manuscript No.
(will be inserted by the editor)

The Balanced Facility Location Problem: Complexity and Heuristics

Malena Schmidt · Bismark Singh

Received: date / Accepted: date

Abstract In recent work, Schmitt and Singh [22] propose a new quadratic facility location model to address ecological challenges faced by policymakers in Bavaria, Germany. Building on this previous work, we significantly extend our understanding of this new problem. We develop connections to traditional combinatorial optimization models and show the problem is $\mathcal{NP}$-hard. We then develop several classes of easy-to-implement heuristics to solve this problem. These are rooted in solving special cases of the generalized quadratic assignment problem as a subproblem; this subproblem is also $\mathcal{NP}$-hard. On moderate sized instances from Bavaria—that were previously intractable—our proposed heuristics compute feasible solutions that are 0.5% (on average) improved over the naive solution method in just over a minute (on average), even when the naive solver runs for 20,000 seconds. Larger instances show an improvement of 5% (on average) compared to the naive solution method in an average of 410 seconds.

Keywords facility location problem · heuristics · quadratic assignment · complexity · waste management

Mathematics Subject Classification (2000) 90C27 · 90C59

1 Introduction

We revisit a recently proposed model of the undesirable facility location problem (FLP) that seeks to assign users to so-called obnoxious facilities in a fair manner [22]. Here, the authors develop a framework for systematic closures of facilities—such as, airports, recycling centers, and landfills—which are necessary for public use yet exert a negative impact when used in an imbalanced manner. Their specific motivation is the pervasive shutdown of recycling centers in the German state of Bavaria over the last two decades due the damaging ecological impact by operation of these facilities, see, e.g., [2]. The authors formulate this relevant and timely problem as a quadratic optimization model with binary variables thereby distinguishing it from most other FLPs that include a linear objective function. This model seeks to assist policymakers by maintaining a balanced usage of existing recycling centers while still ensuring

Malena Schmidt
Delft University of Technology
2628 CT, Delft, Netherlands

Bismark Singh, Corresponding author
University of Southampton
SO17 1BJ, Southampton, United Kingdom
B.Singh@southampton.ac.uk

high levels of access for the populations visiting them. Our work significantly extends this new body of literature in ways we describe below.

Due to the computational intractability of the model proposed in [22], the authors are unable to solve it for all the considered users and facilities in Bavaria. Thus, they resort to a number of ad-hoc computational enhancements, e.g., relaxations of equality constraints to inequalities or ignoring users far away from facilities. Although such enhancements provide practically viable solutions, a formal understanding of this new problem's structure is left unattended. As a result, extending the problem from Bavaria to all of Germany is practically untenable: the model generation alone takes several hours even on a high performance computer. Our work seeks to fill this gap by systematically investigating this recent problem from both a theoretical and a computational perspective. To this end, we study the complexity of this problem (and, an associated subproblem) as well as the relationship with a number of other combinatorial optimization models. These relationships lead us to develop several classes of heuristics tractable for problem sizes an order of magnitude larger than those previously considered. Specifically, we seek to show how relatively simple heuristics, including those rooted in traditional procedures, when implemented smartly obviate the need for more sophisticated schemes.

Employing the same notation as that in [22], we consider a set of users $i \in I$ with populations $U_i > 0$, a set of facilities $j \in J$ with capacities $C_j > 0$, preferences of users to facilities $0 < P_{ij} < 1$, and a budget $B \leq |J|$ of the number of open facilities. Then, the following is the central optimization model proposed in [22]:

$$
\begin{aligned}
z^* = \min_{x,y} \ & \sum_{j \in J} C_j \left(1 - \frac{\sum_{i \in I} U_i P_{ij} x_{ij}}{C_j}\right)^2 && && \text{(1a)}\\
\text{s.t.} \ & \sum_{j \in J} y_j \leq B && && \text{(1b)}\\
& \sum_{i \in I} U_i P_{ij} x_{ij} \leq C_j && \forall j \in J && \text{(1c)}\\
& \sum_{j \in J} x_{ij} = 1 && \forall i \in I && \text{(1d)}\\
& x_{ij} \leq y_j && \forall i \in I, j \in J && \text{(1e)}\\
& y_j \in [0,1] && \forall j \in J && \text{(1f)}\\
& x_{ij} \in \{0,1\} && \forall i \in I, j \in J. && \text{(1g)}
\end{aligned}
$$

In model (1), the decision variables y_j and x_{ij} indicate whether facility j is opened, and whether user i is assigned to facility j, respectively; the binary restrictions in constraints (1g) with the model's structure ensures an optimal solution has binary values for the y variables in constraint (1f) as well. Then, constraint (1e) ensures that users are only assigned to open facilities. The quantity $W_{ij} = U_i P_{ij}$ is interpreted as the discounted population of user i actually visiting facility j if i is assigned to j. Constraint (1c) restricts the number of assigned users to a facility by its capacity, while constraint (1b) bounds the number of open facilities by the parameter B. Constraint (1d) and (1g) jointly ensure that every user is assigned to exactly one facility. The key novelty in model (1) lies in the objective function (1a) that seeks to ensure a low variance in the utilization of the facilities (given by $u_j = \frac{\sum_{i \in I} U_i P_{ij} x_{ij}}{C_j}, \forall j \in J$) while simultaneously maximizing overall access of users to the recycling facilities (given by $\frac{\sum_{j \in J, i \in I} U_i P_{ij} x_{ij}}{\sum_{i \in I} U_i}$). The structure of this optimization model, especially as driven by the objective function, provides several interesting properties that we investigate in this work. We study this model as presented in this form; for further details on the model including the choice of this particular objective function, see [22]. We name the problem defined by model (1) as the Balanced Facility Location Problem (BFLP), and state its decision version in Section 2.

