A Combinatorial Branch-and-Bound Algorithm for the Capacitated Facility Location Problem under Strict Customer Preferences

Christina Büsing, Felix Engelhardt, Sophia Wrede

Combinatorial Optimization, RWTH Aachen University, Im Süsterfeld 9, 52072 Aachen, Germany

Abstract

This work proposes a combinatorial branch-and-bound (B&B) algorithm for the capacitated facility location problem under strict customer preferences (CFLP-SCP). We use combinatorial insights into the problem structure to do preprocessing, model branching implications, enforce feasibility or prove infeasibility in each node, select variables and derive primal and dual bounds in each node of the B&B tree.

We benchmark this algorithm against the strongest currently known mixed-integer linear programming formulation for the CFLP-SCP. We find that our B&B algorithm finds better primal solutions for more instances and, if slowly, makes progress on solving instances with 1000 customers and 1000 facilities. In comparison, the solver struggles with memory requirements and fails on larger instances. However, we find the linear programming based dual bounds generally outperform our combinatorial bounds, allowing the solver to prove optimality faster for small instances.

Keywords: Branch-and-bound, facility location, capacities, customer preferences, combinatorial optimization

1. Introduction

Our problem setting is motivated by healthcare facility location planning, specifically the planning of preventive care locations. Here, customers can not be assigned to arbitrary facilities, but independently decide on which open facility they visit. We also assume that facilities have capacities for serving customer demands. Finding a set of open facilities that allows for a cost-effective assignment respecting customers' preferences constitutes the *capacitated facility location problem with strict customer preferences*, the CFLP-SCP. We propose a combinatorial branch-and-bound (B&B) algorithm for the CFLP-SCP.

For any (sub-)set of facilities in the CFLP-SCP, assignments are unique and feasibility can be determined in strongly polynomial time (Büsing et al., 2025). Thus, enumeration of possible sets of open facilities sets is a possible if inefficient exact algorithm for the CFLP-SCP. However, we can combinatorically derive primal/dual bounds for the best objective value on arbitrary subsets of facilities without having to solve linear programs (LPs). For optimal subsets of facilities, the bounds correspond to optimal solutions as well. To efficiently enumerate the possible facility subsets we use B&B. Due to our purely combinatorial approach, the evaluation of individual nodes is fast, even for larger instances. We then show how combinatorial arguments can be used to enable the performance of the algorithm, using the concepts of implied demand, domain propagation and strong branching. The resulting B&B algorithm is competitive with a state-of-the-art solver that uses the best integer programming formulations available, outperforming it on larger instances.

As such, this article's contributions are twofold: First, we propose a new state-of-the-art solution method for the CFLP-SCP. Second, we showcase how to combine insights from computational mixed–integer linear programming (MILP) and combinatorial optimisation for computationally efficient optimisation.

Email addresses: buesing@combi.rwth-aachen.de (Christina Büsing), engelhardt@combi.rwth-aachen.de (Felix Engelhardt), wrede@combi.rwth-aachen.de (Sophia Wrede)

In the following, we define the CFLP-SCP in Section 2 and give an overview of relevant facility location literature in Section 3. Then, we introduce the B&B algorithm in Section 4. The combinatorial B&B and the MILP are benchmarked in a computational study in Section 5, and a short conclusion is given in Section 6.

2. Problem definition and notation

In the *capacitated facility location problem with strict customer preferences* (*CFLP-SCP*), we are given a set of customers $i \in I$ and a set of potential facility locations $j \in J$. Serving a customer $i \in I$ incurs a demand $d_i \in \mathbb{Z}_{\geq 0}$, and each facility has a capacity $Q_j \in \mathbb{Z}_{\geq 0}$ for serving customers. Opening facility $j \in J$ incurs cost of $f_j \geq 0$, and assigning customer $i \in I$ to facility $j \in J$ incurs cost of $c_{ij} \geq 0$. In solutions for the traditional facility location problem, customers are viewed as passive participants who will gladly follow their determined assignment. This approach reaches its limits in models where customers have an individual agenda regarding the facilities they want to be served at. To this end, each customer ranks all potential facilities in a strict order, i.e. no customer is indifferent between two facilities. This assumption is often made in the corresponding literature, independently of whether or not capacities are considered (Cánovas et al., 2007; Calvete et al., 2020; Cabezas and García, 2022). In the following, $j <_i k$ indicates that customer $i \in I$ strictly prefers facility $j \in J$ over facility $k \in J$. We formally define the CFLP-SCP next.

Definition 1. *Let $(I, J, (d_i)_{i\in I}, (Q_j)_{j\in J}, (f_j)_{j\in J}, (c_{ij})_{i\in I, j\in J}, (<_i)_{i\in I})$ be an instance of the capacitated facility location problem. The* capacitated facility location problem with strict customer preferences *(CFLP-SCP) consists of finding a subset $F \subseteq J$ of facilities that are opened and an assignment $\Lambda : I \to F$ of customers to these facilities such that*

1. *the capacity limit of each open facility is met, i.e. $\sum_{i\in\Lambda^{-1}(j)} d_i \leq Q_j$ must hold for each $j \in J$,*
2. *each customer is served at their most preferred open facility, i.e. there are no $i \in I$ and $j \in F$ such that $j <_i \Lambda(i)$,*
3. *the cost of opening facilities and assigning customers to their most-preferred open facilities is minimised, i.e. minimize $\sum_{j\in F} f_j + \sum_{i\in I} c_{i\Lambda(i)}$.*

The set $J_{ij}^{<} = \{k \in J : k <_i j\}$ indicates for each customer $i \in I$ and facility $j \in J$ the sets of facilities customer i strictly prefers over j. In the remainder of this paper, we use the following two concepts to encode the impact of assigning a customer to a facility on the assignment of other customers.

Definition 2 (Implied customers (Büsing et al., 2025))**.** *We call a customer $\ell \in I$* implied by customer *$i \in I$* at facility *$j \in J$ if the set of facilities customer ℓ prefers over j is a subset of the set of facilities i prefers over j, i.e. $J_{\ell j}^{<} \subseteq J_{ij}^{<}$. By definition, i implies itself at j. We denote with $\mathcal{I}(i, j) = \{\ell \in I : J_{\ell j}^{<} \subseteq J_{ij}^{<}\}$ the set of all customers who are implied by i at j. Similarly, the set of customers $\mathcal{I}(I', j) \subseteq I$ implied by customer set $I' \subseteq I$ at facility j is the set of customers $\ell \in I$ who are implied by at least one customer $i \in I'$ at j, i.e. $\mathcal{I}(I', j) = \cup_{i\in I'}\mathcal{I}(i, j)$.*

To illustrate the concept of implied customers, consider the example given in Figure 1. Notably, if costumers need to be assigned, this also incurs demand:

Definition 3 (Implied demand (Büsing et al., 2025))**.** *The demand implied by customer $i \in I$ at facility $j \in J$ is the total demand of all customers implied by customer $i \in I$ at facility j, i.e. $\mathcal{D}(i, j) = \sum_{\ell\in\mathcal{I}(i,j)} d_\ell$. Similarly, the demand implied by customer set $I' \subseteq I$ at facility $j \in J$ is the total demand of all customers implied by customer set I' at facility j, i.e.*

$$\mathcal{D}(I', j) = \sum_{\ell\in\mathcal{I}(I',j)} d_\ell. \qquad \text{(Implied Demand)}$$

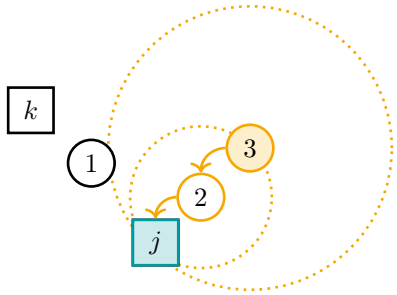


Figure 1: Instance with customers $I = \{1, 2, 3\}$ and facility locations $J = \{j, k\}$. Preferences are defined by distances in the Euclidean space. Assigning customer 3 to j implies the assignment of 2 to j, i.e. $\mathcal{I}(3, j) = \{2, 3\}$.

3. Literature

Facility Location Problems under Preferences. Facility location problems have been thoroughly studied in the literature, see, e.g. Laporte et al. (2019) or Celik Turkoglu and Erol Genevois (2020) for recent surveys. In the basic models, however, customers are viewed as passive participants who will gladly follow their determined assignment. This behaviour model reaches its limits in cases where customers have an individual agenda regarding the facilities they want to be served at, for example, when accessing healthcare facilities (Güneş et al., 2019). In this work, we specifically consider customer preferences.

Uncapacitated Facility Location Problems under Preferences. The seminal work by Hanjoul and Peeters (1987) both first introduced customer preferences in the context of facility location problems, and proposed a first B&B approach for the uncapacitated problem. Their algorithm works via repeatedly solving the problem without customer preferences via linear programming. Branching is then done on facilities with the aim of removing the largest number of violations of the preference orderings. Acknowledging the importance of strong primal bounds for early termination, Hanjoul and Peeters (1987) also proposed two heuristics for solving the uncapacitated facility location problem with customer preferences (UFL-CP). In general, most articles considering customer preferences focus on the uncapacitated case. To solve the UFL-CP problem in practice, preprocessing strategies and custom valid inequalities are effective strategies, see Cánovas et al. (2007) and Vasilyev et al. (2013).

Capacitated Facility Location Problems under Preferences. There are two main approaches to dealing with potentially overloaded facilities in the capacitated case (Calvete et al., 2020). In the first approach, if too many customers want to use a facility, some customers will be served at less preferred facilities; the goal is to globally maximise customer preferences while respecting capacities. In the second approach, facilities leading to a conflict between preferences and capacities have to be closed, and each customer has to be served at their most-preferred open facility; the goal is to minimise total opening and assignment costs.

Most research revolves around the first approach (Casas-Ramírez et al., 2018; Calvete et al., 2020; Polino et al., 2023). The authors consider variations of a bilevel setting where the leader opens facilities with the aim to minimise the total sum of opening and assignment costs; the follower assigns each customer according to a known ranking to maximise the sum of customers' preference rankings. Kang et al. (2023) studied another variation of the CFLP-CP, where customers are allowed not to be served at any of the open facilities. More recently, Domínguez and de Dios Jaime-Alcántara (2025) also proposed mixed-integer linear formulations for assignments that ensure customers have no incentive to deviate.

For the second approach, Büsing et al. (2022, 2025) studied combinatorial structures of the CFLP with customer preferences (CFLP-CP) – results which we make use of in the following. Büsing et al. (2025) introduce preprocessing and cover-based inequalities, which also serve to enable our B&B algorithm. To the best of our knowledge, a problem-specific B&B algorithm has not yet been studied for the CFLP-CP. However, facility location research is a wide field. Thus, we also look at CFLP without preferences.

Capacitated Facility Location Problems without Preferences. Several B&B algorithms have been developed for the capacitated facility location problem (CFLP).

We first consider the non–integer case, where customers are allowed to split their demands among multiple facilities. For that, Nauss (1978) studies a B&B algorithm; Dual bounds are obtained via a Lagrangian relaxation, which is solved via iteratively solving Knapsack problems, primal bounds are obtained by a network flow problem. Beasley (1988) proposes a B&B algorithm for a CFLP with both lower and upper bounds on the demand to be served at each open facility. The dual bound is derived from a Lagrangian heuristic, the upper bound is based on a network flow problem. Görtz and Klose (2012) develop a B&B algorithm that derives both primal and dual bounds from a Lagrangian relaxation of the CFLP, which is solved via subgradient optimization, which is used to compute an approximate solution to the relaxed LP. Similarly to Hanjoul and Peeters (1987), branching is done on facilities in all three references.

If integer decisions are also required for the customer assignment, this subproblem becomes computationally harder. Holmberg et al. (1999) propose a combinatorial B&B algorithm for this case. They derive dual bounds from a Lagrangian heuristic and primal bounds from a heuristic based on a repeated matching algorithm. The algorithm starts by branching on the location variables until all are fixed and, if needed, it continues to branch on allocation variables.

Summary of Insights. Nearly all algorithms from the literature solve a Lagrangian relaxation in order to obtain dual bounds, indicating this is a promising approach. Specifically, Cornuejols et al. (1991) come to the conclusion that, from a computational perspective, it is most promising to consider the Lagrangian subproblem in which the complete assignment constraint of each customer is relaxed.

B&B is being used almost exclusively in the context of integer programming. While several B&B algorithms have been developed for the class of capacitated facility location problems, only one algorithm is designed for the uncapacitated facility location problem with customer preferences. With this, we contribute a first combinatorial B&B algorithm for the CFLP-SCP.

4. Branch and bound algorithm

In this section, we present our combinatorial B&B algorithm for the CFLP-SCP. Figure 2 gives an overview of the main elements of the algorithm.

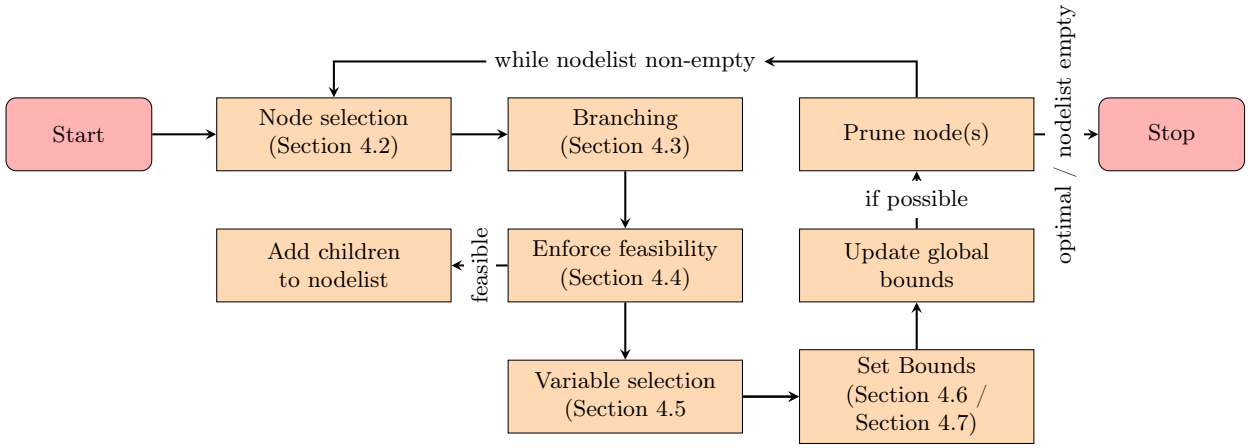


Figure 2: Main elements of the branch-and-bound algorithm. Note that "Add children to nodelist" also implies checking those children for feasibility, setting up bounds, and pruning if necessary. Additionally, preprocessing, see Section 4.1, is performed once at the beginning of the algorithm.

We refer to Morrison et al. (2016) for a general discussion of branch-and-bound algorithms, their constituent parts and the different approaches that are possible for implementing each.

