Re_+$ indicates the price of each cluster $k \in M$.

*Constraints.* The constraint set includes the following assignment constraints:

$$\sum_{k \in M} x_{ik} = 1 \quad \forall i \in N$$

and the constraint on the minimum required annual profit:

$$\sum_{k \in M} \sum_{i \in N} w_i q_k x_{ik} \geq V.$$

*Objective.* The objective is to minimize the *offset*, that is the maximum difference between $p_i$ and $q_k$ for any item $i \in N$ assigned to cluster $k \in M$.

In [1] two basic properties of the price clustering problem were proven: (i) there exists an optimal solution made by non-overlapping clusters; (ii) the minimum offset for each cluster $k$ is given by $q_k - p_k^-$, where $p_k^-$ indicates the minimum

original price of the items in the cluster.

Following [1], we indicate by $W_i^j$ the sum of the weights of the items from $i$ to $j$, i.e. $W_i^j = \sum_{h=i}^{j} w_h$.

Moreover, we define a primary solution correspondinng to a given clustering as the solution that is obtained by setting the price of each cluster $k$ at a value $q_k = p_k^- + z$, where $z$ is set to half the width of the largest cluster, i.e. $z = \max_{k \in M}\{p_k^+ - p_k^-\}/2$. Secondary solutions corresponding to the same clustering are obtained by increasing $z$ by any value $\delta \geq 0$ yielding a corresponding increase in profit equal to $\delta W_1^n$.

## 2 Upper and lower bounding algorithms

The algorithms described in Sections 3 and 4 exploit some upper bounding and lower bounding subroutines, that we describe in this section.

*Heuristic (equitable partition of prices).* The following heuristic provides a set of $m$ non-overlapping clusters, a corresponding (possibly infeasible) primary solution and a feasible (possibly secondary) solution.

Compute the values $\alpha_k = p_1 + k(p_n - p_1)/m$ for each $k \in M$. Define clusters $C_k = \{i \in N \setminus \bigcup_{h=1}^{k-1} C_h : p_i \leq \alpha_k\}$ for each $k \in M$. Define $p_k^- = \min_{i \in C_k}\{p_i\}$ and $p_k^+ = \max_{i \in C_k}\{p_i\}$ for each cluster $k \in M$. Compute the offset $z = \max_{k=1}^{m}\{p_k^+ - p_k^-\}/2$. Compute the price $q_k = p_k^- + z$ for each cluster $k \in M$. Compute the value $v = \sum_{k=1}^{m} q_k \sum_{i=p_k^-}^{p_k^+} w_i$ for the primary solution obtained in this way. If $v \geq V$, the primary solution is feasible and $z$ is a valid upper bound. If $v < V$, consider the secondary solution with profit $v' = V$ and offset $z' = z + (V - v)/W_1^n$. Such a solution is feasible and then $z'$ is a valid upper bound.

*Algorithm DPz.* Algorithm $DPz$ is a dynamic programming algorithm that computes a solution of minimum offset with no restrictions on the profit.

The state in $DPz$ is $(i, k)$ meaning that all items in the range $\{1, \ldots, i\}$ are grouped in $k$ clusters. The set of states is initialized with all partial primary solutions with $k = 1$: $z(i, 1) = (p_i - p_1)/2 \ \forall i \in N$.

All pairs $(i, k) \in N \times M$ are scanned in two nested loops: the outer loop scans the set of items with an index $i$ ranging from 2 to $n$; the inner loop scans the set of clusters with an index $k$ ranging from 2 to $\min\{i, m\}$. When a state $(i, k)$ is considered, the algorithm compares all possible ways of grouping items in $\{1, \ldots, i\}$ in $k$ clusters, such that items in $\{1, \ldots, j\}$ are grouped in $k - 1$ clusters and items $\{j + 1, \ldots, i\}$ in the last cluster; this is done for all values of $j = k - 1, \ldots, i - 1$, to guarantee an implicit enumeration of all possible sets of non-overlapping clusters. When a state $(j, k - 1)$ is extended to generate a state $(i, k)$, the following recursion applies:

$$z(i, k) = \min_{j = k-1, \ldots, i-1} \{\max\{z(j, k - 1), (p_i - p_{j+1})/2\}\}.$$

The optimal value is $z(n, m)$.

The time complexity of $DPz$ is $O(n^2 m)$, i.e. strictly polynomial.

The profit corresponding to the optimal unconstrained solution is not guaranteed to satisfy the profit constraint, that has been neglected. However, the optimal solution of this relaxation provides a valid lower bound; hence we indicate it with $x_{LB}$ and its offset and profit with $z_{LB}$ and $v_{LB}$.