The feasible region of model (1), defined by constraints (1b)-(1g), is closely related to several classes of traditional FLPs. As an example, consider the capacitated FLP (CFLP) defined as follows:

$$\min_{x,y} \sum_{i\in I}\sum_{j\in J} e_{ij} d_i x_{ij} + \sum_{j\in J} f_j y_j \tag{2a}$$

$$\text{s.t. } \sum_{i\in I} d_i x_{ij} \le C_j y_j \qquad \forall j \in J \tag{2b}$$

$$\sum_{j\in J} x_{ij} = 1 \qquad \forall i \in I \tag{2c}$$

$$y_j \in \{0,1\} \qquad \forall j \in J \tag{2d}$$

$$x_{ij} \in [0,1] \qquad \forall i \in I, j \in J. \tag{2e}$$

The CFLP and BFLP differ foremost in the form of their objective functions due to the former's consideration of costs: the CFLP minimizes the cost of operating facilities, f, and transporting users to facilities, e. Another difference is in the parameter d_i that denotes demands (or equivalently, fulfillment levels) of user i. This is analogous to the parameter $U_i P_{ij}$ of the BFLP but does not consider preferences of users to facilities; i.e., the BFLP enriches the CFLP by allowing different levels of demand satisfaction based on preferred facilities. Model (7) of [22] presents a special case of the BFLP that removes these preferences by setting $P_{ij} \leftarrow P_i, \forall i \in I, j \in J$; then, the authors prove that an optimal solution for this model has exactly the same level of utilization for all the open facilities. This result is related to the classical notion of proportional fairness, see, e.g., [9]. The third difference is that the BFLP includes a budget on the number of open facilities, B. This imposes an additional combinatorial layer over the constraints of the CFLP as the BFLP selects at most B of the $|J|$ facilities to provide an assignment of users. Including this constraint makes the feasible region of model (1) comparable to the traditional p-median problem whose feasible region is modeled by constraints (1b) (with an equality), (1d)-(1g). Finally, the x variables are fractional in the CFLP unlike those in the BFLP. Although the BFLP allows a continuous relaxation of the binary restrictions on the y variables without loss of optimality, it does not allow the same for the binary restrictions on the x variables; this is again due to the structure of the objective function (1a). This fact is unlike the p-median problem that does allow a continuous relaxation of the x variables without loss of optimality.

Most variants of FLPs are $\mathcal{NP}$-hard, see, e.g., [10]; thus, a variety of heuristics are available for their solution. In this work, we choose the well-studied ADD and DROP heuristics, and adapt these for the BFLP. These heuristics were originally presented in [11] and [6], respectively, and have been extensively studied over the last few decades, see, e.g., [7, 23]. The key idea of these heuristics is to sequentially add or drop particular facilities, from a given set, that are determined as good to open or close by another procedure. Typically, this second procedure involves computing an assignment of users to a new but known set, $S \subseteq J$, of facilities; i.e., the procedure involves solving a generalized assignment problem (GAP). The GAP has a feasible region given by constraints (2b)-(2e) with the y variables fixed; i.e., $y_j = 1, \forall j \in S$ and $y_j = 0, \forall j \in J \setminus S$. Thus, the feasible region is $\left\{x : \sum_{i\in I} d_i x_{ij} \le C_j, \forall j \in S; \sum_{j\in S} x_{ij} = 1, \forall i \in I; x_{ij} \in \{0,1\}, \forall i \in I, j \in S\right\}$. Finding a feasible solution to this region is known to be $\mathcal{NP}$-complete [13].

The above-mentioned schemes decompose a hard problem into subproblems that are also, unfortunately, hard; however, solving the latter problem is typically easier than the former. If the subproblem is also computationally intractable, as we find in our experiments, tailored solution methods are used for its solution. For example, to solve the hard GAP subproblem several polynomial time approximation algorithms are available [17]. Further, local search approaches based on so-called λ-moves that reassign at most λ users between facilities are also frequently employed to solve the GAP, see, e.g., [14,15]. The heuristics we develop to solve the BFLP are in a similar spirit. In Section 2, we show that model (1) is $\mathcal{NP}$-hard. This observation is not surprising, given that the computational results of the original work in [22] demonstrate intractability to solve the model naively. Next, we develop schemes to smartly fix a set of facilities as open. The arising subproblem is then a special case of the Generalized Quadratic Assignment Problem (GQAP) instead of the GAP due to the structure of the objective function. Despite the special case, this subproblem is still $\mathcal{NP}$-hard as we show in Section 2. However, unlike the GAP, this subproblem is computationally intractable even for moderate sized instances; thus, in Section 3, we derive an additional set of heuristics for its solution. This subproblem of the BFLP for a *fixed* set of open facilities, $S \subseteq J$ (where, $|S| \leq B$) is as follows.

$$z^*_S = \sum_{j \in J \setminus S} C_j + \min_x \sum_{j \in S} C_j \big(1 - \frac{\sum_{i \in I} U_i P_{ij} x_{ij}}{C_j}\big)^2 \tag{3a}$$

$$\text{s.t. } (1c), (1d), (1g). \tag{3b}$$

The continuous relaxation of model (3) (we revisit this later in model (9)) is a convex optimization model, see, Proposition S1 in Appendix A; thus, a standard mixed-integer-programming (MIP) solver, such as Gurobi, is sufficient to solve model (9) to optimality [24]. We name the problem defined by model (3) as the Balanced User Assignment Problem (BUAP), and again state its decision version in Section 2. A feasible solution for model (3) extends to model (1) by setting $y_j = 1, \forall j \in S; y_j = 0, \forall j \in J \setminus S$; further, it follows from Proposition S3 in Appendix A that $z^*_S \geq z^*, \forall S \subseteq J$, where $|S| \leq B$).
With this background, the following are the key contributions of this work:

(i) we formally define the BFLP and the BUAP and show both are $\mathcal{NP}$-hard;
(ii) we extend and compare several heuristic methods available for FLPs to both these problems;
(iii) we provide extensive computational experiments on two real-world case studies from Bavaria as well as two artificially generated instances from Germany to guide public policymakers facing similar problems;
(iv) We publicly release all our code and instances to allow further developments to this new problem, as well as related FLPs.