4.1. Preprocessing

For each customer $i \in I$ and active node N, we keep a list L_i of all facilities ordered according to the customer's preferences. Here, the first element $L_i[0]$ is most preferred; in the example in Figure 3 below, this would be j_4. Clearly, if k is a lower bound on the number of facilities that have to be opened, a customer will always get one of their $|J| - k + 1$ most preferred facilities. In the example in Figure 3, for $k = 4$, we delete the last three facilities for every customer. In our algorithm, we obtain the minimum number of facilities needed by computing the Implied Demand of each customer at each facility, which can be done in polynomial time, see Algorithm 2. Then, if the k-least preferred facilities are overloaded, we can delete them from the preference lists.

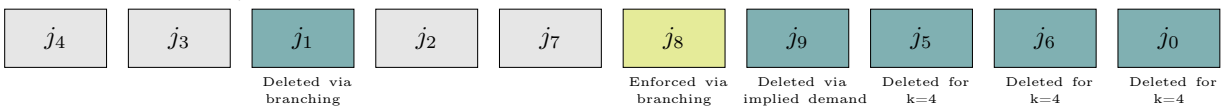


Figure 3: Visualization of a preference list for a customer i including facilities deleted/enforced due to branching, as well as unfavoured facilities deleted since they are never part of a feasible solution.

Note that in the example above, if j_2 were also overloaded, we would note this for the following sections, but not delete j_2 from the preflist as this would make it feasible to open j_2 and assign customer i to a less preferred facility, i.e. j_7 or j_8. However, for the k-least preferred facilities this is not an issue because there are no less-preferred facilities a customer might be assigned to.

4.2. Node selection

For each step, we select an open node with lowest dual bound. This strategy is referred to as *best first search* and focuses on improving the dual bound as quickly as possible (Achterberg, 2007).

4.3. Modelling branching implications

When generating a new node N', we inherit its parent node's information on the least preferred potential facility of each customer and create the temporary preference list subject to the restrictions incurred during branching. If a facility $j \in J$ is deleted during branching, this facility is removed from the respective temporary preference lists of all customers $i \in I$ in N', see j_1. If a facility $j \in J$ is enforced during branching, for each customer, facilities less preferred than j will never be part of a feasible solution assignment. Thus, we delete all elements after $L_i[arg(j)]$, in the example above that would apply to j_9, j_5, j_6, j_0 if those had not already been removed.

4.4. Enforcing feasibility

For any new node, we first determine whether it is feasible. For that, we use a modified version of the greedy algorithm from Büsing et al. (2025). Pseudocode thereof is given in Algorithm 1 in Appendix B.

Basic algorithm. First, open all facilities that are still allowed to be opened, then close overloaded facilities and reassign their demand until termination. To compute this efficiently, we track assignments, i.e. for any facility $j \in J$ we have a set of customers $A_j \subseteq I$ that are currently assigned to j and a residual capacity Q_j^{res} of j. Note that the algorithm is constructive: It either returns a subset of facilities J^* where each customer can be served at their most preferred facility, or it proves that no such subset exists. For the root node, the remaining subset of facilities must contain the optimal set of open facilities as a subset, if any exists.

Modification. We can combine the concept of implied demands with the greedy algorithm from Büsing et al. (2025). For that, consider a customer i that is served at $L_i[k]$. In this case, all more preferred facilities $J^{<}_{iL_i[k]}$ have to be closed; we then compute a feasible subset J^* on the set of facilities $J \setminus J^{<}_{iL_i[k]}$ by applying Algorithm 1. If J^* does not exist, delete all L_i values starting from $L_i[k]$, since further deletion of facilities will not lead to more feasible solutions. If J^* exists but $L_i[k]$ is closed, set the implied demand of i at $L_i[k]$ to $\mathcal{D}(i, L_i[k]) = Q_{L_i[k]} + 1$, which indicates that this assignment is infeasible. Otherwise, determine the implied demand of i at $L_i[k]$ based on the facilities in J^*. See Algorithm 2 in Section Appendix B for the corresponding pseudocode. In our code, we track customers that cannot be served at a facility as well as the least preferred potential facility for each customer. Notably, combining branching with implied demands gives us progressively stronger results:

Proposition 1. *The implied demand of a customer $i \in I$ at a location $j \in J$ monotonically increases as the number of potential facility locations decreases.*

Proof. We prove our claim by showing that

$$\mathcal{I}(i,j)_X = \{i\} \cup \underbrace{\{\ell \in I : X^{\leq}_{\ell j} \subseteq X^{<}_{ij} \cup \{j\}\}}_{\substack{\text{Customers implied by } i \text{ at } j \\ \text{if } X \subseteq J \text{ corresponds to the} \\ \text{set of open facilities}}} \subseteq \{i\} \cup \underbrace{\{\ell \in I : Y^{\leq}_{\ell j} \subseteq Y^{<}_{ij} \cup \{j\}\}}_{\substack{\text{Customers implied by } i \text{ at } j \\ \text{if } Y \subseteq X \text{ corresponds to the} \\ \text{set of open facilities}}} = \mathcal{I}(i,j)_Y$$

holds for all $i \in I$ and any two sets of potential facility locations $Y, X \subseteq J$ with $j \in Y \subset X$. For all customers $\ell \in \mathcal{I}(i,j)_X$, relation $X^{\leq}_{\ell j} = \{k \in X : k \leq_\ell j\} \subseteq X^{<}_{ij} \cup \{j\} = \{k \in X : k <_\ell j\} \cup \{j\}$ holds by definition of implied customers. Then, relation $X^{\leq}_{\ell j} \cap Y \subseteq (X^{<}_{ij} \cup \{j\}) \cap Y$ follows immediately. Since $Y \subset X$, we immediately conclude that $Y^{\leq}_{\ell j} = \{k \in Y : k \leq_\ell j\} = X^{\leq}_{\ell j} \cap Y \subseteq (X^{<}_{ij} \cup \{j\}) \cap Y = \{k \in Y : k <_\ell j\} \cup \{j\} = Y^{<}_{\ell j} \cup \{j\}$ holds. Thus, any customer in $\mathcal{I}(i,j)_X$ is a customer in $\mathcal{I}(i,j)_Y$, and our claim holds. □

In our algorithm, we update the implied demand whenever we delete a facility via branching. However, creating a modified preference list for each considered node is computationally too expensive. In our algorithm, we therefore create one preference list in the rootnode and only note the location of the least preferred potential facility for each customer afterwards. Based on this, we can then initialise a temporary preference list at each node.

4.5. Variable selection

The quality of our dual bounds is closely linked to the size of the remaining search space: If $\sum_{i \in I} |L_i| = |I|$, i.e. each customer has only one facility left in their preference list, there remains at most one solution in the corresponding subtree. Therefore, we use the size of the remaining solution space, i.e. $\sum_{i \in I} |L_i|$, as a pseudocost for selecting branching variables.

Gamrath proposes combining (strong) branching with *domain propagation*, i.e. fixing more variables based on the decision made during branching (Gamrath, 2014). We apply the same principle: First, we check for each branchable facility $j \in J \setminus J^+$ whether closing of j would lead to infeasibility using Algorithm 1. If yes, we immediately add j to the set of open facilities J^+. Otherwise, compute a feasible subset $J^* \subseteq J \setminus \{j\}$ and set

$$score(j_-) = \sum_{i \in I} |L_i \cap J^*|.$$

For the positive fixation, we cut off less preferred facilities, as explained in Subsection 4.3 and then analogously compute $score(j_+)$ on the modified preference lists. We pick a facility that minimises the sum of $score(j_+) + score(j_-)$ as a branching candidate.

4.6. Primal bounds

Primal heuristics are an important factor in designing efficient B&B algorithms, a fact already pointed out by Hanjoul and Peeters (1987) that is well-established in computational constraint and mixed-integer programming (Berthold, 2006, 2013). Thus, we consider multiple problem-specific heuristics.

Simple greedy. Let J' be a feasible (sub-)set of facilities in J which we derive by applying the greedy algorithm in Appendix B. By construction, the capacity limit of each facility $j \in J'$ is met if each customer is served at their most preferred open facility. The sum over the opening cost of all facilities in J' together with the total assignment cost of all customers to their serving facility in J' provides a trivial upper bound.

However, some of the facilities in J' might not serve any customer at all. Without loss of generality, we close any facility $j \in J'$ which serves no customer. This bound is equal to the optimal objective value if we arrive at a node in the branching tree where we branch open all facilities in J^* and branch close all facilities in $J \setminus J^*$ with $J^* \subseteq J$ the set of all open facilities in an optimal solution. Notably, facilities which do not serve any customer in the current solution may still be part of an optimal solution.

Outside of optimal subsets, however, the greedy bound overestimates the objective value, especially in the beginning where a lot of facilities are open while only serving few customers. Thus, we iteratively try closing a random one of the k most promising facilities based on the estimated cost reduction, i.e. for a facility j the difference between assignment costs to the second-most preferred facility for all customers currently assigned to j, as well as the fixed cost for opening j. After closing and before evaluating, if necessary, we reapply Algorithm 1.

Greedy min-cost flow. We can improve on the simple greedy principle as follows. First, define a set of open facilities $J^+ \subseteq J'$ that have to be open due to branching decisions. In the example in Figure 3, this could, e.g. be $J^+ = \{j_8\}$, if no other facility has been branched on so far. Then, we solve a minimum cost flow problem in which each customer and each facility correspond to a node. There is an arc from a customer node i to a facility node j if (a) customer i likes j at least as much as their most preferred facility in J^+ and (b) facility j can serve the implied demand of i at j. In the example, for i this would be j_4, j_3, j_2 and j_8.

Using arc (i, j) incurs cost of 0 if i prefers j most and c_{ij}/d_i otherwise. We introduce a node s that is connected to each customer node, each facility node is connected to a node t. Arc (s, i) has capacity d_i and incurs cost of zero. Arc (j, t) has capacity Q_j and incurs cost 0 if facility j is opened via branching and, otherwise, of $(f_j + \sum_{k \in I: L_k[0]=j} c_{kj})/Q_j$. The graph is illustrated in Figure 4. By definition of the arc costs, the network encourages sending flow from a customer node i to a facility node j if i prefers j most and, if j is branched open, sending flow from node j to t.

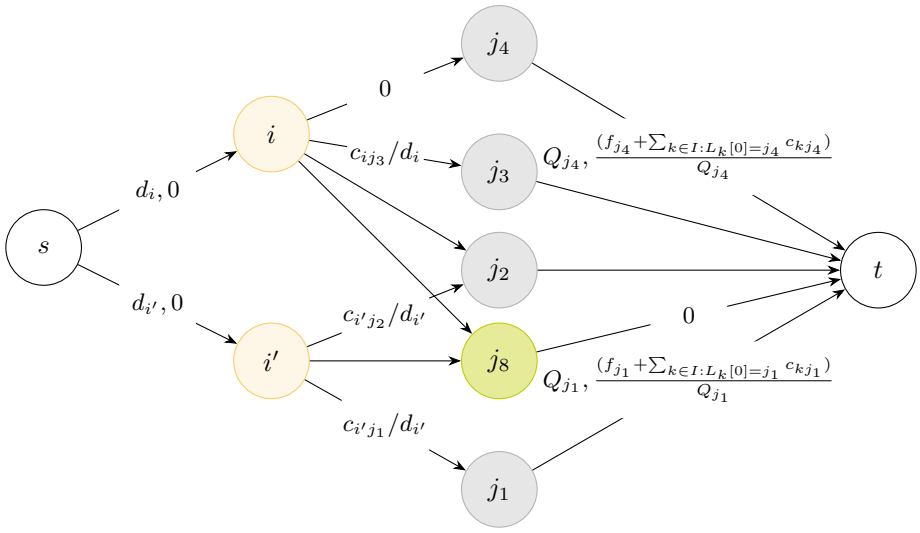


Figure 4: Minimum cost flow network with open facilities and customer preferences corresponding to the preference list example 3. Not all labels are given, the given ones serve to illustrate the min-cost-flow graph structure. Exemplary arc labels contain capacity, and cost per unit of flow.

We define the set of open facilities $F' = J^+ \cup \{j \in J' : f(j, t) > 0\}$ consisting of J^+ and all facilities with positive flow $f(j, t)$ on arc (j, t). If F' does not induce a feasible solution, we add the facility with

lowest value $(f_j + \sum_{k \in I: L_k[0]=j} c_{kj})/Q_j$ to F', and we solve the modified minimum cost flow instance in which facilities in F' are treated as facilities that are opened via branching. Continue this procedure until the constructed solution is feasible. In the worst case, we have to open all facilities in the set J', which gives us the trivial upper bound.

If F' induces a feasible solution, we check whether the objective value can be improved via another greedy algorithm. For that, we first sort the facilities in F' in monotonically increasing order according to the ratio between left-over capacity and fixed cost from $F' \setminus J^+$, i.e. $(Q_j - \sum_{k \in I:(L_k \cap F')[0]=j} d_k)/(f_j + \sum_{k \in I:(L_k \cap F')[0]=j} c_{kj})$ for $j \in F' \setminus J^+$. In this ordering, either very expensive facilities or facilities with a lot left-over capacity are at the beginning. For each facility in the ordering, we try to close it. If closing it leads to an infeasible solution, we try to regain feasibility by applying Algorithm 1, and, in case of success, we close the considered facility and consider the next facility in the ordering. We repeat the whole process until it is no longer possible to close further facilities. Lastly, we remove all facilities from $F' \setminus J^+$ that serve no customer.

4.7. Dual bounds

In order to compute dual bounds at a node in the branching tree, we determine the minimum cost for opening facilities and assigning customers. A simple lower bound consists of calculating both independently. We strengthen this bound, provide an alternative lower bound based on minimum cost flows and show how to combine both for an even stronger lower bound. Finally, we discuss a Lagrangian relaxation which provides another lower bound.

Simple lower bound. For a given CFLP-SCP instance, the opening costs must be at least

$$\min_{J' \subseteq J} \left\{ \sum_{j \in J'} f_j : \sum_{j \in J'} Q_j \geq \sum_{i \in I} d_i \right\}$$

and the assignment costs must be at least

$$\sum_{i \in I} \min_{j \in L_i} c_{ij}.$$

Furthermore, facilities opened due to the branching decisions, i.e. facilities in the set $J^+ \subseteq J$, are open and their opening cost contributes to the lower bound while their capacity contributes to the provided capacity. We may also use $j \in J^+$ to update (shorten) the preference lists L_i of customers. Then, a lower bound on the objective is given by

$$\min_{J' \subseteq J \setminus J^+} \left\{ \sum_{j \in J' \cup J^+} f_j : \sum_{j \in J' \cup J^+} Q_j \geq \sum_{i \in I} d_i \right\} + \sum_{i \in I} \min_{j \in L_i} c_{ij}.$$

Computationally, the only challenging thing about the bound above is the cover implied in the first optimisation problem. However, we can solve its linear relaxation, i.e. drop integrality on the choice of J' to approximate the lower bound.

In practice, this bound is too weak. Improving it requires linking both assignment of customers and opening of facilities. We use the following insight: If a facility j is open, customers who have j as a first priority will always use it. Thus, opening facility j will incur fixed assignment costs for those customers. These will be at least as large as the minimal assignment costs, therefore we only need to count the cost increase. This results in the stronger bound given by Proposition 2.