A feasible solution $x_h$, with offset $z_h$ and profit $v_h$, can be obtained from $x_{LB}$ by keeping the clustering unchanged and by increasing the offset until the profit matches the required threshold $V$, so that $v_h = V$ and

$$z_h = z_{LB} + (V - v_{LB})/W_1^n.$$

*Algorithm $DPv$.* $DPv$ is a dynamic programming algorithm that maximizes the profit with a constraint on the maximum allowed value of the offset. Hence, when $DPv(\overline{z})$ is executed, a maximum allowed value for the offset, indicated by $\overline{z}$, is specified in input. As in $DPz$, the state in $DPv(\overline{z})$ is $(i, k)$ and has an associated maximum profit $v(i, k)$ for the given bound $\overline{z}$ on the offset. The set of states is initialized as follows:

$$v(i, 1) = (p_1 + \overline{z})/W_1^i \quad \forall i \in N : p_i - p_1 \leq 2\overline{z}.$$

The order in which the states are extended is the same as in algorithm $DPz$. The extension rule from a state $(j, k-1)$ to a state $(i, k)$ is:

$$v(i, k) = \max_{j=k-1,\ldots,i-1} \{v(j, k-1) + (p_{j+1} + \overline{z})W_1^n\}.$$

The optimal value is $v(n, m)$.

As with $DPz$, also the time complexity of $DPv(\overline{z})$ is $O(n^2 m)$, i.e. strictly polynomial.

For values of $\overline{z}$ smaller than the (unknown) optimal value $z^*$, the solution computed by $DPv(\overline{z})$ is infeasible, i.e. its profit is smaller than $V$. On the other hand, when the solution $x_f$ generated by $DPv(\overline{z})$ satisfies the profit constraint, it is not guaranteed to be primary. Starting from $x_f$, a better feasible solution is obtained by decreasing the offset and the profit until either a primary solution is reached or the profit is equal to $V$. The former case occurs when the value of the offset is decreased up to half the width of the largest cluster in $x_f$. In the latter case, an optimal solution has been found and the search stops. More formally, let $\Delta_f$ be the maximum width of a cluster in $x_f$ and let $z_f$ and $v_f$ be the offset and the profit in $x_f$. If $v_f - (z_f - \Delta_f/2)W_1^n > V$, then a primary solution $x_{UB}$ is found with

$$z_{UB} = \Delta_f/2 \qquad v_{UB} = v_f - (z_f - \Delta_f)W_1^n;$$

otherwise, an optimal solution $x^*$ is found with

$$z^* = z_f - (v_f - V)/W_1^n \qquad v^* = V.$$

Figure 1 shows both cases.

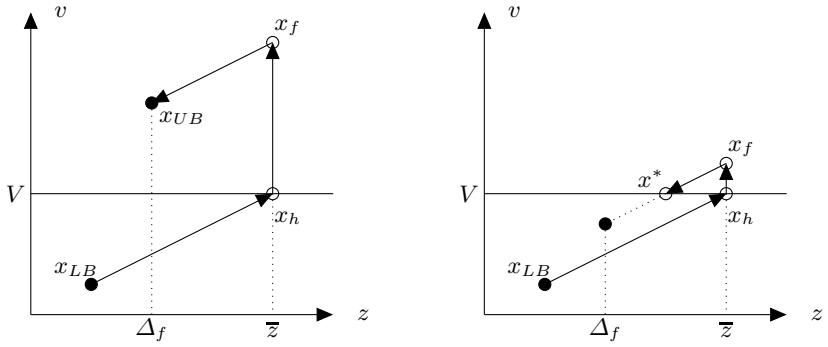


**Fig. 1.** The solutions that are computed in turn: $x_{LB}$ by $DPz$, $x_h$ by making $x_{LB}$ feasible, $x_f$ by $DPv(\overline{z})$ and $x_{UB}$ (primary, on the left) or $x^*$ (optimal, on the right). Primary solutions are indicated by black dots, secondary solutions by white dots.

## 3 Search by bisection

Relying upon $DPz$ and $DPv(\overline{z})$, the search by bisection algorithm $SB$ works as follows. Initially, the optimal solution of the relaxed probelm is computed by $DPz$. If it is feasible (i.e. $v \geq V$), then it is also optimal and the algorithm terminates. Otherwise, it provides a valid lower bound and we indicate it as $x_{LB}$.