The rest of this article is structured as follows. In Section 2, we define the decision versions of our two problems and study their theoretical hardness. Motivated by this finding, we then derive heuristics for both these problems. Section 3 presents two algorithms, plus local search enhancements for each, to solve model (3). In Section 4, we employ these heuristics as procedures and develop three algorithms to solve model (1). In Section 5, we present the original data we use as well as new data we synthesize for our computational experiments that we report in Section 6. These computations compare the heuristics to each other and also to a naive solution of the underlying models. We conclude in Section 7 and provide further proofs, pseudocodes, examples, and analysis in the appendices. We provide all associated datasets and our code at the GitHub repository mentioned below.

2 Complexity

In this section, we study the theoretical complexity of the two problems central to this work mentioned in Section 1: the BFLP and the BUAP defined by model (1) and model (3), respectively. Without loss of generality, we drop the constant term in the objective function of model (3) in this section. We begin by defining their decision versions.

Definition 2.1 The Balanced User Assignment Problem (`BUAP`)
Instance: Given a set of users $i \in I = \{i_1, i_2, \ldots, i_{|I|}\}$, $|I| \geq 2$, a set of facilities $j \in S = \{j_1, j_2, \ldots, j_{|S|}\}$ with corresponding capacities $C_j \in \mathcal{Z}^+$, weights $W_{ij} \in \mathcal{R}^+$ of assigning user i to facility j and a number $M \in \mathcal{R}^+$.

Question: Do there exist subsets of users $I_j \subseteq I, \forall j \in S$ such that the I_j form a partition of I (i.e., $\bigcup_{j \in S} I_j = I$ and $I_j \cap I_{j'} = \emptyset, \forall j \neq j' \in S$) with $\sum_{i \in I_j} W_{ij} \leq C_j, \forall j \in S$ such that $\sum_{j \in S} C_j \left(1 - \frac{\sum_{i \in I_j} W_{ij}}{C_j}\right)^2 \leq M$?

Definition 2.2 The Balanced Facility Location Problem (`BFLP`)
Instance: Given a set of users $i \in I = \{i_1, i_2, \ldots, i_{|I|}\}$, $|I| \geq 2$, a set of facilities $j \in J = \{j_1, j_2, \ldots, j_{|J|}\}$ with corresponding capacities $C_j \in \mathcal{Z}^+$, weights $W_{ij} \in \mathcal{R}^+$ of assigning user i to facility j, a budget $B \leq |J| \in \mathcal{Z}^+$ and a number $M \in \mathcal{R}^+$.

Question: Does there exist a subset of facilities $S \subseteq J$ with $|S| \leq B$ and subsets of users $I_j \subseteq I, \forall j \in S$ such that the I_j form a partition of I (i.e., $\bigcup_{j \in S} I_j = I$ and $I_j \cap I_{j'} = \emptyset, \forall j \neq j' \in S$) with $\sum_{i \in I_j} W_{ij} \leq C_j, \forall j \in S$ and $\sum_{j \in J} C_j \left(1 - \frac{\sum_{i \in I_j} W_{ij}}{C_j}\right)^2 \leq M$?

We next show that the `BUAP` is $\mathcal{NP}$-complete even for two facilities and even in the absence of an objective function; i.e., determining a feasible solution of model (3) itself is $\mathcal{NP}$-complete. To this end, we define the following problem—given by constraints (1c), (1d), and (1g)—that determines feasibility of model (3); we also use this problem in our algorithms in Section 3.

Definition 2.3 The User Assignment Problem (`UAP`)
Instance: Given a set of users $i \in I = \{i_1, i_2, \ldots, i_{|I|}\}$, $|I| \geq 2$, a set of facilities $j \in S = \{j_1, j_2, \ldots, j_{|S|}\}$ with corresponding capacities $C_j \in \mathcal{Z}^+$, and weights $W_{ij} \in \mathcal{R}^+$ of assigning user i to facility j.

Question: Do there exist subsets of users $I_j \subseteq I, \forall j \in S$ such that the I_j form a partition of I (i.e., $\bigcup_{j \in S} I_j = I$ and $I_j \cap I_{j'} = \emptyset \forall j \neq j' \in S$) with $\sum_{i \in I_j} W_{ij} \leq C_j, \forall j \in S$?

Lemma 2.1 *([13, Chapter 7]) The `UAP` is $\mathcal{NP}$-complete.*

We also provide a slightly different and detailed proof of Lemma 2.1 in Appendix A in Theorem 2.1. Although—as Lemma 2.1 shows—determining a feasible solution for model (3) is $\mathcal{NP}$-hard in general, solutions to special cases of instances of the `UAP` are easy. For example, relaxing constraint (1d) from an equality to an inequality renders the model always feasible ($x_{ij} = 0, \forall i \in I, j \in S$ is feasible). A more interesting special case occurs when the capacities are large enough to accommodate any user: $C_j \geq \sum_{i \in I} W_{ij}, \forall j \in S$; a trivial feasible solution is then obtained by assigning any user to any facility. Next, we show that although feasibility of the problem in this special case is easy, determining an optimal set of facilities is still $\mathcal{NP}$-hard. In Appendix A, we provide two examples, Example S1 and Example S2 for how other similarly intuitive solutions to this particular problem are suboptimal. We define the decision version of this special case of the `BUAP` as follows.

Definition 2.4 The Sufficient Capacity User Assignment Problem (`SCUAP`)
Instance: Given a set of users $i \in I = \{i_1, i_2, \ldots, i_{|I|}\}$, $|I| \geq 2$, a set of facilities $j \in S = \{j_1, j_2, \ldots, j_{|S|}\}$

with corresponding capacities $C_j \in \mathcal{Z}^+$, weights $W_{ij} \in \mathcal{R}^+$ of assigning user i to facility j s.t. $C_j \geq \sum_{i\in I} W_{ij}, \forall j \in S$ and a number $M \in \mathcal{R}^+$.