Proposition 2. *For a given CFLP-SCP instance with fixed open facilities J^+ and corresponding preference lists L_i, a lower bound on the objective is given by*

$$\min_{J' \subseteq J \setminus J^+} \left\{ \sum_{j \in J' \cup J^+} \left(f_j + \sum_{i \in I: L_i[0]=j} (c_{ij} - \min_{j \in L_i} c_{ij}) \right) : \sum_{j \in J' \cup J^+} Q_j \geq \sum_{i \in I} d_i \right\} + \sum_{i \in I} \min_{j \in L_i} c_{ij}.$$

This bound may still underestimate total cost by an arbitrary amount. Consider an example with one small, unpopular facility j^* that has low assignment costs $c_{ij} = 0$ for all customers i. Clearly, assigning all customers to j^* is not feasible, but since j^* exists, the lower bound on assignment costs will always be zero.

Flow-based lower bound. We can compute an alternative lower bound that avoids the above-mentioned caveat. To do so, we prove that the cost of a fractional minimum cost flow in a network as described in Figure 4 translates into a lower bound.

Proposition 3. *Denote with $J^+ \subseteq J$ the set of facilities that have to be open due to branching decisions. Consider a flow network as described in Figure 4. Denote the value of the corresponding minimum cost flow with OPT. Then, $OPT + \sum_{j \in J^+}(f_j + \sum_{i \in I: L_i[0]=j} c_{ij})$ is a lower bound on the CFLP-SCP with open facilities at J^+.*

Proof. Let $J^+ \subseteq J$ be the set of facilities that have to be open due to branching decisions. Then, the CFLP-SCP can be formulated as an ILP. A state-of-the-art ILP formulation for the CFLP-SCP is given in Appendix A. In this context, Cánovas et al. (2007) observe that $x_{ij} = y_j$ holds for all $i \in I, j \in J$ with $L_i[0] = j$, where x are assignment and y are opening variables. Hence, we can rewrite objective function (A.1a) as

$$\begin{aligned}
&\sum_{j \in J} f_j y_j + \sum_{j \in J} \sum_{i \in I} c_{ij} x_{ij} \\
&= \sum_{j \in J^+} (f_j + \sum_{i \in I: L_i[0]=j} c_{ij}) + \sum_{j \in J \setminus J^+} (f_j + \sum_{i \in I: L_i[0]=j} c_{ij}) y_j + \sum_{j \in J} \sum_{i \in I: L_i[0] \neq j} c_{ij} x_{ij}.
\end{aligned}$$

Introducing new cost functions

$$\bar{c_{ij}} = \begin{cases} c_{ij} & \text{if } L_i[0] \neq j \\ 0 & \text{othwerwise} \end{cases} \qquad \text{and} \qquad \bar{f_j} = \begin{cases} f_j + \sum_{i \in I: L_i[0]=j} c_{ij} & \text{if } j \in J \setminus J^+ \\ 0 & \text{othwerwise} \end{cases}$$

allows us to rewrite the objective function as

$$\underbrace{\sum_{j \in J^+} (f_j + \sum_{i \in I: L_i[0]=j} c_{ij})}_{\text{Enforced open facilities}} + \underbrace{\sum_{j \in J} \bar{f_j} y_j}_{\text{Opening}} + \underbrace{\sum_{j \in J} \sum_{i \in I} \bar{c_{ij}} x_{ij}}_{\text{Assignment}}.$$

It is easy to see that linear programming (LP) formulation

$$\begin{aligned}
\min \quad & \sum_{j \in J} \bar{f_j} y_j + \sum_{j \in J} \sum_{i \in I} \bar{c_{ij}} x_{ij} && && \text{(1a)} \\
\text{s.t.} \quad & \text{(A.1b), (A.1c), (A.1e)} && && \text{(1b)} \\
& x_{ij}, y_j \in [0, 1] && \forall i \in I,\ j \in J && \text{(1c)}
\end{aligned}$$

contains (A.1), and, therefore, provides a lower bound on the objective value of (A.1). Moreover, formulation (1) corresponds to a simple min-cost flow problem which differs from the instance considered in Figure 4 by arcs (i, j) from customer nodes $i \in I$ to facility nodes $j \in J$ if (a) customer i likes j less than their most preferred facility in J^+ or (b) facility j cannot serve the implied demand of i at j. However, since such assignments do not occur in any feasible solution of the CFLP-SCP, we may remove any arc (i, j) violating (a) or (b) from the network and still have a lower bound. This modified network now corresponds to the network described in Proposition 3. □

Note that this lower bound corresponds to the optimal value after branching all facilities in an optimal solution open and all other facilities closed. By construction, a feasible flow assigns each customer fractionally to facilities, and fractionally opens facilities in $J \setminus J^+$ while opening facilities in J^+ completely. In comparison to the previous bound, this one does not overuse cheap facilities, but mostly ignores customer preferences.

We derive the following relation between the bounds introduced in Propositions 2 and 3.

Proposition 4. *The flow-based bound from Proposition 3 dominates the bound from Proposition 2.*

4.7.1. Lagrangian relaxation

The Lagrangian relaxation of a MILP formulation of a problem poses as another, performant approach to generate dual bounds in a B&B algorithm (Geoffrion and Bride, 1978; Holmberg et al., 1999). Here, complicated constraints are replaced with a penalty term in the objective function (Fisher, 1985). In the following, we first derive a Lagrangian relaxation of formulation (A.1) by relaxing constraints (A.1b), (A.1c) and (A.1d). Note that it is possible to keep constraints (A.1c) as hard constraints in the relaxation; however, similarly to Beasley (1993), we observed better computational results when relaxing them.

Relaxation of complete assignment and preference constraints. We rewrite constraints (A.1c) as $\sum_{i\in I}(d_i/Q_j)x_{ij} \leq y_j$ for each $j \in J$ before relaxing them. Furthermore, we observed a positive impact on the computational performance by rewriting constraints (A.1d) to equivalent constraints $y_j \leq \sum_{k\in J_{ij}^{\leq}} x_{ik}$ for $j \in J, i \in I$ (Espejo et al., 2012) before relaxing constraints (A.1b) and (A.1d). Then, we can consider the objective function $(L_{\lambda,\mu,\rho})$, which is given by

$$(L_{\lambda,\mu,\rho}) = \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \overbrace{\sum_{i\in I}\lambda_i(1-\sum_{j\in J}x_{ij})}^{\text{Relaxation of (A.1b)}} + \overbrace{\sum_{j\in J}\rho_j(\sum_{i\in I}(d_i/Q_j)x_{ij} - y_j)}^{\text{Relaxation of (A.1c)}} + \underbrace{\sum_{i\in I}\sum_{j\in J}\mu_{ij}(y_j - \sum_{k\in J_{ij}^{\leq}} x_{ik})}_{\text{Relaxation of (A.1d)}}$$

with Lagrangian multipliers $\lambda_i \in \mathbb{R}$, $\mu_{ij}, \rho_j \geq 0$ for $i \in I, j \in J$. By reordering the elements, we derive function

$$(L_{\lambda,\mu,\rho}) = \sum_{j\in J}(f_j + \sum_{i\in I}\mu_{ij} - \rho_j)y_j + \sum_{i\in I}\sum_{j\in J}\big[\big(c_{ij} - \lambda_i + \rho_j(d_i/Q_j)\big)x_{ij} - \mu_{ij}\sum_{k\in J_{ij}^{\leq}} x_{ik}\big] + \sum_{i\in I}\lambda_i.$$

The optimisation over the assignment variables x_{ij} can be carried out independently (Geoffrion and Bride, 1978; Nauss, 1978). If $y_j = 0$ for a $j \in J$, then $x_{ij} = 0$ follows immediately for all $i \in I$. If $y_j = 1$, then $x_{ij} = 0$ for $i \in I$ with $j \notin L_i$, i.e. customers with j not occurring in their preference list must not be served at j. Furthermore, we set $x_{ij} = 1$ if i prefers j most. In this case, there is no need to relax (A.1d). Then, the values of assignment variables x_{ij} correspond to an optimal solution to

$$\min \quad (L^j_{\lambda,\mu,\rho}) := (f_j - \rho_j + \sum_{\substack{i\in I:\\ j\in L_i}}\mu_{ij}) + \sum_{\substack{i\in I:\\ j\in L_i\setminus J^+}}(c_{ij} - \lambda_i + \rho_j(d_i/Q_j) - \sum_{k\in J_{ij}^{\geq}\cap L_i}\mu_{ik})x_{ij} \tag{2a}$$

$$\text{s.t.} \quad x_{ij} \in \{0,1\} \qquad \forall i \in I : j \in L_i \setminus J^+. \tag{2b}$$

We can solve this instance in polynomial time by defining

$$x_{ij} = \begin{cases} 1 & \text{if } c_{ij} - \lambda_i + \rho_j(d_i/Q_j) - \sum_{k\in J_{ij}^{\geq}}\mu_{ik} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

for all undecided assignment variables x_{ij}. We denote a solution of formulation (2) with $v(L^j_{\lambda,\mu})$. Following Nauss (1978), we can strengthen the Lagrangian relaxation by adding the constraint $\sum_{j\in J} Q_j y_j \geq \sum_{i\in I} d_i$ which ensures that the overall capacity of opened facilities exceeds the total demands of all customers. We can further strengthen the formulation by adding constraint $\sum_{j\in J} y_j \geq k$ with $k \in \mathbb{Z}_{\geq 0}$ a lower bound on the number of facilities that have to be opened; cf. Section 4.1. Finally, we determine the values for y_j by solving

$$\min \quad \sum_{j\in J} v(L^j_{\lambda,\mu,\rho})y_j + \sum_{\substack{j\in J^+, i\in I:\\ L_i[0]=j}}(c_{ij} - \lambda_i + \rho_j\frac{d_i}{Q_j} - \mu_{ij}) + \sum_{i\in I}\lambda_i \tag{3a}$$

$$\text{s.t.} \quad \sum_{k \in J} x_{ik} = x_{ij} = 1 \qquad \forall i \in I : L_i[0] = j \in J^+ \qquad (3b)$$

$$y_j + \sum_{k \in J^{>}_{ij}} x_{ij} \leq 1 \qquad \forall i \in I, j \in J^{>}_{iL_i[-1]} \qquad (3c)$$

$$\sum_{j \in J} Q_j y_j \geq \sum_{i \in I} d_i \qquad (3d)$$

$$\sum_{j \in J} y_j \geq k \qquad (3e)$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I : j \notin L_i \qquad (3f)$$

$$y_j = 1 \qquad \forall j \in J^+ \qquad (3g)$$

$$y_j \in \{0,1\} \qquad \forall j \in J. \qquad (3h)$$

We denote a solution for this problem with $v(L_{\lambda,\mu,\rho})$. It is evident that $y_j = 1$ for all $j \in J$ with $v(L^j_{\lambda,\mu,\rho}) \leq 0$. In our B&B, we combinatorially solve the LP-relaxation of formulation (3) once without Constraints (3d) and once without Constraints (3e), and consider the solution with best objective value. We solve each of these two problems by applying a simple greedy heuristic for the continuous covering problem.

Lagrangian multipliers. The performance of the Lagrangian relaxation depends on the choice of multipliers $\lambda_i, \mu_{ij}, \rho_j \geq 0$ for $i \in I, j \in J$. Set $t = 0$. In the root node, we start with multipliers $\lambda_i^t = \max_{j \in L_i}\{c_{ij}\}$ and $\mu_{ij}^t = \rho_j^t = 0$ for $i \in I, j \in J$. Then, we solve formulation (3) for these fixed multipliers and determine based on solution $(\mathbf{x}^t, \mathbf{y}^t)$ subgradients

$$L_i^t = 1 - \sum_{j \in J} x_{ij}^t, \qquad M_{ij}^t = y_j^t - \sum_{k \in J^{\leq}_{ij}} x_{ik}^t, \qquad R_j^t = \sum_{i \in I} (d_i/Q_j) x_{ij}^t - y_j^t$$

for $i \in I, j \in J$. Following Beasley (1988, 1993), we observe a positive computational impact by setting $L_i^t = 0$, $M_{ij}^t = 0$ and $R_j^t = 0$ if $\lambda_i^t = 0$ and $L_i^t < 0$, $\mu_{ij}^t = 0$ and $M_{ij}^t < 0$, and $\rho_j^t = 0$ and $R_j^t < 0$, respectively. Given these subgradients, we update the multipliers to

$$\lambda_i^{t+1} = \lambda_i^t + \alpha^t \cdot L_i^t, \qquad \mu_{ij}^{t+1} = \max\{0, \mu_{ij}^t + \alpha^t \cdot M_{ij}^t\}, \qquad \rho_j^{t+1} = \max\{0, \rho_j^t + \alpha^t \cdot R_j^t\}$$

with the so-called *step size*

$$\alpha^t = \frac{2 \cdot (1.05 \cdot Z^* - v(L_{\lambda^t,\mu^t,\rho^t}))}{\sum_{i \in I}((L_i^t)^2 + \sum_{j \in J}(M_{ij}^t)^2) + \sum_{j \in J}(R_j^t)^2}$$

with Z^* the best primal bound, as proposed by Fisher (1985). We iteratively repeat this process until we either hit a limit on the number of iterations or the computed lower bound exceeds Z^*. As proposed by Beasley (1988), we also multiply $Z_{\max}$ factor 1.05. Furthermore, we observe an improved performance by defining $\mu_{ik} = 0$ for all $i \in I, j \in J$, and $k \in J^{>}_{ij}$ if k does not occur in updated preference list L_i.

Note, if $v(L_{\lambda^{t+1},\mu^{t+1},\rho^{t+1}}) > v(L_{\lambda^t,\mu^t,\rho^t})$, then we set the dual bound to $v(L_{\lambda^{t+1},\mu^{t+1},\rho^{t+1}})$. If the best value found so far during the subgradient method stays unchanged for a fixed amount of iterations, we (a) divide step size α^t by two for the remainder of the iterations and (b) reconsider the Lagrangian multipliers associated with the best lower bound found during the currently running subgradient method.

In a tree node that is not the root node, we set the initial Lagrangian multipliers to the best multipliers found in the parent node as proposed by Beasley (1988) and Fisher (2004). Then, we proceed analogously to the root node. We refer to Fisher (1985) for a visualisation of this procedure within a B&B algorithm.

Upper bounds. Based on solutions for the Lagrangian relaxation, we can derive upper bounds. For that, we start routine *greedy min-cost flow* in Section 4.6 with all facilities set to one in the best solution to the Lagrangian relaxation as potentially open facilities, and execute the routine based on these starting values. This is similar to the approach used by Görtz and Klose (2012) for the CFLP.

4.8. Pruning

We prune by either bound, if the dual bound of any node is greater or equal than our current incumbent, or by infeasibility if Algorithm 1 determines a node to have no more feasible solutions. If L_i only contains a single element or $L_i[0]$ is opened, we can immediately fix the assignment of i. To keep track of that, we note the opened facilities in every node. If an opened facility is to be deleted, the corresponding node is infeasible, i.e. we prune it as well.