Then, in Step 1, $x_h$ is obtained from $x_{LB}$ by keeping the clusters unchanged and increasing the offset and the profit. In Step 2 $x_f$ is computed from $x_h$, through $DPv(z_h)$. In Step 3, starting from $x_f$, a primary solution is searched, by keeping the clusters unchanged, decreasing the offset and the profit. If the search for a primary solution ends up in a feasible solution $x_{UB}$, a valid upper bound is obtained; otherwise, the algorithm terminates and returns $x^*$.

From the lower bound $x_{LB}$ and the upper bound $x_{UB}$, a new value for $\overline{z}$ is computed and $DPv(\overline{z})$ is executed again, iterating the procedure. In particular, in our tests we selected the value

$$\overline{z} = z_{LB} + (z_{UB} - z_{LB})(V - v_{LB})/(v_{UB} - v_{LB}). \tag{1}$$

Some possible iterations of the $BS$ algorithm are shown in Figure 2 and 3.

In the left part of Figure 2 the threshold $\overline{z}^1$ has been computed by (1). The solution computed by $DPv(\overline{z}^1)$ is infeasible (below the threshold $V$); hence it provides a new valid lower bound $x^2_{LB}$ that replaces $x^1_{LB}$. In Step 1, the feasible solution obtained from $x^1_{LB}$ by keeping the clusters unchanged is worse than the current upper bound $z(x^1_{UB})$; therefore it is discarded and Steps 2 and 3 are not done. The next iteration starts with a new lower bound is $x^2_{LB}$, while the upper bound remains at $x^1_{UB}$.

In the right part of Figure 2 the solution computed by $DPv(\overline{z}^1)$ is infeasible, but the feasible solution $x_h$ obtained in Stetp 1 by keeping its clusters unchanged has an offset value $z_h$ better than $x^1_{UB}$. Hence, in Step 2 $DPv(z_h)$ is run and it

provides a feasible solution $x_f$. In Step 3, a primary solution $x^2_{UB}$ is found with the same clusters as $x_f$. In this iteration of $BS$ both the lower bound and the upper bound have been improved from $(x^1_{LB}, x^1_{UB})$ to $(x^2_{LB}, x^2_{UB})$.

In Figure 3 the solution $x_f$ computed by $DPv(\overline{z}^1)$ is feasible (above the threshold $V$); hence, Step 1 and Step 2 arer not done. In Step 3, a primary solution $x^2_{UB}$ is obtained by keeping the clusters of $x_f$ unchanged. The new upper bound is $x^2_{UB}$, while the lower bound remains at $x^1_{LB}$.

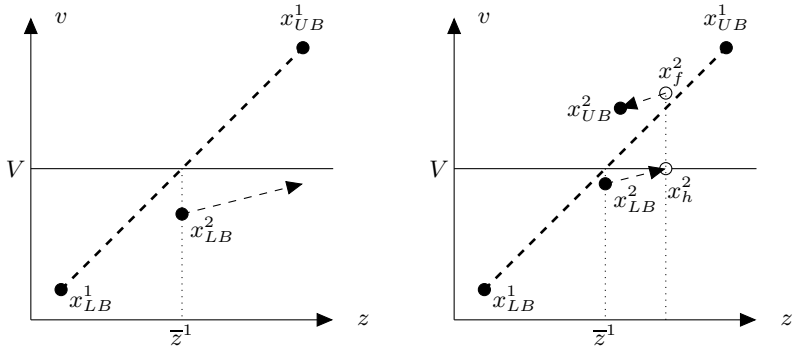


**Fig. 2.** Two possible iterations of the $BS$ algorithm.

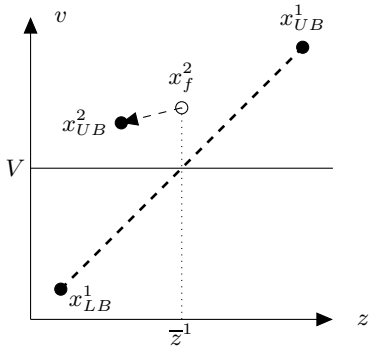


**Fig. 3.** An iteration of the $BS$ algorithm.

The iterations of the $BS$ algorithm terminate when an optimal solution $x^*$ is found in a Step 3 or when the current gap $z_{UB} - zLB$ is smaller than a user-defined threshold $\epsilon$ (we set it to $10^{-6}$ in our tests). In this last case $x_{UB}$ is

returned with the guarantee that it is $\epsilon$-optimal.