Question: Do there exist subsets of users $I_j \subseteq I, \forall j \in S$ such that the I_j form a partition of I (i.e., $\bigcup_{j\in S} I_j = I$ and $I_j \cap I_{j'} = \emptyset, \forall j \neq j' \in S$) with $\sum_{j\in S} C_j \left(1 - \frac{\sum_{i\in I_j} W_{ij}}{C_j}\right)^2 \leq M$?

Our proof rests on reduction from the partition problem, whose definition and complexity we state below, and a proposition whose proof being straightforward we reserve for A.

Definition 2.5 The Partition Problem (PP)

Instance: Given a set of positive integers $T = \{t_1, \ldots, t_{|T|}\}$, $|T| \geq 2$; i.e., $t_k \in \mathbb{Z}^+, \forall k = 1, 2, \ldots, |T|$.

Question: Does there exist a subset $L \subseteq \{1, \ldots, |T|\}$ such that $\sum_{k\in L} t_k = \frac{1}{2}\sum_{t\in T} t = \sum_{k\in\{1,\ldots,|T|\}\setminus L} t_k$?

Lemma 2.2 ([8]) *The partition problem is $\mathcal{NP}$-complete.*

Proposition 2.1 *Given $y \in \mathcal{R}^+$ the function $f(x_1, x_2) = (1 - \frac{x_1}{y})^2 + (1 - \frac{x_2}{y})^2$ subject to $x_1 + x_2 = y$, where $x_1, x_2 \in \mathcal{R}^+$, is uniquely minimized at $x_1 = \frac{1}{2}y = x_2$; hence, $f(\frac{1}{2}y, \frac{1}{2}y) = \frac{1}{2}$.*

Proof. See Appendix A. □

Theorem 2.1 *The SCUAP is $\mathcal{NP}$-complete.*

Proof. Given an instance of PP, we construct an instance of SCUAP as follows:

- $I \leftarrow \{1, \ldots, |T|\}$;
- $S \leftarrow \{1, 2\}$;
- $C_1 = C_2 \leftarrow \sum_{t\in T} t$;
- $W_{i1} = W_{i2} \leftarrow t_i \forall i \in I$;
- $M \leftarrow \frac{1}{2}\sum_{t\in T} t$.

Note that our hypothesis of sufficient capacity is satisfied since $\sum_{i\in I} W_{ij} = \sum_{k\in I} t_k = \sum_{t\in T} t = C_j, \forall j \in S$. Also, observe that the SCUAP is in $\mathcal{NP}$ and that the construction of the instance of the SCUAP is polynomial in the input size $|T|$. Next, we show that an instance of the PP is a YES instance, if and only if the transformed instance is a YES instance for the SCUAP.

$\Longrightarrow$ First, consider a YES instance of the PP given by a subset $L \subseteq \{1, \ldots, |T|\}$. We then construct subsets of the users leading to a YES instance of the SCUAP as follows: $I_1 \leftarrow L$ and $I_2 \leftarrow \{1, \ldots, |T|\} \setminus L$. Then, $I_1 \cup I_2 = L \cup (\{1, \ldots, |T|\} \setminus L) = \{1, \ldots, |T|\} = I$ and $I_1 \cap I_2 = L \cap (\{1, \ldots, |T|\} \setminus L) = \emptyset$; thus, this assignment is a partition of I. This solution has an objective value $\sum_{j\in S} C_j \left(1 - \frac{\sum_{i\in I_j} W_{ij}}{C_j}\right)^2 =$

$$C_1\big(1 - \frac{\sum_{i\in I_1} W_{i1}}{C_1}\big)^2 + C_2\big(1 - \frac{\sum_{i\in I_2} W_{i2}}{C_2}\big)^2 \tag{4a}$$

$$= \sum_{t\in T} t\big(1 - \frac{\sum_{k\in L} t_k}{\sum_{t\in T} t}\big)^2 + \sum_{t\in T} t\big(1 - \frac{\sum_{k\in\{1,\ldots,|T|\}\setminus L} t_k}{\sum_{t\in T} t}\big)^2 \tag{4b}$$

$$= \sum_{t\in T} t\big(1 - \frac{\frac{1}{2}\sum_{t\in T} t}{\sum_{t\in T} t}\big)^2 + \sum_{t\in T} t\big(1 - \frac{\frac{1}{2}\sum_{t\in T} t}{\sum_{t\in T} t}\big)^2 \tag{4c}$$

$$= 2\sum_{t\in T} t\big(1 - \frac{1}{2}\big)^2 = \frac{1}{2}\sum_{t\in T} t \tag{4d}$$

$$= M. \tag{4e}$$

Equation (4a) follows from the definition of the SCUAP instance, equation (4b) holds by construction, equation (4c) holds since the PP instance is a YES instance, equation (4d) is a simplification, while equation (4e) holds by construction from the definition of M. As equality holds throughout, the constructed instance is a YES instance of the SCUAP.