5. Computational Results

In this section, we compare the performance of Gurobi for solving formulation (A.1) with and without our preprocessing (cf. Section 4.1), denoted with *Gurobi* and $Gurobi_{PPC}$, respectively. Furthermore, we test two variants of our B&B algorithm. In the first variant, denoted with $B\&B$, the lower bound at each node is defined as the maximum of the solution from the Lagrangian relaxation and the min-cost-flow problem. However, computing lower bounds by solving the Lagrangian relaxation is computationally expensive, especially for very big instances consisting of 1000 customers and facility locations, each. Therefore, we also observe the performance of our B&B algorithm if solely lower bounds based on the min-cost-flow problem are considered. We refer to the second variant as $B\&B_{\text{Flow}}$.

5.1. Computational Setup

Instances. In our our tests, we consider 104 instances developed for the classical capacitated facility location problem, cf. Table 1, which are available via Università degli Studi di Brescia (2025).

Instance-families' size (# Locations,# Customers)	# Instances per family	Source
$(300, 300)$, $(500, 500)$, $(700, 700)$, $(1000, 1000)$	20	Avella and Boccia (2009)
$(50, 75)$, $(75, 100)$	12	Taken from Beasley (1988), modified according to Cánovas et al. (2007)

Table 1: A total of $4 \cdot 20 + 2 \cdot 12 = 104$ instances from two instance families are considered in this computational study.

The considered instances consist of six different sizes ranging between 50 facility locations and 75 customers and 1000 facility locations and customers, each. The 1000 times 1000 instances are the largest symmetric instances available from that set. The instances consisting of 50 locations, 75 customers and 75 locations, 100 customers are modified variants of instances *capa, capb,* and *capc* from Beasley (1988). Analogously to Cánovas et al. (2007), we consider the first 50 (75) locations and 75 (100) customers and multiply the opening costs with 0.0375 (0.075). In a similar manner to Calvete et al. (2020), we set the capacity of each facility $j \in J$ to $\lceil Q_j/(\xi \cdot \bar{d}) \rceil$ with Q_j the original capacity of j, $\bar{d}$ the average of the customer demands in the original instance, and $\xi = 2, 3, 4$ for *capa, capb,* and *capc*, respectively. In contrast to Calvete et al. (2020), who set $\xi = 1$, we choose higher values as, otherwise, several instances turn infeasible for one of the considered preference types.

Preference types. We consider two preference types. In the first type, preferences are defined by assignment costs: Customer $i \in I$ prefers facility $j \in J$ over facility $k \in J$ if the assignment costs of i to j are lower than the costs of assigning i to k. We denote this preference type with *CA*. In the second preference type, we slightly perturb the first preference type. Like before, facilities with small assignment costs are generally preferred over facilities with great assignment costs. Yet, a customer's preference ordering regarding facilities with similar assignment costs might deviate from the preference ordering according to lowest assignment costs. In order to determine the second preference ordering, we follow the procedure described by Cánovas et al. (2007). We denote this preference type with *PCA*. If a customer is indifferent between multiple

instances in any of the considered instances, we break ties arbitrarily. Among the instances studied here, 7 are infeasible if preferences are defined by assignment costs. Previous work shows that the preference type affects the problem's theoretical and computational complexity (Büsing et al., 2025; Büsing et al., 2025). We therefore study the performance of all four algorithms for each preference type individually.

Hard- and Software. All algorithms have been implemented in *Python* 3.10.4, and all experiments have been performed on a Linux machine (Rocky Linux 9.6 Blue Onyx) using a single 2.1 GHz CPU and 10 GB RAM. We use an integer program for benchmarking, its CFLP-SCP formulation is given in Appendix A. We solve the ILPs with Gurobi (2025) version 11.0.3, restricted to a single CPU as well for fairness of comparison. We solve the min-cost-flow problems described in Section 4.7 with the *NetworkX*-library for python (Hagberg et al., 2008). We set the timelimit to 3 600 seconds. For solving Lagrangian relaxation (3) to obtain a lower bound for approach B&B, we perform 6 iterations of the subgradient method. If the best value found so far during the subgradient method stays unchanged for 4 iterations, we divide the step size α by two for the remainder of the iterations.

All data and code are publicly available via Engelhardt and Wrede (2025).

5.2. Comparison of the Algorithms' Performance

We evaluate the performance of the studied algorithms by considering the *shifted geometric mean* as well as *primal-dual, primal* and *dual integrals*. See Appendix C for further information on their definitions and interpretations. Table 2 summarises the results for each algorithm and both preference types.

		Gurobi		**Gurobi$_{\mathbf{PPC}}$**		**B&B**		**B&B$_{\mathbf{Flow}}$**	
		CA	PCA	CA	PCA	CA	PCA	CA	PCA
1	primal solution found	49	49	**78**	69	**78**	83	**78**	**84**
1a	– proven optimum	24	**12**	**27**	**12**	12	–	–	–
1b	– gap remaining	25	37	49	55	66	83	**78**	**84**
1c	– no dual bound	–	–	2	2	–	–	–	–
2	infeasiblility detected	4	0	**7**	0	**7**	0	**7**	0
3	total sum	53	49	**85**	69	**85**	83	**85**	**84**
4	no solution found	51	55	**19**	35	**19**	21	**19**	**20**
4a	– timeout during preprocessing	–	–	19	21	19	21	19	20
4b	– memory error in root node	51	55	–	14	–	–	–	–
5	gap after 1 hour [%]	22.71	31.82	**11.48**	**26.63**	21.70	44.64	36.52	48.83
6	primal-dual integral	195.44	326.19	**16.16**	141.73	52.76	149.10	49.79	**110.79**
7	primal integral	208.77	457.65	32.55	253.16	**17.51**	**29.81**	23.55	50.64
8	dual integral	274.08	674.76	**36.29**	338.21	232.68	**316.33**	501.57	327.56
9	ø memory/node [*MB*]	5.17	5.32	**1.73**	**4.32**	10.00	14.34	8.94	12.35
10	time to optimality [*s*]	9.00	**158.70**	**8.98**	161.20	501.27	–	–	–
11	time to infeasibility [*s*]	550.14	–	**20.76**	–	36.89	–	37.57	–

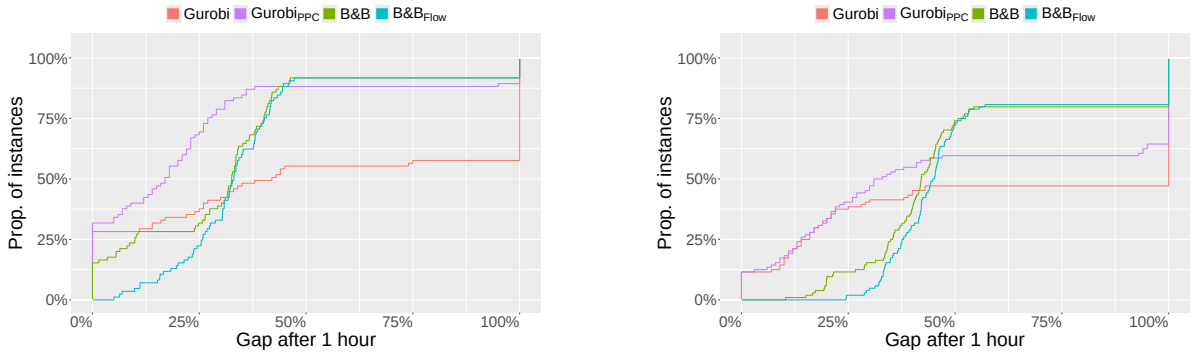
Table 2: Computational results for the different approaches studied here. We aggregate the results starting from the gaps after 1 hour (Row 5) by using the shifted geometric mean with shifting parameter $s = 1\%$ for the gaps and integrals, and $s = 1$ for the times and average memory per node. Please note that time to optimality/infeasibility are only computed for instances solved to optimality/infeasibility. Thus, these values are not directly comparable between algorithms.

Solutions found. Rows 1 – 3 show for how many instances either infeasibility was detected or a solution was found for each algorithm. Rows 1a – 1c then rank the found solutions based on quality: Proven optima, solutions where we ran out of time with a non-trivial optimality gap remaining and cases where a primal solution was found but no dual bound but the trivial lower bound zero. The latter case only happens if

Gurobi is given our preprocessing, which allows it to heuristically find a primal solution, and then fails to solve the root node LP relaxation in under an hour. Row 2 contains all instances where infeasibility was detected. Since our preprocessing does so automatically, $Gurobi_{\text{PPC}}$ achieves the same performance as the B&B algorithms. Overall, the B&B algorithms find feasible solutions more consistently, but both *Gurobi* variants succeed in proving optimality for many more instances.

For some instances, neither algorithm finds a solution, see Row 4. For instances of size 1000, we find that our preprocessing tends to time out, which can be seen in Row 4a. Except for this, the B&B runs smoothly. In comparison, *Gurobi* regularly struggles to even solve the root node LP, especially when not given access to our preprocessing, see Row 4b.

MIP gaps. For most applications, not only the number of instances solved to optimality, but also the progress towards good solutions and proving optimality constitute important metrics. In Table 2, this is given in Row 5. Note that these values can not be directly compared between instances, since more instance terminate without solution and thus any gap for *Gurobi*. To provide a more detailed picture, below, Figure 5 illustrates the MIP-gap achieved at the end of one hour across all feasible instances:



(a) Preferences defined by assignment costs.

(b) Preferences defined by perturbed assignment costs.

Figure 5: Cumulative distribution of the remaining gaps after 3 600 seconds for *Gurobi*, $Gurobi_{\text{PPC}}$, and both variations of the combinatorial B&B for feasible instances. The x-axis depicts an upper bound on the gap γ obtained after 1 hour, the y-axis depicts the percentage of instances that obtain an objective value that is at most γ. Algorithms that achieve high values for small x-values are most desirable. If no solution is found for an instance that is not known to be infeasible, we set the gap obtained by the respective algorithm to 100%.

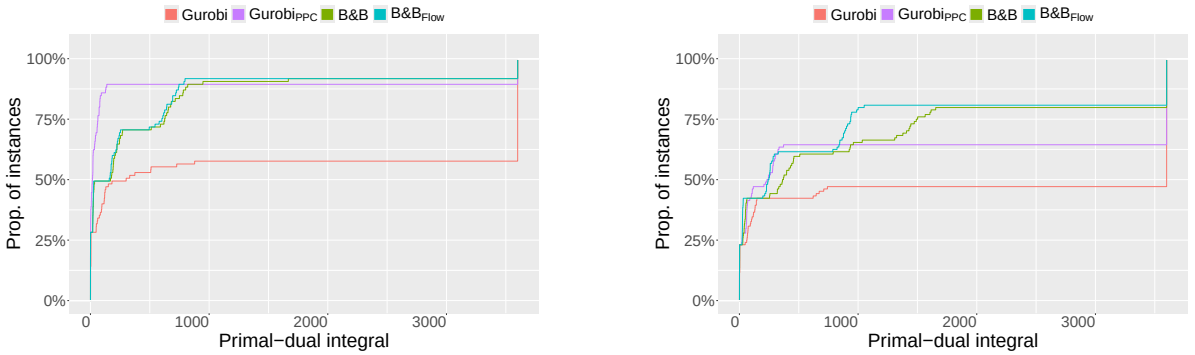
If preferences are defined by assignment costs, as seen in Figure 5a, Gurobi$_{\text{PPC}}$ performs best on most instances, outperforming the default B&B. Interestingly, a more mixed picture emerges if preferences are given by perturbed assignments, see Figure 5b. Here, our preprocessing has no notable impact on instances that are solved both by *Gurobi* and $Gurobi_{\text{PPC}}$. However, it allows $Gurobi_{\text{PPC}}$ to solve more instances. This adds to previous results from Büsing et al. (2025), who already showed that a weaker version of Algorithm 2 can improve performance on assignment-cost instances, but did not detect any notable positive effect on perturbed instances. In comparison, we find that the improved Algorithm 2 does contribute to a larger number of instances being solved, even if preferences do not directly correspond to assignment costs.

Overall, we see that for some instances, *Gurobi* manages to find an optimal solution in under an hour, especially for instances with assignment-cost preferences. Based on analysis of solver logs, we conjecture that this is due to the strength of LP-based dual bounds outperforming our own dual bounds, especially if enhanced by valid inequalities.

Row 9 together with Row 4b illustrate the advantages and disadvantages of using a solver: small instances are solved to optimality by *Gurobi*, with low memory requirement per node, and large instances fail completely, often already running out of memory in the root node. In comparison, since we only generate local preference lists as needed and only track the parameters necessary to reconstruct them, this is less of a limiting factor for the combinatorial B&B algorithms, which then obtain optimality gaps of at most 60%

even for big instances for which the *Gurobi*-based approaches fail to produce any solution. Lastly, note that the dual bounds based on the Lagrangian relaxation have an effect if preferences are defined by assignment costs.

Primal–dual integrals. The gap at termination is prone to variations caused by bound changes around the time limit (Berthold, 2013). Furthermore, the optimality gap does not give insights into the performance of the primal and dual bounds. An alternative measure are the primal-dual as well as the primal and dual integrals (Achterberg, 2007; Berthold, 2013). In Table 2, these are given in Rows 6 – 8. Additionally, we visualise the combined primal–dual integrals over time in Figure 6 below.



(a) Preferences defined by assignment costs.

(b) Preferences defined by perturbed assignment costs.

Figure 6: Cumulative distribution function of the primal-dual integrals for *Gurobi*, *Gurobi*$_{\text{PPC}}$, and both variations of the combinatorial B&B. The x-axis depicts an upper bound on the primal-dual integral γ for each instance, the y-axis depicts the percentage of instances that obtain a primal-dual integral of at most γ. Algorithms that achieve high values for small x-values are most desired. If no solution is found for an instance that is not known to be infeasible, we set the primal-dual integral to 3 600.

For preferences defined by assignment costs, see Figure 6a, Gurobi$_{\text{PPC}}$ clearly outperforms the other approaches on most instances. This is predominantly due to quickly solving a subset of instances to optimality. In comparison, for preferences defined by perturbed assignment costs, see Figure 6b, both B&B algorithms outperform *Gurobi*, which terminates on many instances due to reaching 10 GB RAM. At the same time, *Gurobi*$_{\text{PPC}}$, B&B, and B&B$_{\text{Flow}}$ are difficult to compare: Each of these algorithms has phases where it dominates the other algorithms.

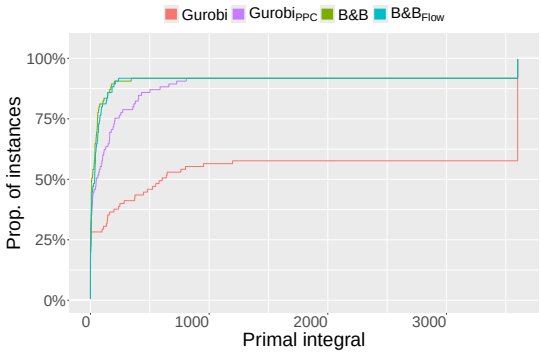
Note that the distributions of approaches B&B and B&B$_{\text{Flow}}$ have visibly constant phases. Each part between such constant phases corresponds to distinct instance sizes, with lower integral values corresponding to smaller instances. This behaviour is likely due to the time needed to solve subproblems increasing with increasing instance size. Finally, we observe that in terms of progress on primal and dual bounds, computing dual bounds derived by solving the Lagrangian relaxation for the B&B is not worth the additional effort if preferences are defined by assignment costs.