In our computational test, we observed that most of the computing time was spent in the execution of $DPz$ to compute an initial lower bound. This suggested the design of an algorithm relying on $DPv$, but not on $DPz$. Such an algorithm is illustrated in the next section.

## 4 Iterative improvement

Instead of starting from an optimal solution for the relaxed problem without the profit constraint, the iterative improvement algorithm $II$ starts from a heuristic solution, say $x'$, for the same relaxation. Solution $x'$ is computed with the same heuristic outlined in Section 2. As with the bisection search algorithm, a feasible solution $x_h$ is computed from $x'$ (Step 1), a maximum profit solution $x_f$ is computed from $x_h$ by $DPv(z_h)$ (Step 2) and a primary feasible solution $x_{UB}$ is computed from $x_f$ (Step 3). If $v_{UB} \leq V$, an optimal solution $x^*$ is found and the search terminates; otherwise, $\overline{z}$ is set to $z_{UB} - \epsilon$ and $DPv(\overline{z})$ is executed again. If its optimal solution $x_f$ is feasible (i.e. $v_f \geq V$), then the upper bound is updated and a new iteration starts; otherwise, the algorithm stops and $x_{UB}$ is an $\epsilon$-optimal solution.

A sketch of some iterations of $II$ is illustrated in Figure 4.

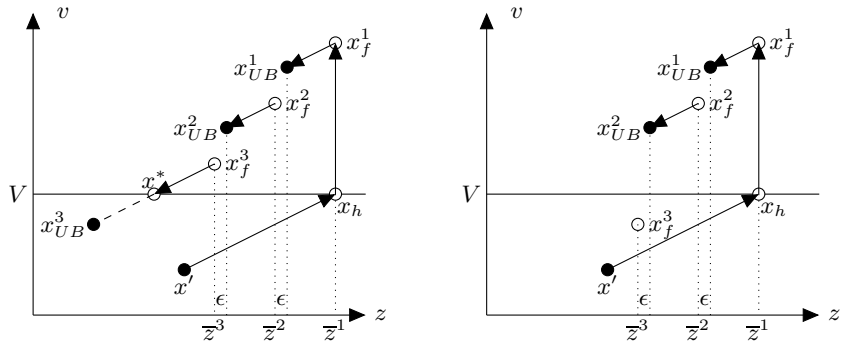


**Fig. 4.** Illustration of three iterations of the $II$ algorithm; the index of each iteration is indicated as an apex. On the left, the search terminates in an optimal secondary solution; on the right the search terminates in an $\epsilon$-optimal primary solution.

## 5 Computational results

*Generation of random instances.* We tested the $BS$ and the $II$ algorithms on instances with $n = 4096$ items (instead of 256 as in [1]). Four types of instances were generated at random, in the same way as in [1].

- Type A: $p$ values are distributed according to a power law: a random number $r$ is generated from a uniform distribution between 1 and $2^{12}$ and a corresponding value of $p$ is defined as $p = r^{3/2}$. The values of $w$ are inversely proportional to $p$: $w_i = T/(np_i)$, rounded to the closest integer, where $T$ is a used-defined parameter (we set it to $10^8$ in our tests).
- Type B: $p$ values are generaed from a uniform distribution between 1 and $2^{18}$ and $w$ values are generated as in type A.
- Type C: the same values of $p$ are used as in type A instances; the values of $w$ are generated from a uniform distribution between 1 and $2T/(n\overline{p})$, where $\overline{p}$ is the mean of the $p$ values.
- Type D: the same $p$ values are used as in type B instances and the $w$ values are generated as in type C instances.

For each type, six instances were solved with $m \in \{\sqrt{n}/2, \sqrt{n}, 2\sqrt{n}\}$ and $\alpha \in \{1.2, 1.5\}$.

Algorithms were coded in FreePascal. All tests were carried out on a laptop equipped with Intel Core i7-8565U 1.99 GHz CPU, 16 GB RAM, Windows 64 bit. Table 1 reports the results.