$\Longleftarrow$ Next, consider a YES instance of the SCUAP given by two subsets I_1 and I_2. We now construct a partition leading to a YES instance of the PP. Consider $L \leftarrow I_1$. It follows that $\{1, \ldots, |T|\} \setminus L = I_2$ since I_1, I_2 partition $I = \{1, .., |T|\}$. It remains to be shown that $\sum_{k \in L} t_k = \sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k$.
Consider the optimal objective function value of this SCUAP instance.

$$\frac{1}{2}\sum_{t \in T} t = M \geq \sum_{j \in S} C_j \left(1 - \frac{\sum_{i \in I_j} W_{ij}}{C_j}\right)^2 \tag{5a}$$

$$= C_1\left(1 - \frac{\sum_{i \in I_1} W_{i1}}{C_1}\right)^2 + C_2\left(1 - \frac{\sum_{i \in I_2} W_{i2}}{C_2}\right)^2 \tag{5b}$$

$$= \sum_{t \in T} t\left(1 - \frac{\sum_{k \in L} t_k}{\sum_{t \in T} t}\right)^2 + \sum_{t \in T} t\left(1 - \frac{\sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k}{\sum_{t \in T} t}\right)^2. \tag{5c}$$

Equation (5a) follows since the optimal objective function value is at most M by definition, where $M = \frac{1}{2}\sum_{t \in T} T$ by construction. Equations (5b) and (5c) follow from the definition of the SCUAP instance and our definition of L. Dividing throughout by $\sum_{t \in T} t > 0$ simplifies equation (5) to

$$\frac{1}{2} \geq \left(1 - \frac{\sum_{k \in L} t_k}{\sum_{t \in T} t}\right)^2 + \left(1 - \frac{\sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k}{\sum_{t \in T} t}\right)^2. \tag{6}$$

We now use Proposition 2.1 with $y \leftarrow \sum_{t \in T} t > 0$. Then, the function

$$\left(1 - \frac{x_1}{\sum_{t \in T} t}\right)^2 + \left(1 - \frac{x_2}{\sum_{t \in T} t}\right)^2, \tag{7}$$

subject to $x_1 + x_2 = \sum_{t \in T} t$, with $x_1, x_2 \in \mathcal{R}^+$ is uniquely minimized at $x_1^* = x_2^* = \frac{1}{2}\sum_{t \in T} t$ giving a value of $\frac{1}{2}$. For the feasible solution $x_1 \leftarrow \sum_{k \in L} t_k$ and $x_2 \leftarrow \sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k$ we then have,

$$\frac{1}{2} \leq \left(1 - \frac{\sum_{k \in L} t_k}{\sum_{t \in T} t}\right)^2 + \left(1 - \frac{\sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k}{\sum_{t \in T} t}\right)^2. \tag{8}$$

Since the minimum is attained uniquely, equations (6) and (8) together yield $x_1^* = x_2^* = \sum_{k \in L} t_k = \sum_{k \in \{1,\ldots,|T|\} \setminus L} t_k$; i.e., the considered set L indeed defines a YES instance of the PP. □

In Appendix A, we provide an example of this mapping. The above discussion directly leads us to our main results of this section that both model (3) and model (1) are $\mathcal{NP}$-hard.

Theorem 2.2 *The BUAP is $\mathcal{NP}$-complete.*

Proof. This follows directly from Lemma 2.1, or from Theorem 2.1 since the SCUAP is a special case of the BUAP. □

Theorem 2.3 *The BFLP is $\mathcal{NP}$-complete.*

Proof. This follows directly from Theorem 2.2 since the BUAP is a special case of the BFLP. In particular, an instance of the BUAP with given inputs I', S' and M' reduces to the BFLP by setting $I \leftarrow I', J \leftarrow S', B \leftarrow |J|$ and $M \leftarrow M'$. □

In the proceeding sections, we develop heuristics to solve both these problems. Fortunately, we find that despite the theoretical hardness of the UAP and the BUAP we can still obtain feasible and high-quality solutions via our heuristics; our computational experiments in Section 6.2 further corroborate this. This observation is similar to finding solutions for the PP that we use for our reduction proofs above, which despite its hardness allows for high-quality heuristic solutions, e.g., the so-called Longest Processing Time scheduling algorithm [19].

3 Heuristics for model (3)

Next, we present three heuristics for solving the BUAP formulated in model (3): the greedy algorithm proposed in [21], an algorithm based on rounding a fractional solution, and an algorithm based on a local search. In what follows, we let $\arg\max\{\cdot\}$ denote the value of one index at which the input set takes its maximum value; if there is more than one such index, we arbitrarily pick one while if there is none the output is empty. We define a function $\texttt{sort}(L, A_l, \text{ascending/descending})$ that sorts elements $l \in L$ depending on their value A_l in ascending or descending order. We define the auxiliary variable $u_j = \dfrac{\sum_{i \in I_j} W_{ij} x_{ij}}{C_j}, \forall j \in J$; Further, we let $[x], [y]$, and $[u]$ denote the entire set of decision variables corresponding to x_{ij}, y_j, and u_j of appropriate dimension. An "assignment" is a solution, $[x]$, to the BUAP, while an "incomplete assignment" is a solution, $[x]$, to the always feasible model (3), where constraint (1d) is relaxed to $\sum_{j \in S} x_{ij} \leq 1$. This means that not all users are assigned a facility. To distinguish between heuristics for the BUAP and the BFLP, we call the former as procedures and latter as algorithms. Finally, we let $\bar{z}_S \geq z_S^*$ and $\bar{z} \geq z^*$ denote the objective function values of model (3) and (1) obtained from any heuristic. For simplicity, we use W_{ij} to denote $U_i P_{ij}$.

3.1 The greedy algorithm of [21]

The algorithm of [21] seeks to provide a feasible solution to the BFLP; however, in this section, we are only interested in a solution for the BUAP. Thus, we extract the relevant parts of this scheme, and summarize it in Procedure 1; we later compare our results with those obtained by this scheme. For each user, we first determine their most preferred facility from those that have sufficient capacity to accommodate it (line 5). If no such facility is available, we terminate reporting infeasibility. Else, we consider each facility sequentially, and assign users to it in order of their preferences until the facility's capacity is exhausted (lines 8 - 12). We repeat this assignment until all users are allocated. For details, see [21].