Primal integrals. Now, we look at primal and dual integral separately, beginning with the primal integral for both preference types depicted in Figure 7.

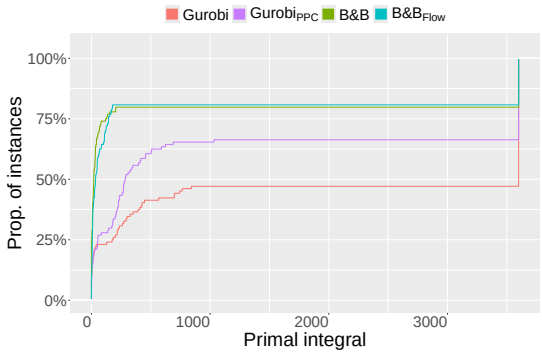
For both preference types, we observe that B&B performs best in obtaining good primal bounds early on, independently of the instance sizes. If preferences are defined by assignment costs, Gurobi$_{\text{PPC}}$ is also competitive, yet, it is dominated by B&B for most instances.

Dual integrals. Figure 8 depicts the dual integral for both preference types.

Here, we see that *Gurobi*$_{\text{PPC}}$ clearly outperforms all other algorithms in terms of quickly finding strong dual bounds for preferences defined by assignment costs. Notably, these bounds are stronger than even the bounds we find via Lagrangian relaxation, pointing towards an advantage of *Gurobi*.

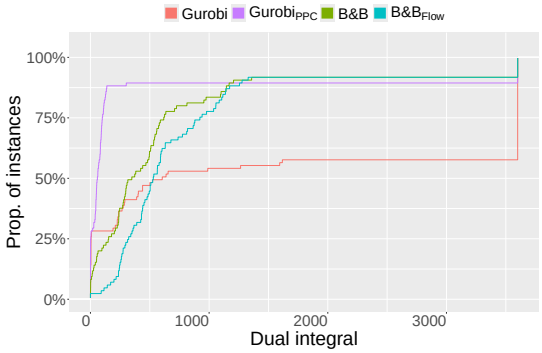


(a) Preferences defined by assignment costs.

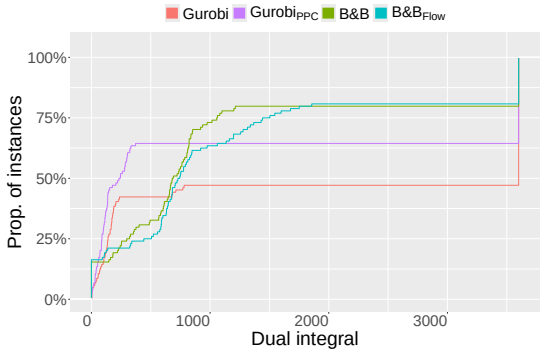


(b) Preferences defined by perturbed assignment costs.

Figure 7: Cumulative distribution function of the primal integrals for *Gurobi*, *Gurobi*$_{\text{PPC}}$, and both variants of the combinatorial B&B. The x-axis depicts an upper bound on the primal integral γ for each instance, the y-axis depicts the percentage of instances that obtain a primal integral of at most γ. Algorithms that achieve high values for small x-values are most desired. If no primal solution is found for an instance that is not known to be infeasible, we set the primal integral to 3 600.



(a) Preferences defined by assignment costs.



(b) Preferences defined by perturbed assignment costs.

Figure 8: Cumulative distribution of the dual integrals for *Gurobi*, *Gurobi*$_{\text{PPC}}$, and both variants of the combinatorial B&B. The x-axis depicts an upper bound on the dual integral γ for each instance, the y-axis depicts the percentage of instances that obtain a dual integral of at least γ. Algorithms that achieve high values for small x-values are most desired. If no dual bound is found for an instance that is not known to be infeasible, we set dual integral to 3 600.

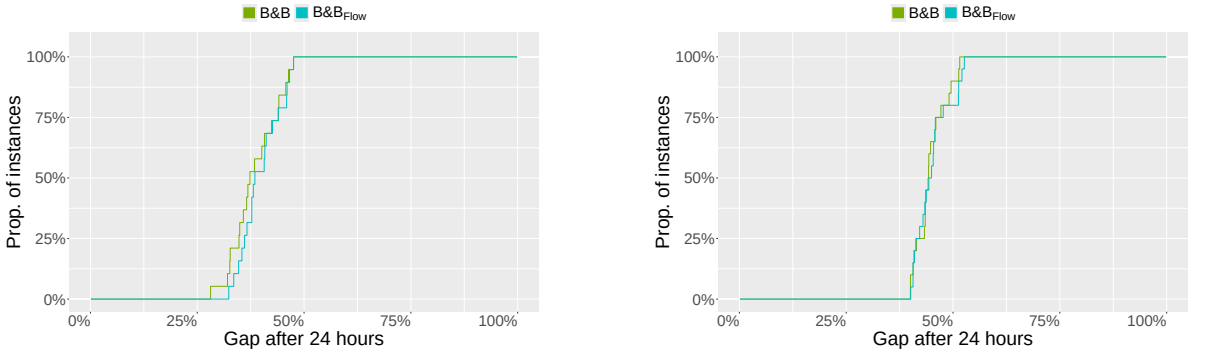
If preferences are defined by perturbed assignment costs, Gurobi$_{\text{PPC}}$ dominates Gurobi, and the B&B-based algorithms perform similarly for most instances. However, whether the B&B-based algorithms or *Gurobi* perform better depends on the point in time. Based on solver logs, we found *Gurobi* extremely effective in improving the dual bound in the root node, but then often making little and expensive progress in the B&B tree. In comparison, combinatorial B&B progress is more consistent, but overall bounds tend to be weaker, which also has a tendency to lead to large B&B trees.

Longer time horizons. The 10 GB RAM we used for memory in this study is the amount of memory per core available at RWTH Aachen's High Performance Cluster's *Claix-2023 (large memory)*[1] partition. We acknowledge that more memory can be provided, but doing so is unlikely to result in an MILP that scales well to solve large instances, especially if the solver already fails at solving the root node. In comparison,

[1]See https://help.itc.rwth-aachen.de/en/service/rhr4fjjutttf/article/9108f4a6f43c40a3a168919afd36839d/, last accessed on November 4, 2025.

since facility locations problems are generally strategic, time is less of an restriction. Thus, we are interested in to what extent providing B&B and B&B$_{\text{Flow}}$ with more time leads to progress on hard instances.

Figure 9 shows the progress made on solving the hardest instances we considered (1000 customers and 1000 potential facility locations) after 24 hours, again with a single core.



(a) Preferences defined by assignment costs.

(b) Preferences defined by perturbed assignment costs.

Figure 9: Remaining gap after 24 hours for instances of size 1000 for both variants of the combinatorial B&B.

We see that progress is very much instance-dependent, with some of the gap being closed for half of the instances and little to no progress for the other half. As Figure 10 illustrates, this can be explained by the time needed to solve individual nodes. In the first row, we see the number of nodes solved in the B&B tree for both algorithms. Clearly, some progress is being made if preferences are defined by assignment cost, but fewer than 40 nodes each are solved if preferences are defined by perturbed assignment costs.

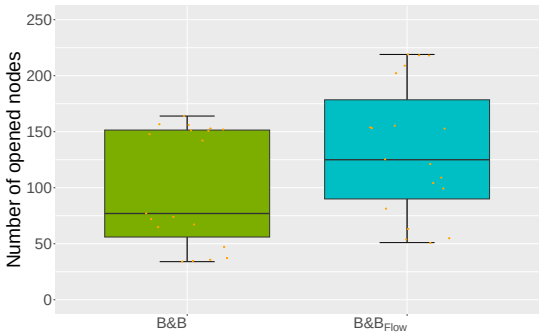
We reason that the number of nodes solved is due to the time per node, which is by the second row: Here, for the perturbed assignments, we require up to 6 hours on average to solve a single node, which clearly is not practical. The fact that this is instance-dependent, suggest that perturbed assignments are different and harder than closest assignments and approaches that work well for the latter do not necessarily do the same for perturbed assignments.

6. Conclusion

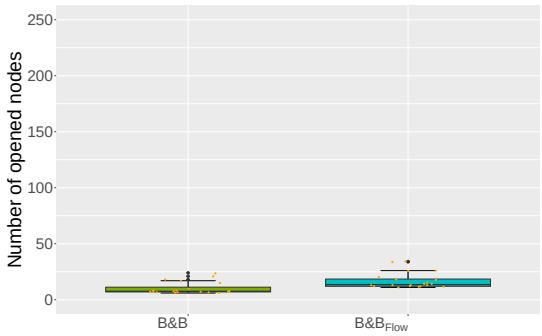
In this work, we developed a combinatorial branch-and-bound algorithm for the capacitated facility location problem with strict customer preferences (CFLP-SCP). The B&B algorithm utilises that (i) primal and dual CFLP-SCP bounds can be determined in polynomial time by iteratively closing overloaded facilities/ computing network flows and (ii) the set of open facilities in the first feasible solution contains all optimal solutions. Thus, all possibilities of opening and closing facilities can be enumerated via branching on the location decisions.

We compared our B&B algorithm with a state of the art MILP solver. In summary, we manage to find good solutions more consistently, but the solver is better at computing strong dual bounds to prove optimality, especially for small instances. A major advantage of the B&B approach is its memory usage allowing it to scale to larger instances.

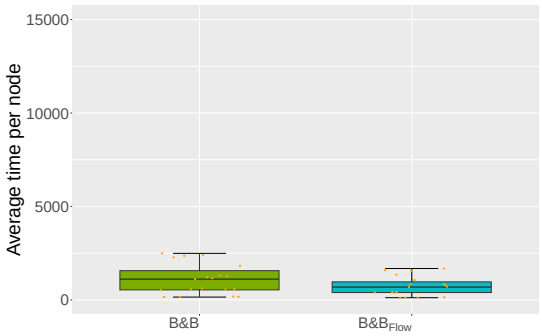
We also found that solver performance is highly dependent on instance structure, with perturbed assignment costs being harder. Here, the modified preprocessing not only has an effect on default assignment cost instances, as seen in Büsing et al. (2025), but also on the performance of perturbed assignment cost instances, which is why we recommend using the modified preprocessing for any approach trying to solve CFLP-SCP, regardless of instances type. Apart from that, both combinatorial B&B and ILP are a effective strategies for solving CFLP-SCP. For smaller instances and proving optimality, we recommend a ILP solver; for larger instances, especially with perturbations, we recommend using the combinatorial approach without the Lagrangian relaxation instead.



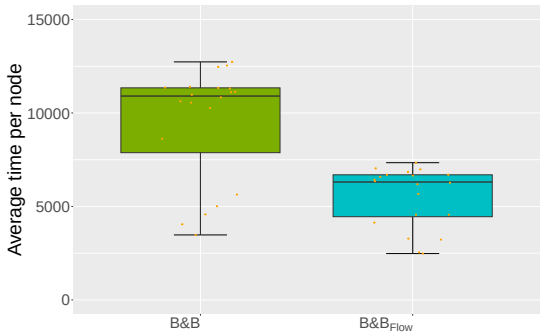
(a) Preferences defined by assignment costs.



(b) Preferences defined by perturbed assignment costs.



(c) Preferences defined by assignment costs bounds.



(d) Preferences defined by perturbed capacity bounds.

Figure 10: Memory for instances of size 1000 for both variants of the combinatorial B&B. The upper row gives the number of nodes solved, the lower row the average time per node in seconds

Future work should focus on combining the strengths of both algorithms, e.g. trying to situationally use stronger, LP-based bounds as part of a custom B&B and/or integrating domain propagation based on the results of this work into a solver as a callback, e.g. by integrating both into an open source framework such as the SCIP Optimization Suite (2025). Focusing on the B&B approach, it might also be interesting to study further variable selection approaches, as well as alternative approaches for lower bounds. From a computational perspective, a reimplementation in a faster programming language (Julia, Rust) might be helpful, and both solvers and the given B&B approaches are suitable for parallelization.

Finally, we would like to point out that many of the methods outlined in this work are generic and not restricted to the CFLP-SCP. As such combinatorial B&B may be a promising approach for any computationally hard problem where lower and upper bounds can be computed easily. This applies to many types of facility location and planning problems where binary decisions in some first stage (e.g. opening facilities) determine some second stage assignment costs.

Appendix A. IP Formulation

We use integer linear programming formulation (A.1) as a baseline for evaluating the performance of our algorithm. The formulation uses decision variables $y_j \in \{0, 1\}$, which indicate whether facility $j \in J$ is opened ($y_j = 1$), and variables $x_{ij} \in \{0, 1\}$, which indicate whether customer $i \in I$ is assigned to facility $j \in J$ ($x_{ij} = 1$).

$$\min \quad \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} c_{ij} x_{ij} \tag{A.1a}$$

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

$$\sum_{i\in I} d_i x_{ij} \leq Q_j y_j \qquad \forall j \in J \tag{A.1c}$$

$$\sum_{k\in J_{ij}^{>}} x_{ik} + y_j \leq 1 \qquad \forall i \in I,\ j \in J \tag{A.1d}$$

$$x_{ij} \leq y_j \qquad \forall i \in I,\ j \in J \tag{A.1e}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I,\ j \in J \tag{A.1f}$$

$$y_j \in \{0,1\} \qquad \forall j \in J \tag{A.1g}$$

Objective function (A.1a) minimises the sum of total opening costs and the cost of assigning each customer. Constraints (A.1b) guarantee that each customer is assigned to a facility. Constraints (A.1c) ensure that the total demand assigned to an open facility respects its capacity and that customers can only be assigned to open facilities. Constraints (A.1d), which have first been introduced by Wagner and Falkson (1975), guarantee that customer preferences are respected. More specifically, these constraints ensure that, if facility j is open, then customer $i \in I$ cannot be assigned to a facility $k \in J$, which they prefer less than j. Linking constraints (A.1e) are redundant, but are well known to significantly strengthen the linear programming relaxation of capacitated facility location problems.

Theorem 1. *Consider formulation* (A.1) *for an instance of the CFLP-SCP. If the location variables take binary values, i.e., constraints* (A.1g) *hold, then we may relax the assignment variables in formulation* (A.1) *to* $x_{ij} \in [0,1]$ *for each* $i \in I, j \in J$ *independently of the objective function. If the assignment variables take binary values, i.e., constraints* (A.1f) *hold, then we may relax the location variable values to* $y_j \in [0,1]$ *for each* $j \in J$.