**Table 1.** Columns 1, 2 and 3: characteristics of the instance; column 4: optimal value; columns 5 and 7: computing time expressed in seconds:hundredths (I/O time is excluded); column 6: number of bisections needed after the computation of the initial upper and lower bounds; column 8: number of iterations from the initial upper bound.

| Type | $m$ | $\alpha$ | $z^*$ | $BS$ | | $II$ | | Type | $m$ | $\alpha$ | $z^*$ | $BS$ | | $II$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | time | it. | time | it. | | | | | time | it. | time | it. |
| A | 32 | 1.2 | 16.03 | 7:07 | 0 | 2:72 | 15 | C | 32 | 1.2 | 96.19 | 9:10 | 0 | 2:29 | 0 |
| | 32 | 1.5 | 16.03 | 7:11 | 0 | 2:68 | 15 | | 32 | 1.5 | 219.55 | 13:52 | 0 | 6:73 | 0 |
| | 64 | 1.2 | 7.95 | 14:19 | 0 | 3:38 | 18 | | 64 | 1.2 | 88.86 | 17:59 | 0 | 4:53 | 0 |
| | 64 | 1.5 | 12.08 | 15:20 | 3 | 17:91 | 78 | | 64 | 1.5 | 212.22 | 26:23 | 0 | 12:17 | 0 |
| | 128 | 1.2 | 4.95 | 37:99 | 6 | 25:77 | 130 | | 128 | 1.2 | 85.28 | 39:95 | 0 | 9:96 | 0 |
| | 128 | 1.5 | 11.59 | 30:49 | 1 | 7:30 | 9 | | 128 | 1.5 | 208.64 | 53:52 | 0 | 26:12 | 0 |
| B | 32 | 1.2 | 56.59 | 7:76 | 3 | 10:90 | 75 | D | 32 | 1.2 | 56.59 | 7:74 | 3 | 10:89 | 75 |
| | 32 | 1.5 | 122.85 | 7:54 | 0 | 1:02 | 0 | | 32 | 1.5 | 122.85 | 7:77 | 0 | 0:97 | 0 |
| | 64 | 1.2 | 50.09 | 14:57 | 0 | 1:34 | 0 | | 64 | 1.2 | 50.09 | 14:55 | 0 | 1:36 | 0 |
| | 64 | 1.5 | 116.68 | 15:28 | 0 | 1:91 | 0 | | 64 | 1.5 | 116.68 | 15:24 | 0 | 1:89 | 0 |
| | 128 | 1.2 | 47.02 | 48:63 | 0 | 2:79 | 0 | | 128 | 1.2 | 47.02 | 28:98 | 0 | 2:87 | 0 |
| | 128 | 1.5 | 113.61 | 30:94 | 0 | 3:96 | 0 | | 128 | 1.5 | 113.61 | 31:02 | 0 | 4:05 | 0 |

*Observations.* The dynamic programming algorithms described in [1] could solve only some of the instances with $n = 4096$ and they took several minutes (up to one hour and a half for some instances). In several cases they ran out of memory, exceeding the available 16 GB RAM. On the contrary, the $BS$ and $II$ algorithms could solve all instances in less than a minute.

As expected, their computing time grows with the number of clusters, because the number of states in the dynamic programming sub-routines linearly depends on $m$.

The $II$ algorithm is often faster but there are exceptions; these occur when the initial upper bound is too far from optimality, forcing the algorithm to explore several different clusterings before finding the optimal one. In turn, this depends on the poor quality of the initial heuristic solution.

Aa a drawback, the final solution provided by $BS$ and $II$ is optimal when it is a secondary solution, obtained setting the profit to $V$ for a given clustering, but it is only $\epsilon$-optimal when it is a primary solution with $v > V$. Hence, care must be taken in setting the value of $\epsilon$.

## 6 Concluding remarks

We have presented two optimization algorithms for the price clustering problem: the $BS$ algorithm is based on bisection search, exploiting the availability of a pair of solutions that provide a lower and an upper bound; the $II$ algorithm is based on iterative improvements and it needs an initial upper bound but not a lower bound. In both cases, dynamic programming sub-routines with strictly polynomial complexity are exploited as sub-routines. Algorithms $BS$ and $II$ allow solving instances with thousands items in less than one minute, providing solutions that can be optimal or $\epsilon$-optimal for any specified $\epsilon$.

Further developments of this research include the study of the problem variation in which the price associated with each cluster is constrained to fall within in the range between the minimum and the maximum price of the items in the cluster (preliminary results can be found in [2]).

## References

1. Favazzi Mollica G., Righini G., Dynamic programming algorithms for a clustering problem on a line, International Conference on Optimization and Learning, Chania, Greece, 2026.
2. Favazzi Mollica G., Algoritmi di ottimizzazione per problemi di clustering monodimensionale vincolati, Bachelor degree thesis, University of Milan, Dept. of Computer Science, 2025.
3. Hsu V.N., Lowe T.J., Tamir A., Structured $p$-facility location problems on the line solvable in polynomial time, Operations Research Letters 21 (1997) 159-164.