3.2 A basic rounding algorithm

In this section, we provide a basic scheme based on rounding the fractional solution x obtained from the continuous relaxation of model (3). Consider the following convex optimization model:

$$\underline{z}_S = \min_x \sum_{j \in S} C_j \Big(1 - \frac{\sum_{i \in I} U_i P_{i,j} x_{i,j}}{C_j}\Big)^2, \text{s.t.}(1c), (1d), x_{i,j} \in (0,1), \forall i \in I, j \in S. \tag{9}$$

Procedure 2 seeks to determine binary assignments of the x_{ij} variables from the solution of model (9). We begin by assigning user i to the facility with the largest x_{ij} value (line 2). We denote the subset of users assigned to a j by $I_j \subseteq I$. We adapt this solution if the capacities of some facilities are exceeded. We denote by S' (line 5) the subset of facilities whose capacities are exceeded. We then seek to reassign

Procedure 1 `greedy assign` (adapted from [21])

Input: an instance of model (3).

Output: status $f \in \{$`feasible`, `infeasible`$\}$ for the given inputs; if `feasible`: solution $[x]$, utilization $[u]$, and objective function value $\overline{z}_S$; if `infeasible`: $[x] \leftarrow 0, [u] \leftarrow 0, \overline{z}_S \leftarrow +\infty$.

```
1: Initialize: $I' \leftarrow I$; $[x] \leftarrow 0$; $R_j \leftarrow C_j, \forall j \in S$.
2: while $I' \neq \emptyset$ do
3:     $M_j \leftarrow \emptyset, \forall j \in S$.
4:     for $i \in I'$ do
5:         $j' \leftarrow \arg\max_{\{j \in S : W_{ij} \leq R_j\}} \{P_{ij}\}$ .
6:         $M_{j'} \leftarrow M_{j'} \cup \{i\}$.
7:     if $M_j = \emptyset, \forall j \in S$, return $f \leftarrow$infeasible; $[x] \leftarrow 0; [u] \leftarrow 0; \overline{z}_S \leftarrow +\infty$; “heuristic failed”.
8:     for $j \in S$ do
9:         $I'' \leftarrow$ sort$(M_j, P_{ij}, \text{descending})$.
10:        for $i \in I''$ do
11:            if $W_{ij} \leq R_j$
12:                $R_j \leftarrow R_j - W_{ij}$; $I' \leftarrow I' \setminus \{i\}$; $x_{ij} \leftarrow 1$.
13: return $f \leftarrow$ feasible; $[x]$; $u_j \leftarrow \frac{\sum_{i \in I} W_{ij} x_{ij}}{C_j}, \forall j \in J$; $\overline{z}_S \leftarrow \sum_{j \in J} C_j (1 - u_j)^2$.
```

users of these facilities to others that have available capacity. Consider a given $j \in S'$. We begin the reassignment by computing the subset of users assigned to this j that have $W_{ij} > C_j$, since these users cannot be assigned to j in a binary solution; we denote this subset as I'_j (line 7). Then, all users in the set I'_j need to be reassigned. We reassign them to the facility other than j that they most prefer and that still has capacity for them in lines 8-11. If we are within the capacity for this facility j, we continue to the next facility (line 12). If we are still over capacity, we further reassign users but now do so seeking to keep assignments in order of preferences; i.e., we first remove users that have low preferences to j (lines 13-18). We stop this reassignment for j when the capacity constraint is satisfied, and go to the next facility (line 18).

Similar to Procedure 1, Procedure 2 can fail to determine a feasible solution even if such a solution exists. This failure is determined in lines 10 or 19; i.e., if a facility with a violated capacity cannot be reassigned, we exit immediately and report a failure. However, the key difference is that unlike Procedure 1, Procedure 2 begins with a complete although infeasible assignment. This assignment comes from a fractional solution that we now seek to make feasible. The similarity is in the steps taken to make this solution feasible; specifically, we (re)assign users to facilities with the highest preference that have sufficient capacity for them.

3.3 Local search approach

Finally, we investigate a few local search heuristics to improve a given feasible solution for model (3). We seek to ensure feasibility of the solution in each refinement, and consider two schemes motivated by [18]: (i) reassigning one user to a different facility, and (ii) swapping the assignments of two users. For each reassignment or swap, we compute the change in the objective function value; here, we only consider facilities that have sufficient capacity available to accommodate the new user. If this change indicates an improvement, we perform the reassignment or swap. As we show in Section 6.2, this decision is computationally cheap. We continue these random reassignments or swaps until a certain time limit or sufficient improvement in the objective function is reached. Such schemes are simple to implement,

Procedure 2 `relaxation rounding`

```
Input: an instance of model (3); a scalar $n_r \leq |S|$.
Output: status $f \in \{$feasible, infeasible$\}$ for the given inputs; if feasible: solution $[x]$, utilization $[u]$, and objective
    function value $\overline{z}_S$; if infeasible: $[x] \leftarrow 0, [u] \leftarrow 0, \overline{z}_S \leftarrow +\infty$.
1: solve model (9) with $n_r$ "most preferred" facilities for each user, get continuous $x$.
2: $a_i \leftarrow \{\arg\max_{j \in S}\{x_{ij}\}\}, \forall i \in I$.
3: $I_j \leftarrow \{i \in I : a_i = j\} \subseteq I, \forall j \in S$.
4: $R_j \leftarrow C_j - \sum_{i \in I_j} W_{i,j}, \forall j \in S$.
5: $S' \leftarrow \{j \in S : R_j < 0\} \subseteq S$.
6: for $j$ in $S'$ do
7:     $I'_j \leftarrow \{i \in I_j : W_{ij} > C_j\} \subseteq I_j$.
8:     for $i \in I'_j$ do
9:         $k \leftarrow \arg\max_{k'}\{P_{ik'} : R_{k'} - W_{ik'} \geq 0, k' \neq j\}$.
10:        if $k = \emptyset$, return $f \leftarrow$ infeasible; $[x] \leftarrow 0; [u] \leftarrow 0; \overline{z}_S \leftarrow +\infty$; "heuristic failed".
11:        $I_k \leftarrow I_k \cup \{i\}$; $I_j \leftarrow I_j \setminus \{i\}$; $R_k \leftarrow R_k - W_{ik}$; $R_j \leftarrow R_j + W_{ij}$.
12:    if $R_j \geq 0$, break. Continue with next iteration of outer for loop (line 6).
13:    $I''_j = \texttt{sort}(I_j \setminus I'_j, P_{ij}, \text{ascending})$
14:    for $i \in I''_j$ do
15:        $k \leftarrow \arg\max_{k'}\{P_{ik'} : R_{k'} - W_{ik'} \geq 0, k' \neq j\}$.
16:        if $k = \emptyset$, break. Continue with next iteration of for loop (line 14).
17:        $I_k \leftarrow I_k \cup \{i\}$; $I_j \leftarrow I_j \setminus \{i\}$; $R_k \leftarrow R_k - W_{ik}$; $R_j \leftarrow R_j + W_{ij}$.
18:        if $R_j \geq 0$, break. Continue with next iteration of outer for loop (line 6).
19:    if $R_j < 0$, return $f \leftarrow$ infeasible; $[x] \leftarrow 0; [u] \leftarrow 0; \overline{z}_S \leftarrow +\infty$; "heuristic failed".
20: return $f \leftarrow$ feasible; $x_{ij} \leftarrow 1, \forall i \in I_j$, else $x_{ij} \leftarrow 0$; $u_j \leftarrow \frac{\sum_{i \in I} W_{ij} x_{ij}}{C_j}, \forall j \in J$; $\overline{z}_S \leftarrow \sum_{j \in J} C_j (1 - u_j)^2$.
```