Proof. We start by proving the first claim. Consider a feasible solution $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^{|I|+|I||J|}$ for formulation (A.1). Consider an arbitrary but fixed customer $i \in I$ and a facility $j^* \in J$ with $y_{j^*} = 1$ and $y_j = 0$ for all $j \in J_{ij^*}^{<}$. Such a most-preferred open facility exists for each customer since each customer has a strict preference ordering and location variable values are binary due to constraints (A.1g). Due to constraints (A.1d) and $y_{j^*} = 1$, it is $x_{ik} = 0$ for all $k \in J_{ij^*}^{>}$. Then, due to constraints (A.1e) and $y_j = 0$ for all $j \in J_{ij^*}^{<}$, constraints (A.1b) yield $x_{ij^*} = 1$. In conclusion, the assignment variables take binary values in any feasible solution. Since we did not use information implied by the objective function, this property holds independently of it.

Let us now prove the second claim. Consider a solution $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^{|I|+|I||J|}$ for formulation (A.1). Consider an arbitrary but fixed facility $j \in J$. If at least one $i \in I$ is served at j, then $y_j = 1$ follows immediately due to constraints (A.1e) and (A.1f). Suppose that no customer is served at j. Set $y_j = 0$. Constraints (A.1c) and (A.1e) are met for j since both their left- and right-hand side are zero. Constraints (A.1d) are also met for j since $\sum_{k\in J_{ij}^{>}} x_{ik} + y_j = \sum_{k\in J_{ij}^{>}} x_{ik} \leq \sum_{k\in J} x_{ik} = 1$ holds for each $i \in I$ due to constraints (A.1b). Lastly, note that it is advantageous to set $y_j = 0$ if no customer is served at j due to objective function (A.1a). □

Note that this property only holds if customer preferences are strict.

Appendix B. Pseudocode

We compute an initial feasible solution for any CFLP-SCP instance by applying the Algorithm 1. Büsing et al. (2025) show the correctness of this algorithm.

Algorithm 1

Require: CFLP-SCP instance with strict customer preferences.
Ensure: Feasible solution for the CFLP-SCP.
Open all facilities, and assign each customer to their most preferred open facility.
while an open facility is overloaded **do**
Close the overloaded facility.
Assign all customers who used to be served by it to their next preferred open facility.
end while
if all facilities are closed **then**
return The considered instance is infeasible.
else
return The set of open facilities implies a feasible solution.
end if

We strengthen the concept of implied demands as described in Büsing et al. (2025) by applying Algorithm 2.

Algorithm 2

Require: CFLP-SCP instance with strict customer preferences.
Ensure: Strengthened implied demands for customer $i \in I$.
for $k = 1, \ldots, |L_i|$ **do**
Suppose that i is served at $L_i[k]$ and, therefore, all facilities in $J^{<}_{iL_i[k]}$ are closed.
Apply Algorithm 1 to $J \setminus J^{<}_{iL_i[k]}$ and denote the algorithm's output with $J^* \subseteq J \setminus J^{<}_{iL_i[k]}$.
if $J^* = \emptyset$ **then**
Delete $L_i[k], L_i[k+1], \ldots L_i[|L_i|]$ from i's preference list.
return Updated preference list L_i of i and their implied demand at each $j \in L_i$.
end if
if $L_i[k] \notin J^*$ **then**
Set the implied demand of i at $L_i[k]$ to $\mathcal{D}(i, L_i[k]) = Q_{L_i[k]} + 1$.
else
Update the set of implied customers to

$$\mathcal{I}(i, L_i[k]) = \{i\} \cup \{\ell \in I : (J^{<}_{\ell L_i[k]} \cap J^*) \subseteq (J^{<}_{iL_i[k]} \cap J^*) \cup \{L_i[k]\}\},$$

and update the implied demand to $\mathcal{D}(i, L_i[k]) = \sum_{\ell \in \mathcal{I}(i, L_i[k])} d_\ell$.
end if
end for
return Updated positions of the least preferred potential facility of each customer and, for each facility, a list of customers that must not be served at it.

Appendix C. Performance measures

We consider the following concepts to analyse the performance of the algorithms in Section 5.

- The *shifted geometric mean* is defined as $(\Pi_{i=1}^{k}(v_i + s)^{1/k}) - s$ for values $v_1, \ldots, v_k \in \mathbb{R}_{\geq 0}$ and a *shifting parameter* $s \in \mathbb{R}_{\geq 0}$ (Achterberg, 2007). Conversely to the arithmetic mean, the shifted geometric mean is not overly sensitive to very small or very large values in the computations.
- The *primal-dual integral* reduces the impact of changes in dual or primal bounds that happen around the time limit. The following example is taken from Berthold (2013). Suppose there are two algorithms,

which have the same gap for the first 3599 seconds. Suppose the first algorithm improves the gap after 3599 seconds, and the other algorithm improves it after 3601 seconds. Then, the gap after one hour can be arbitrarily different. The primal-dual integral, however, will differ by less than 0.1%.

In order to compute the primal-dual integral, we first have to compute a *gap function*

$$\gamma(t) = \begin{cases} 1 & \text{if no feasible solution is found until time } t \\ \frac{|f(x(t)) - z(t)|}{|f(x(t))|} & \text{otherwise} \end{cases}$$

with $f(x(t))$ the objective value of the best primal solution found up to time t and with $z(t)$ the current best provable dual bound on the optimum. The gap function $\gamma(t)$ is a step function. We therefore compute the integral by considering the sum of rectangular areas, i.e., $P(T_{max}) = \sum_{i=1}^{n} \gamma(i) \cdot (t_i - t_{i-1})$ with T_{max} the minimum of the time limit and the time needed to obtain optimality, with n the total number of changes in the bound, and with t_i the time when the i-th change in the bound is observed. The primal-dual integral simultaneously rewards an algorithm for finding good primal solutions quickly, improving the dual bound quickly, and its overall speed. A lower value of the primal-dual integral indicates better overall performance.

- The *primal integral* (Berthold, 2013) depends on the quality of primal solutions and on the time when they are found. Analogously to the primal-dual integral, we first compute a so-called *primal gap function*. Instead of subtracting the best dual bound, we subtract the best known objective value. We compute the *dual integral* analogously to the primal integral. The lower the value of the primal (dual) integral, the better the algorithm's primal performance, as it indicates that good solutions were found quickly.

Appendix D. Tables of all computational results

Tables D.3 to D.8 depict the actual computational results obtained in our computational study.

Table D.3: Computational results for instances of sizes $(50, 75)$ and $(75, 100)$.

	Gurobi				*Gurobi*$_{\text{PPC}}$				B&B				B&B$_{\text{Flow}}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time
1_0_cap_a_50_75	1449086	1449086	**0.0**	3	1449086	1449086	**0.0**	1	1449086	1449086	**0.0**	3003	1392681	1466265	5.02	3600
1_0_cap_b_50_75	877702	877702	**0.0**	1	877702	877702	**0.0**	0	877702	877702	**0.0**	136	816616	877702	6.96	2757
1_0_cap_c_50_75	732615	732615	**0.0**	1	732615	732615	**0.0**	2	732615	732615	**0.0**	149	604259	740075	18.35	3122
2_0_cap_a_50_75	1290085	1290085	**0.0**	8	1290003	1290085	**0.0**	3	1290084	1290085	**0.0**	3550	1156401	1301128	11.12	3329
2_0_cap_b_50_75	812756	812756	**0.0**	1	812756	812756	**0.0**	0	812751	812756	**0.0**	495	691225	821270	15.83	3315
2_0_cap_c_50_75	716193	716193	**0.0**	1	716193	716193	**0.0**	0	716107	716193	0.01	16	556628	717348	22.4	3583
3_0_cap_a_50_75	1207727	1207727	**0.0**	7	1207727	1207727	**0.0**	3	1207727	1207727	**0.0**	1234	1043050	1230584	15.24	3097
3_0_cap_b_50_75	762648	762648	**0.0**	1	762648	762648	**0.0**	0	762647	762648	**0.0**	665	599375	762954	21.44	3600
3_0_cap_c_50_75	707654	707654	**0.0**	1	707654	707654	**0.0**	0	707654	707654	**0.0**	34	521679	708809	26.4	3600
4_0_cap_a_50_75	1144110	1144110	**0.0**	7	1144110	1144110	**0.0**	4	1144110	1144110	**0.0**	1289	922477	1153612	20.04	3600
4_0_cap_b_50_75	751971	751971	**0.0**	2	751971	751971	**0.0**	0	751971	751971	**0.0**	696	553344	756100	26.82	3600
4_0_cap_c_50_75	707654	707654	**0.0**	1	707654	707654	**0.0**	0	707654	707654	**0.0**	47	465830	708809	34.28	3600
1_0_cap_a_75_100	3161664	3161664	**0.0**	9	3161664	3161664	**0.0**	5	3000019	3205280	6.4	3600	3001459	3205280	6.36	1738
1_0_cap_b_75_100	1686745	1686745	**0.0**	6	1686745	1686745	**0.0**	6	1559558	1738441	10.29	3428	1464949	1757623	16.65	1886
1_0_cap_c_75_100	1221518	1221518	**0.0**	11	1221518	1221518	**0.0**	4	1158487	1260639	8.1	3600	966244	1264966	23.62	2792
2_0_cap_a_75_100	2548469	2548674	**0.0**	147	2548674	2548674	**0.0**	186	2366888	2627186	9.91	3600	2320853	2574732	9.86	2925
2_0_cap_b_75_100	1524369	1524467	**0.0**	9	1524467	1524467	**0.0**	6	1426617	1562566	8.7	3600	1279736	1593762	19.7	2443
2_0_cap_c_75_100	1143610	1143688	**0.0**	22	1143688	1143688	**0.0**	3	1120917	1161457	3.49	3600	874091	1180110	25.93	3444
3_0_cap_a_75_100	2246957	2246957	**0.0**	162	2246957	2246957	**0.0**	109	2148360	2275156	5.57	3600	2021906	2275156	11.13	3600
3_0_cap_b_75_100	1379664	1379732	**0.0**	48	1379732	1379732	**0.0**	18	1294455	1449762	10.71	3600	1094579	1467907	25.43	3600
3_0_cap_c_75_100	1116400	1116400	**0.0**	18	1116400	1116400	**0.0**	5	1101769	1118451	1.49	3600	793932	1140904	30.41	3600
4_0_cap_a_75_100	2053138	2053138	**0.0**	63	2053043	2053138	**0.0**	30	1942247	2055218	5.5	3600	1763801	2092828	15.72	3600
4_0_cap_b_75_100	1321516	1321516	**0.0**	31	1321516	1321516	**0.0**	16	1237799	1374650	9.96	3600	993016	1394234	28.78	3600
4_0_cap_c_75_100	1079491	1079491	**0.0**	6	1079491	1079491	**0.0**	1	1079490	1079491	**0.0**	1819	724361	1119532	35.3	3600
1_1_cap_a_50_75	2160029	2160223	**0.0**	435	2160114	2160223	**0.0**	421	1829060	2194973	16.67	3600	1657503	2194973	24.49	3600
1_1_cap_b_50_75	1650453	1650453	**0.0**	30	1650453	1650453	**0.0**	18	1346607	1673449	19.53	3600	1263520	1675934	24.61	3179
1_1_cap_c_50_75	1336949	1336949	**0.0**	23	1336949	1336949	**0.0**	19	1088375	1358801	19.9	3600	837707	1358801	38.35	3600
2_1_cap_a_50_75	1948792	1948850	**0.0**	642	1948850	1948850	**0.0**	360	1651512	1997080	17.3	3600	1422641	1997080	28.76	3600
2_1_cap_b_50_75	1525217	1525217	**0.0**	43	1525217	1525217	**0.0**	150	1202806	1529794	21.37	3600	1074400	1533310	29.93	3600
2_1_cap_c_50_75	1325867	1325867	**0.0**	30	1325867	1325867	**0.0**	99	1069575	1334808	19.87	3600	771081	1330461	42.04	3600
3_1_cap_a_50_75	1856697	1856697	**0.0**	636	1856569	1856697	**0.0**	556	1595080	1879001	15.11	3600	1290137	1877368	31.28	3600
3_1_cap_b_50_75	1457020	1457077	**0.0**	301	1457033	1457077	**0.0**	217	1142921	1457077	21.56	3600	964014	1457077	33.84	3600
3_1_cap_c_50_75	1325867	1325867	**0.0**	36	1325867	1325867	**0.0**	36	1064530	1328337	19.86	3600	714933	1329849	46.24	3600
4_1_cap_a_50_75	1762473	1762502	**0.0**	977	1762364	1762502	**0.0**	1322	1598351	1782427	10.33	3600	1190777	1782427	33.19	3600
4_1_cap_b_50_75	1412296	1412296	**0.0**	482	1412296	1412296	**0.0**	357	1139257	1412315	19.33	3600	885179	1412315	37.32	3600
4_1_cap_c_50_75	1312371	1312404	**0.0**	255	1312404	1312404	**0.0**	161	1063038	1329067	20.02	3600	669323	1312404	49.0	3600
1_1_cap_a_75_100	3743958	4196840	**11.0**	3600	3732978	4190004	**11.0**	3600	3177655	4457908	28.72	3600	3137251	4433708	29.24	3600
1_1_cap_b_75_100	2355702	2615083	10.0	3600	2500264	2579421	**3.0**	3600	1798727	2729006	34.09	3600	1756262	2724970	35.55	3600
1_1_cap_c_75_100	1980595	2129518	7.0	3600	1993899	2129518	**6.0**	3600	1398511	2277899	38.61	3600	1269224	2168427	41.47	3600
2_1_cap_a_75_100	3191824	3636633	**12.0**	3600	3187720	3635804	**12.0**	3600	2706983	3825620	29.24	3600	2579111	3828634	32.64	3600
2_1_cap_b_75_100	2107204	2380141	11.0	3600	2197128	2380141	**8.0**	3600	1614282	2500412	35.44	3600	1510382	2538716	40.51	3600
2_1_cap_c_75_100	1852112	2034826	9.0	3600	1901999	2034826	**7.0**	3600	1362057	2126553	35.95	3600	1139131	2111054	46.04	3600
3_1_cap_a_75_100	2879844	3310761	**13.0**	3600	2873495	3296504	**13.0**	3600	2489581	3497307	28.81	3600	2257409	3377220	33.16	3600
3_1_cap_b_75_100	1997351	2245992	11.0	3600	2037468	2264007	**10.0**	3600	1539042	2363990	34.9	3600	1368379	2364348	42.12	3600
3_1_cap_c_75_100	1771449	1977057	**10.0**	3600	1766569	2021212	13.0	3600	1347385	2046889	34.17	3600	1042658	2031219	48.67	3600
4_1_cap_a_75_100	2701430	3147340	14.0	3600	2708148	3125374	**13.0**	3600	2346702	3199998	26.67	3600	2034703	3218198	36.78	3600
4_1_cap_b_75_100	1938058	2124533	**9.0**	3600	1932153	2124533	**9.0**	3600	1523159	2281982	33.25	3600	1259087	2213357	43.11	3600
4_1_cap_c_75_100	1752769	1938121	10.0	3600	1760485	1938121	**9.0**	3600	1363642	1987178	31.38	3600	972450	1964146	50.49	3600

Table D.4: Computational results for instances of size 300.