and are well-known in the literature on local search, see, e.g., [16]. We adapt these for our models, and provide specific procedures and details for both these schemes in Appendix B.1.

4 Heuristics for the BFLP

4.1 Background

In this section, we return to our central problem—the BFLP as defined by model (1)—and study heuristics for its solution. These heuristics utilize those we investigate in Section 3 for the BUAP. A key difference between these two problems is the additional combinatorial restriction imposed by the BFLP of opening at most B out of $|J|$ facilities. If model (1) is feasible, there exists an optimal solution that opens exactly B facilities; see Proposition S2 in Appendix A. However, a feasible solution, or even an optimal solution, of the BUAP with B arbitrarily chosen facilities might still be suboptimal for the BFLP. The aim of this section is to determine not only a good user assignment, but also a good set of B facilities corresponding to this assignment. To this end, we develop three schemes adapted from classical solution methods of the FLP.

Our first two schemes, that we study in Section 4.2 and Section 4.3, build upon the so-called DROP and ADD algorithms. These classic schemes rely on a given set of existing facilities from which we sequentially close and open facilities, respectively. We adapt these algorithms for the specifics of our problem. As such, the termination criteria in our adaptations is when exactly B facilities are open. Additionally, we employ a local search heuristic to improve solutions provided by the above two schemes. For each of our schemes, we use the BUAP heuristics discussed in Section 3 as internal procedures or subroutines. In Section 6, we provide computational results that compare each of these schemes with each other, the results of the naive solution and with a local search heuristic from [21]. We begin with a summary of these heuristics based on model (2) and then provide specific details later in this section.

```
Input: an instance of model (2).
Output: a set of open facilities $S$.
1: $S \leftarrow J$; $\delta^* \leftarrow 1$.
2: while $\delta^* > 0$ do
3:     $\delta_j \leftarrow f_j + T(I,S)$
           $-T(I, S \setminus \{j\}), \forall j \in S$.
4:     $j^* \leftarrow \arg\max_{j\in S} \delta_j$;
       $\delta^* \leftarrow \max_{j\in S} \delta_j$.
5:     if $\delta^* > 0$ , $S \leftarrow S \setminus \{j^*\}$.
6: return $S$.
```

(a) DROP

```
Input: an instance of model (2); an additional facility $j'$ with a
    large cost and a large capacity.
Output: a set of open facilities $S$.
1: $S \leftarrow \{j'\}$; $\delta^* \leftarrow 1$.
2: while $\delta^* > 0$ do
3:     $\delta_j \leftarrow T(I,S) - T(I, S \cup \{j\})$
           $-f_j, \forall j \in S$.
4:     $j^* \leftarrow \arg\max_{j\in S} \delta_j$; $\delta^* \leftarrow \max_{j\in S} \delta_j$.
5:     if $\delta^* > 0$ , $S \leftarrow S \cup \{j^*\}$.
6: return $S \setminus \{j'\}$.
```

(b) ADD

Fig. 1: The DROP and ADD algorithms of [7].

Let $T(I,S)$ denote the objective function value of model (2) for S facilities. We summarize the generic DROP scheme [6,7] in Algorithm 1a for model (2). Beginning with all the facilities as open ($y_j = 1, \forall j \in J$) we compute the facility closing (or "dropping") which leads to the largest decrease in the objective function value. Due to the structure of the BFLP, its objective function value increases as we close facilities (Proposition S3 in Appendix A); however, for model (2) dropping facilities is not guaranteed to decrease its objective function value. Another difference from the classical FLP is that we additionally have a budget on the number of facilities to keep open. We thus adapt the classical procedure to continue until at most B facilities remain open, as opposed to the classical DROP algorithm that continues until no more facilities can be dropped without increasing the objective function. We present details of how we adapt the DROP algorithm to the BFLP in Section 4.2.

The ADD algorithm [11,7] is similar and is summarized in Figure 1b. The difference here is that we begin with a given set of facilities (e.g., the empty set), which could be infeasible for model (2). Jacobsen circumvents this issue by including a fake facility, j', to begin with that has a large enough cost and capacity [7]; for appropriately large costs, this facility is no longer needed to satisfy user demands as the algorithm progresses. The rest of the algorithm proceeds in a similar way to the DROP algorithm, but by adding facilities sequentially. We present details of our adaptation to the BFLP in Section 4.3.