	Gurobi				*Gurobi*$_{\text{PPC}}$				B&B				B&B$_{\text{Flow}}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time
0_0_300	15256	22539	32.0	3600	20416	21601	**5.0**	3600	14204	23628	39.88	3600	14341	23160	38.08	3600
1_0_300	infeasible	infeasible	infeasible	228.83	infeasible	infeasible	infeasible	3.16	infeasible	infeasible	infeasible	5.52	infeasible	infeasible	infeasible	5.74
2_0_300	14503	22381	35.0	3600	21700	21700	**0.0**	632	16200	24346	33.46	3600	16403	24346	32.62	3600
3_0_300	17354	31441	45.0	3600	27783	27784	**0.0**	1134	20913	30347	31.09	3600	21120	30347	30.4	3600
4_0_300	infeasible	infeasible	infeasible	280.89	infeasible	infeasible	infeasible	3.82	infeasible	infeasible	infeasible	5.51	infeasible	infeasible	infeasible	6.02
5_0_300	6136	7865	22.0	3600	6661	7649	**13.0**	3600	5849	7948	26.41	3600	5896	7948	25.82	3600
6_0_300	6181	8876	30.0	3600	6788	8279	**18.0**	3600	5818	8656	32.79	3600	5884	8656	32.03	3600
7_0_300	5983	8068	26.0	3600	6542	7757	**16.0**	3600	5732	8112	29.33	3600	5837	8727	33.12	3600
8_0_300	5387	7997	33.0	3600	6135	7116	**14.0**	3600	5138	7528	31.75	3600	5207	7528	30.83	3600
9_0_300	5767	10165	43.0	3600	6480	8130	**20.0**	3600	5098	8796	42.04	3600	5187	9144	43.27	3600
10_0_300	3590	5399	34.0	3600	3816	4573	**17.0**	3600	3314	4986	33.53	3600	3375	4664	27.65	3600
11_0_300	3297	4358	24.0	3600	3393	4308	**21.0**	3600	2991	4425	32.4	3600	3060	4425	30.84	3600
12_0_300	4125	5542	26.0	3600	4290	5529	**22.0**	3600	3664	6052	39.46	3600	3720	6059	38.6	3600
13_0_300	3774	5146	27.0	3600	3949	5066	**22.0**	3600	3209	5424	40.83	3600	3243	5990	45.86	3600
14_0_300	3846	5141	25.0	3600	4070	4912	**17.0**	3600	3399	5165	34.18	3600	3444	5217	33.99	3600
15_0_300	2524	2840	11.0	3600	2840	2840	**0.0**	2640	2256	2973	24.1	3600	2266	2973	23.77	3600
16_0_300	2541	2970	14.0	3600	2601	2970	**12.0**	3600	2273	3086	26.34	3600	2295	3301	30.47	3600
17_0_300	2926	3531	17.0	3600	3040	3560	**15.0**	3600	2692	3582	24.84	3600	2706	3582	24.46	3600
18_0_300	2794	3260	14.0	3600	2855	3260	**12.0**	3600	2574	3381	23.88	3600	2616	3414	23.37	3600
19_0_300	2552	3022	16.0	3600	2601	3022	**14.0**	3600	2299	3171	27.48	3600	2307	3096	25.47	3600
0_1_300	15325	19725	22.0	3600	15283	19419	**21.0**	3600	12317	21261	42.07	3600	12372	21261	41.81	3600
1_1_300	14138	19855	29.0	3600	14241	19180	**26.0**	3600	11159	20301	45.03	3600	11215	20837	46.17	3600
2_1_300	13515	18854	28.0	3600	13720	18522	**26.0**	3600	11150	19999	44.25	3600	11202	19999	43.99	3600
3_1_300	17823	29560	40.0	3600	17998	27920	**36.0**	3600	13975	29042	51.88	3600	14062	30143	53.35	3600
4_1_300	16237	23063	30.0	3600	16266	22155	**27.0**	3600	13191	22813	42.18	3600	13274	22955	42.17	3600
5_1_300	6686	8441	**21.0**	3600	6728	8485	**21.0**	3600	5727	9275	38.24	3600	5803	8910	34.87	3600
6_1_300	6906	8564	**19.0**	3600	6838	8649	21.0	3600	5784	9201	37.14	3600	5859	8825	33.6	3600
7_1_300	6459	8061	20.0	3600	6520	8036	**19.0**	3600	5426	8439	35.71	3600	5580	8304	32.8	3600
8_1_300	5968	7587	21.0	3600	6031	7449	**19.0**	3600	4925	8176	39.76	3600	5086	8113	37.31	3600
9_1_300	6299	8030	**22.0**	3600	6308	8239	23.0	3600	4941	8274	40.28	3600	5030	8274	39.21	3600
10_1_300	4027	4928	18.0	3600	4026	4878	**17.0**	3600	3200	4967	35.57	3600	3278	4930	33.51	3600
11_1_300	3844	4620	17.0	3600	3900	4528	**14.0**	3600	3065	4667	34.33	3600	3084	4942	37.59	3600
12_1_300	4524	6065	25.0	3600	4581	5869	**22.0**	3600	3625	6058	40.16	3600	3699	5963	37.97	3600
13_1_300	4168	5201	20.0	3600	4190	5108	**18.0**	3600	3200	5401	40.74	3600	3381	5231	35.36	3600
14_1_300	4237	5049	16.0	3600	4237	4927	**14.0**	3600	3377	5081	33.53	3600	3412	5037	32.25	3600
15_1_300	2936	3419	14.0	3600	3005	3419	**12.0**	3600	2271	3419	33.58	3600	2280	3419	33.3	3600
16_1_300	3001	3632	**17.0**	3600	3003	3752	20.0	3600	2349	3723	36.89	3600	2363	3723	36.52	3600
17_1_300	3454	4103	**16.0**	3600	3420	4103	17.0	3600	2817	4293	34.37	3600	2893	4441	34.84	3600
18_1_300	3303	3850	**14.0**	3600	3261	3837	15.0	3600	2686	4061	33.85	3600	2756	4061	32.13	3600
19_1_300	2980	3557	**16.0**	3600	2983	3567	**16.0**	3600	2270	3903	41.85	3600	2278	3903	41.64	3600

Table D.5: Computational results for instances of size 500. Note, a missing lower bound implies that *Gurobi* failed in the root node.

	Gurobi				*Gurobi*$_{\text{PPC}}$				B&B				B&B$_{\text{Flow}}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time
0_0_500	–	–	–	–	40369	42686	**5.0**	3600	27613	46325	40.39	3600	27968	46325	39.63	3600
1_0_500	26067	100729	74.0	3600	37766	40732	**7.0**	3600	26777	45709	41.42	3600	26917	45950	41.42	3600
2_0_500	26121	102571	75.0	3600	36448	40215	**9.0**	3600	25288	43472	41.83	3600	25461	43569	41.56	3600
3_0_500	infeasible	infeasible	infeasible	1316.48	infeasible	infeasible	infeasible	15.35	infeasible	infeasible	infeasible	29.75	infeasible	infeasible	infeasible	29.76
4_0_500	infeasible	infeasible	infeasible	1079.97	infeasible	infeasible	infeasible	15.5	infeasible	infeasible	infeasible	30.57	infeasible	infeasible	infeasible	36.44
5_0_500	8776	14063	38.0	3424	9398	12222	**23.0**	3600	8093	12490	35.2	3600	8134	13051	37.67	3600
6_0_500	9697	17399	44.0	2800	10137	13752	**26.0**	3600	8652	15642	44.69	3600	8707	16141	46.06	3600
7_0_500	9163	15839	42.0	2385	9749	12872	**24.0**	3600	8460	13734	38.4	3600	8466	14129	40.08	3600
8_0_500	9589	16990	44.0	3173	10221	13243	**23.0**	3600	8611	13858	37.86	3600	8620	15414	44.07	3600
9_0_500	8877	12998	32.0	3600	9288	12029	**23.0**	3600	8054	12133	33.62	3600	8158	12525	34.87	3600
10_0_500	–	–	–	–	5483	7521	**27.0**	3600	4947	7725	35.95	3600	4975	8061	38.28	3600
11_0_500	–	–	–	–	6269	8579	**27.0**	3600	5665	8585	34.0	3600	5701	9185	37.92	3600
12_0_500	–	–	–	–	5957	7779	**23.0**	3600	5378	8055	33.23	3600	5389	8054	33.09	3600
13_0_500	–	–	–	–	5471	6891	**21.0**	3600	5025	7348	31.61	3600	5026	7230	30.49	3600
14_0_500	–	–	–	–	6049	8427	**28.0**	3600	5046	8569	41.11	3600	5078	8716	41.73	3600
15_0_500	–	–	–	–	3911	4774	**18.0**	3600	3616	4869	25.73	3600	3646	5279	30.92	3600
16_0_500	–	–	–	–	4288	5747	**25.0**	3600	3888	5679	31.54	3600	3892	6008	35.22	3600
17_0_500	–	–	–	–	4371	5329	**18.0**	3600	3987	5913	32.58	3600	4001	5914	32.34	3600
18_0_500	–	–	–	–	4471	5578	**20.0**	3600	4086	6064	32.62	3600	4125	6064	31.98	3600
19_0_500	–	–	–	–	3728	4556	**18.0**	3600	3369	4869	30.81	3600	3385	5035	32.78	3600
0_1_500	24179	42478	43.0	3600	24761	35931	**31.0**	3600	19588	40298	51.39	3600	19639	41929	53.16	3600
1_1_500	26196	43323	40.0	3600	26688	38617	**31.0**	3600	21662	42889	49.49	3600	21734	42889	49.32	3600
2_1_500	26081	42048	38.0	2729	26762	38279	**30.0**	3600	21974	40611	45.89	3600	22000	40611	45.83	3600
3_1_500	26394	46224	43.0	3600	27215	41799	**35.0**	3600	21422	42198	49.23	3600	21452	42964	50.07	3600
4_1_500	24039	39160	39.0	3600	24612	36668	**33.0**	3600	19028	37867	49.75	3600	19028	37867	49.75	3600
5_1_500	–	–	–	–	9587	14493	**34.0**	3600	7668	14323	46.46	3600	7668	14609	47.51	3600
6_1_500	–	–	–	–	10541	14372	**27.0**	3600	8667	14942	41.99	3600	8706	15356	43.3	3600
7_1_500	–	–	–	–	10012	13232	**24.0**	3600	8338	13603	38.7	3600	8339	13605	38.71	3600
8_1_500	–	–	–	–	10515	15336	**31.0**	3600	8565	16052	46.64	3600	8571	15745	45.56	3600
9_1_500	–	–	–	–	9769	14033	**30.0**	3600	7977	14236	43.96	3600	7977	14586	45.31	3600
10_1_500	–	–	–	–	6323	11288	44.0	3600	5014	8632	**41.91**	3600	5014	8658	42.09	3600
11_1_500	–	–	–	–	7020	9015	**22.0**	3600	5630	9019	37.57	3600	5634	9020	37.54	3600
12_1_500	–	–	–	–	6782	9490	**29.0**	3600	5416	9300	41.76	3600	5416	9409	42.44	3600
13_1_500	–	–	–	–	6395	88786	93.0	3600	5103	8423	**39.41**	3600	5104	8460	39.66	3600
14_1_500	–	–	–	–	–	80689	100	3600	5075	8437	39.85	3600	5081	8437	**39.77**	3600
15_1_500	–	–	–	–	4671	86835	95.0	3600	3587	6078	**40.98**	3600	3587	6173	41.89	3600
16_1_500	–	–	–	–	5244	88545	94.0	3600	3930	7211	45.5	3600	3930	7058	**44.32**	3600
17_1_500	–	–	–	–	–	93873	100	3600	4017	6752	**40.5**	3600	4017	7457	46.12	3600
18_1_500	–	–	–	–	5328	89812	94.0	3600	4164	7463	**44.21**	3600	4164	7463	**44.21**	3600
19_1_500	–	–	–	–	4565	83621	95.0	3600	3410	6256	45.49	3600	3411	6256	**45.47**	3600

Table D.6: Computational results for instances of size 700. Note, a missing lower bound for *Gurobi* implies that it failed in the root node. Missing lower and upper bounds for the B&B imply that the computation failed due to running into the time limit during preprocessing.

	Gurobi				*Gurobi*$_{\text{PPC}}$				B&B				B&B$_{\text{Flow}}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time
0_0_700	–	–	–	–	53355	57384	**7.0**	3600	34389	64021	46.28	3600	35410	63775	44.48	3600
1_0_700	–	–	–	–	infeasible	infeasible	infeasible	50.73	infeasible	infeasible	infeasible	99.13	infeasible	infeasible	infeasible	94.82
2_0_700	–	–	–	–	50365	53752	**6.0**	3600	37559	60720	38.14	3600	37783	60720	37.77	3600
3_0_700	–	–	–	–	50968	55107	**8.0**	3600	36109	63351	43.0	3600	36453	63351	42.46	3600
4_0_700	–	–	–	–	infeasible	infeasible	infeasible	52.17	infeasible	infeasible	infeasible	91.83	infeasible	infeasible	infeasible	82.06
5_0_700	–	–	–	–	12199	18717	**35.0**	3600	10707	18453	41.97	3600	10726	18453	41.87	3600
6_0_700	–	–	–	–	13492	19032	**29.0**	3600	11594	19551	40.7	3600	11632	19467	40.24	3600
7_0_700	–	–	–	–	13573	19744	**31.0**	3600	11161	20756	46.23	3600	11412	21640	47.26	3600
8_0_700	–	–	–	–	13070	18942	**31.0**	3600	11087	19620	43.49	3600	11088	19953	44.43	3600
9_0_700	–	–	–	–	14939	20319	**26.0**	3600	12591	21052	40.19	3600	12723	21759	41.53	3600
10_0_700	–	–	–	–	7541	11255	33.0	3600	7039	10329	31.85	3600	7046	10329	**31.78**	3600
11_0_700	–	–	–	–	8364	13460	38.0	3600	7566	11965	**36.77**	3600	7566	12764	40.72	3600
12_0_700	–	–	–	–	7560	10716	**29.0**	3600	6654	11112	40.12	3600	6654	10933	39.14	3600
13_0_700	–	–	–	–	7587	10218	**26.0**	3600	7012	10396	32.55	3600	7012	10307	31.97	3600
14_0_700	–	–	–	–	8490	13206	**36.0**	3600	7420	11718	36.68	3600	7420	11991	38.12	3600
15_0_700	–	–	–	–	5155	8103	36.0	3600	4731	6518	**27.42**	3600	4731	6518	**27.42**	3600
16_0_700	–	–	–	–	5323	7666	**31.0**	3600	5065	7589	33.25	3600	5065	7589	33.25	3600
17_0_700	–	–	–	–	5768	107295	95.0	3600	5337	8081	33.96	3600	5346	8081	**33.84**	3600
18_0_700	–	–	–	–	–	107715	100	3600	4983	7503	33.59	3600	4984	7503	**33.57**	3600
19_0_700	–	–	–	–	–	110442	100	3600	4907	7034	**30.24**	3600	4907	7386	33.56	3600
0_1_700	–	–	–	–	36855	62203	**41.0**	3600	29647	62060	52.23	3600	29707	62476	52.45	3600
1_1_700	–	–	–	–	33420	63128	**47.0**	3600	26857	57379	53.19	3600	26857	62536	57.05	3600
2_1_700	–	–	–	–	33759	57754	**42.0**	3600	26705	57110	53.24	3600	26705	56820	53.0	3600
3_1_700	–	–	–	–	39275	63148	**38.0**	3600	30769	67458	54.39	3600	30769	69447	55.69	3600
4_1_700	–	–	–	–	36835	62039	**41.0**	3600	30280	60410	49.88	3600	30280	60091	49.61	3600
5_1_700	–	–	–	–	–	–	–	–	10822	20090	**46.13**	3600	10822	20228	46.5	3600
6_1_700	–	–	–	–	–	–	–	–	11639	21912	**46.88**	3600	11639	22882	49.13	3600
7_1_700	–	–	–	–	–	–	–	–	11136	19703	**43.48**	3600	11136	20515	45.72	3600
8_1_700	–	–	–	–	–	–	–	–	11186	19316	**42.09**	3600	11186	19316	**42.09**	3600
9_1_700	–	–	–	–	–	–	–	–	12674	23066	**45.05**	3600	12674	23536	46.15	3600
10_1_700	–	–	–	–	–	–	–	–	–	–	100	3600	7137	12921	**44.76**	3600
11_1_700	–	–	–	–	–	–	–	–	7646	12768	**40.11**	3600	7646	13681	44.11	3600
12_1_700	–	–	–	–	–	–	–	–	6711	11750	**42.88**	3600	6711	12429	46.0	3600
13_1_700	–	–	–	–	–	–	–	–	7094	12716	**44.21**	3600	7094	13583	47.77	3600
14_1_700	–	–	–	–	–	–	–	–	7480	12792	**41.52**	3600	7480	13695	45.38	3600
15_1_700	–	–	–	–	–	–	–	–	4774	9061	**47.3**	3600	4774	9688	50.72	3600
16_1_700	–	–	–	–	–	–	–	–	5137	9404	**45.37**	3600	5137	9786	47.5	3600
17_1_700	–	–	–	–	–	–	–	–	5420	9623	**43.68**	3600	5420	9623	**43.68**	3600
18_1_700	–	–	–	–	–	–	100	3600	5062	8636	**41.38**	3600	5062	8636	**41.38**	3600
19_1_700	–	–	–	–	–	–	–	–	4985	9098	**45.21**	3600	4985	9098	**45.21**	3600

Table D.7: Computational results for instances of size 1000. Missing lower and upper bounds imply that the computation failed due to running into the time limit during preprocessing.

	Gurobi				*Gurobi*$_{\text{PPC}}$				B&B				B&B$_{\text{Flow}}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time	LB	UB	Gap	time
0_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
1_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
2_0_1000	–	–	–	–	infeasible	infeasible	infeasible	154.36	infeasible	infeasible	infeasible	291.75	infeasible	infeasible	infeasible	291.57
3_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
4_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
5_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
6_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
7_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
8_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
9_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
10_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
11_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
12_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
13_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
14_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
15_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
16_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
17_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
18_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
19_0_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
0_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
1_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
2_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
3_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
4_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
5_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
6_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
7_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
8_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
9_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
10_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
11_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
12_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
13_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
14_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
15_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
16_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
17_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
18_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600
19_1_1000	–	–	–	–	–	–	100	3600	–	–	100	3600	–	–	100	3600

Acknowledgements

Felix Engelhardt was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the grant 2236/2. Sophia Wrede was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the grant 442047500 – Collaborative Research Center "Sparsity and Singular Structures" (SFB 1481). Computations were performed with computing resources granted by RWTH Aachen University.

We would like to thank our colleague Timo Gersing, who never minded listening to our complaints about debugging and dual bounds. We also thank the organising team of CO@Work 2024, i.e. Timo Berthold, Ralf Borndörfer, Ambros Gleixner, Thorsten Koch, and Milena Petkovic. Their workshop motivated and inspired us to do this research.

References

T. Achterberg. *Constraint Integer Programming*. PhD thesis, TU Berlin, 07 2007.

P. Avella and M. Boccia. A cutting plane algorithm for the capacitated facility location problem. *Computational Optimization and Applications*, 43:39–65, 2009. doi: 10.1007/s10589-007-9125-x. URL `https://doi.org/10.1007/s10589-007-9125-x`.

J. Beasley. An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research*, 33(3):314–325, 1988. doi: 10.1016/0377-2217(88)90175-0.

J. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65 (3):383–399, 1993. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(93)90118-7. URL `https://www.sciencedirect.com/science/article/pii/0377221793901187`.

T. Berthold. *Primal Heuristics for Mixed Integer Programs*. PhD thesis, TU Berlin, 01 2006.

T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41:611–614, 11 2013. doi: 10.1016/j.orl.2013.08.007.

C. Büsing, T. Gersing, and S. Wrede. Analysing the complexity of facility location problems with capacities, revenues, and closest assignments. *Proceedings of the 10th International Network Optimization Conference (INOC), Aachen, Germany, June 7–10, 2022*, pages 81–86, 2022. ISSN 2510-7437. doi: 10.48786/inoc.2022.15. URL `https://openproceedings.org/2022/conf/inoc/INOC_2022_paper_22.pdf`.

C. Büsing, T. Gersing, and S. Wrede. Insights into the computational complexity of the single-source capacitated facility location problem with customer preferences. *Optimization Online*, 2025. URL `https://optimization-online.org/?p=28912`.

C. Büsing, M. Leitner, and S. Wrede. Cover-based inequalities for the single-source capacitated facility location problem with customer preferences. *Computers & Operations Research*, 182:107082, 2025. doi: 10.1016/j.cor.2025.107082.

X. Cabezas and S. García. A semi-lagrangian relaxation heuristic algorithm for the simple plant location problem with order. *Journal of the Operational Research Society*, 0(0):1–12, 2022. doi: 10.1080/01605682.2022.2150573. URL `https://doi.org/10.1080/01605682.2022.2150573`.

H. Calvete, C. Galé, J. Iranzo, J. Camacho-Vallejo, and M. Casas-Ramírez. A matheuristic for solving the bilevel approach of the facility location problem with cardinality constraints and preferences. *Computers & Operations Research*, 124:105066, 2020. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2020.105066. URL `https://www.sciencedirect.com/science/article/pii/S0305054820301830`.

Table D.8: Computational results for instances of size 1000 after 24 hours.

	B&B				$B\&B_{Flow}$			
Instance	LB	UB	Gap	time	LB	UB	Gap	time
0_0_1000	47626	90786	**47.54**	28314	47626	90786	**47.54**	21755
1_0_1000	50935	90910	**43.97**	26309	50937	90910	**43.97**	20596
2_0_1000	infeasible	infeasible	infeasible	291.75	infeasible	infeasible	infeasible	291.57
3_0_1000	45329	84506	**46.36**	23182	45332	84898	46.6	18421
4_0_1000	48688	89723	**45.73**	23321	48700	90056	45.92	18503
5_0_1000	17671	28690	**38.41**	86400	17746	29979	40.81	86400
6_0_1000	16233	27104	**40.11**	86400	16262	27410	40.67	86400
7_0_1000	16734	29934	**44.1**	86400	16813	31148	46.02	86400
8_0_1000	15938	27665	**42.39**	86400	15995	27871	42.61	86400
9_0_1000	16662	28110	**40.73**	86400	16684	28344	41.14	86400
10_0_1000	9924	15254	**34.94**	86400	9975	16028	37.77	86400
11_0_1000	10250	15091	**32.08**	86400	10279	15738	34.68	86400
12_0_1000	9996	15831	**36.85**	86400	10058	16161	37.76	86400
13_0_1000	10243	16146	**36.56**	86400	10280	16713	38.49	86400
14_0_1000	10423	16234	35.79	86400	10438	16174	**35.46**	86400
15_0_1000	7707	11816	**34.77**	86400	–	–	100	86400
16_0_1000	7586	11273	**32.71**	86400	7591	11421	33.54	86400
17_0_1000	7542	12037	37.34	86400	7577	11960	**36.64**	86400
18_0_1000	7302	10158	**28.12**	86400	7303	11423	36.07	86400
19_0_1000	7528	11174	32.62	86400	7556	11174	**32.37**	86400
0_1_1000	40618	79752	**49.07**	86400	40620	84792	52.09	86400
1_1_1000	41640	85581	51.34	86400	41655	85581	**51.33**	86400
2_1_1000	38001	78470	51.57	86400	38011	78010	**51.27**	86400
3_1_1000	40603	80460	**49.54**	86400	40611	85774	52.65	86400
4_1_1000	41733	78974	**47.16**	86400	41771	79887	47.71	86400
5_1_1000	16768	30095	**44.28**	86400	16879	30675	44.97	86400
6_1_1000	16171	27613	41.44	86400	16211	27613	**41.29**	86400
7_1_1000	16367	28983	43.53	86400	16447	29057	**43.4**	86400
8_1_1000	15711	26196	**40.02**	86400	15729	27209	42.19	86400
9_1_1000	16491	27881	40.85	86400	16624	27756	**40.1**	86400
10_1_1000	10007	18525	45.98	86400	10045	18510	**45.73**	86400
11_1_1000	10426	18390	**43.31**	86400	10426	18531	43.74	86400
12_1_1000	9904	16518	**40.04**	86400	9904	16686	40.64	86400
13_1_1000	10318	18536	**44.33**	86400	10318	18536	**44.33**	86400
14_1_1000	10480	17660	**40.65**	86400	10480	17660	**40.65**	86400
15_1_1000	7789	13949	44.16	86400	7794	13668	**42.98**	86400
16_1_1000	7672	13579	**43.5**	86400	7672	14040	45.36	86400
17_1_1000	7601	14021	45.78	86400	7601	13938	**45.46**	86400
18_1_1000	7367	13334	**44.75**	86400	7367	13608	45.86	86400
19_1_1000	7578	13434	43.59	86400	7578	12843	**40.99**	86400

L. Cánovas, S. García, M. Labbé, and A. Marín. A strengthened formulation for the simple plant location problem with order. *Operations Research Letters*, 35(2):141–150, 2007. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2006.01.012. URL https://www.sciencedirect.com/science/article/pii/S0167637706000320.

M. Casas-Ramírez, J. Camacho-Vallejo, and I. Martínez-Salazar. Approximating solutions to a bilevel capacitated facility location problem with customer's patronization toward a list of preferences. *Applied Mathematics and Computation*, 319:369–386, 2018. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2017.03.051. URL https://www.sciencedirect.com/science/article/pii/S0096300317302357. Recent Advances in Computing.

D. Celik Turkoglu and M. Erol Genevois. A comparative survey of service facility location problems. *Annals of Operations Research*, 292:399–468, 2020. doi: 10.1007/s10479-019-03385-x. URL https://doi.org/10.1007/s10479-019-03385-x.

G. Cornuejols, R. Sridharan, and J. Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research*, 50(3):280–297, 1991.

C. Domínguez and J. de Dios Jaime-Alcántara. Stable formulations for the capacitated facility location problem with customer preferences. *arXiv preprint*, 2025. doi: 10.48550/arXiv.2507.21944.

F. Engelhardt and S. Wrede. Code for "A Combinatorial Branch-and-Bound Algorithm for the Capacitated Facility Location Problem under Strict Customer Preferences", Nov. 2025. URL https://doi.org/10.5281/zenodo.17524471.

I. Espejo, A. Marín, and A. M. Rodríguez-Chía. Closest assignment constraints in discrete location problems. *European Journal of Operational Research*, 219(1):49–58, 2012. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.12.002. URL https://www.sciencedirect.com/science/article/pii/S0377221711010575.

M. L. Fisher. An applications oriented guide to lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985. URL http://www.jstor.org/stable/25060666.

M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12):1861–1871, 2004. doi: 10.1287/mnsc.1040.0263.

G. Gamrath. Improving strong branching by propagation. *EURO Journal on Computational Optimization*, 2:99–122, 04 2014. doi: 10.1007/s13675-014-0021-8.

A. Geoffrion and R. M. Bride. Lagrangean relaxation applied to capacitated facility location problems. *A I I E Transactions*, 1978. doi: 10.1080/05695557808975181.

S. Görtz and A. Klose. A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. *INFORMS Journal on Computing*, 24(4):597–610, 2012. doi: 10.1287/ijoc.1110.0468. URL https://doi.org/10.1287/ijoc.1110.0468.

E. D. Güneş, T. Melo, and S. Nickel. *Location Problems in Healthcare*, pages 657–686. Springer International Publishing, 2019. ISBN 978-3-030-32177-2. doi: 10.1007/978-3-030-32177-2_23. URL https://doi.org/10.1007/978-3-030-32177-2_23.

Gurobi, 2025. URL https://www.gurobi.com/.

A. A. Hagberg, D. A. Schult, and S. P. J. Exploring network structure, dynamics, and function using networkx. In *7th Python in Science Conference (SciPy2008)*, pages 11–15, 2008.

P. Hanjoul and D. Peeters. A facility location problem with clients' preference orderings. *Regional Science and Urban Economics*, 17(3):451–473, 1987. ISSN 0166-0462. doi: https://doi.org/10.1016/0166-0462(87)90011-1. URL https://www.sciencedirect.com/science/article/pii/0166046287900111.

K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(98)00008-3. URL https://www.sciencedirect.com/science/article/pii/S0377221798000083.

C.-N. Kang, L.-C. Kung, P.-H. Chiang, and J.-Y. Yu. A service facility location problem considering customer preference and facility capacity. *Computers & Industrial Engineering*, 177:109070, 2023. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2023.109070. URL https://www.sciencedirect.com/science/article/pii/S0360835223000943.

G. Laporte, S. Nickel, and F. Saldanha da Gama, editors. *Location Science*. Springer, 2019. ISBN 978-3-030-32177-2. doi: https://doi.org/10.1007/978-3-030-32177-2.

D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2016.01.005. URL https://www.sciencedirect.com/science/article/pii/S1572528616000062.

R. M. Nauss. An improved algorithm for the capacitated facility location problem. *Journal of the Operational Research Society*, 1978. doi: 10.1057/jors.1978.263.

S. Polino, J.-F. Camacho-Vallejo, and J. G. Villegas. A facility location problem for extracurricular workshop planning: bi-level model and metaheuristics. *International Transactions in Operational Research*, 0:1–43, 2023. doi: https://doi.org/10.1111/itor.13368. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.13368.

SCIP Optimization Suite, 2025. URL https://www.scipopt.org/.

Università degli Studi di Brescia. https://or-brescia.unibs.it/instances/instances_sscflp, 2025. Accessed: 2025-08-20.

I. Vasilyev, X. Klimentova, and M. Boccia. Polyhedral study of simple plant location problem with order. *Oper. Res. Lett.*, 41:153–158, 2013. doi: https://doi.org/10.1016/j.orl.2012.12.006.

J. L. Wagner and L. M. Falkson. The optimal nodal location of public facilities with price-sensitive demand. *Geographical Analysis*, 7(1):69–83, 1975. doi: https://doi.org/10.1111/j.1538-4632.1975.tb01024.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.1975.tb01024.x.