Our local search ideas are extensions of perturbation procedures to improve given feasible solutions of the CFLP, see, e.g., [23]. In general, these procedures begin with a solution of the CFLP and select a facility to close. Then they run one iteration of the ADD algorithm to determine a facility to open that provides the largest decrease in the objective function value. This process is repeated until no improvement is achieved. Analogously, these procedures open a single facility and then run one iteration of the DROP algorithm. Our local search scheme combines both these ideas into our third algorithm, distinguishing it from the traditionally separate improvement procedures of [23], to solve the BFLP. We present details in Section 4.4.

As we repeatedly use several methods to solve the BUAP defined by model (3), for brevity we define these in Procedure 3. Here, the procedure takes as input a set of $|I|$ users and $|S| \leq |J|$ facilities, and constructs an assignment using method m; choices of m include those we study in Section 3, e.g., `greedy assign`, `greedy assign` with `localsearch reassignment`, `relaxation rounding`, or `relaxation rounding` with `localsearch swap`.

Procedure 3 `user assignment`

Input: an instance of model (3); a method m to construct a feasible solution of the instance.

Output: status $f \in \{$`feasible`, `infeasible`$\}$ for the given inputs; if `feasible`: solution $[x]$, utilization $[u]$, and objective function value $\overline{z}_S$; if `infeasible`: $[x] \leftarrow 0, [u] \leftarrow 0, \overline{z}_S \leftarrow +\infty$.

```
1: Run assignment method $m$ on the input instance and return the result.
```

4.2 The close greedy algorithm

Our first scheme begins with a simple idea based on the DROP procedure, that we build up in three versions; in Section 6.3.1 we provide computational experiments that compare the progression in the objective function's value and runtime at each version. We denote the class of heuristics within this section as `close greedy`, and the algorithms corresponding to the first and third versions as `close greedy basic` and `close greedy improved`, respectively. The second version changes only a single line of `close greedy basic`, so for brevity we do not provide an additional pseudo-code for this.

Algorithm 1 summarizes the first version. We input an instance of model (1), and some schemes, m, m', for user assignment methods that the proposed algorithm makes repeated use of via Procedure 3. We begin with method m with all facilities, J, open (line 1). Iteratively, we reduce the number of facilities to the allowed B, closing one facility in each iteration (lines 2-9). However, rather than choose facilities to close from the entire set, we restrict our search to a candidate pool of n_c facilities with the lowest utilization values (line 3); here, we follow the suggestion in [22]. Although, this means we do not necessarily make the best choice, even locally, this step helps reduces computational effort. We note there is a natural trade-off between computational effort and n_c; e.g., see, Table S1a in Appendix D that shows the average run time increases from 1,973 seconds to 6,757 seconds when n_c is increased from 5 to 20.

We determine "good" facilities to close—that increase the objective function value the least—via method m in the loop in lines 5-7. Once exactly B facilities are open, we run `user assignment` again (line 10), but this time with a possibly different method m', to check if a better assignment than the one we determined so far is readily available. In Section 6.3.1, we find advantage in using the computationally cheap method m = `greedy assign` multiple times when closing facilities throughout the algorithm coupled with the slower but better method m' = `relaxation rounding` with `local search reassign` after the set of open facilities is determined. We return the obtained solution in line 13. If no feasible assignments can be determined for a closure, we skip and ignore this closure (line 7); while, if none of the closures result in a feasible assignment we report a failure (line 8). This concludes the first version.

The strength of such a scheme rests on the ability to compute good, although suboptimal, user assignments multiple times and speedily; it is here that the quick implementation methods we discuss in Section 3 come useful. For example, in our computational experiments, we find each `user assignment` call takes about two-thirds of a second for instances with $|I| = 2,060, |J| = 1,394$ for m = `greedy assign`. Next, we seek to further improve this. Rather than compute the entire assignment from scratch, we now compute these just for the users previously assigned to the one facility, j', that is closed and keep all other assignments same. Procedure 4, directly adapted from Procedure 1, provides a quick scheme to this effect. Although the value of this procedure decreases when several facilities are closed, this adaptation significantly reduces the computational effort required for the `BUAP` by allowing us to increase n_c by an order of magnitude while still keeping runtimes reasonable; see, computational results in Section 6.3.1. To summarize, we simply replace line 6 in Algorithm 1 with a call to `greedy reassign` with inputs I, S, j' and $[x]$. This completes our second version of the algorithm.

Algorithm 1 `close greedy basic`

```
Input: an instance of model (1); methods $m, m'$ for user assignment defined in Procedure 3; scalar $n_c \leq |J|$.
Output: status $f \in$ {feasible, infeasible} for the given inputs; if feasible: $[x]$, $[y]$, $\overline{z}$.
 1: Initialize: $S \leftarrow J$; $[f, x, u, \overline{z}] \leftarrow$ user assignment$(I, S, m)$.
 2: while $|S| > B$ do
 3:     $J' \leftarrow$ {indices of smallest $n_c$ values of $u_j, j \in S$} $\subseteq J$.
 4:     $[f^*, x^*, u^*, z^*, j^*] \leftarrow$ [infeasible$, 0, 0, +\infty,$ "none"].
 5:     for $j' \in J'$ do
 6:         $[f', x', u', z'] \leftarrow$ user assignment$(I, S \setminus \{j'\}, m)$.
 7:         if $f'$ = feasible and $z' < z^*$, $[f^*, x^*, u^*, z^*, j^*] \leftarrow [f', x', u', z', j']$.
 8:     if $f^*$ = infeasible, return $f \leftarrow$ infeasible; "heuristic failed".
 9:     $S \leftarrow S \setminus \{j^*\}$; $[f, x, u, \overline{z}] \leftarrow [f^*, x^*, u^*, z^*]$.
10: $[f', x', u', z'] \leftarrow$ user assignment$(I, S, m')$.
11: if $f'$ = feasible and $z' < \overline{z}$, $[f, x, u, \overline{z}] \leftarrow [f', x', u', z']$.
12: $y_j = 1, \forall j \in S$; else, $y_j = 0$.
13: return $f$; $[x]$; $[y]$; $\overline{z}$.
```

Procedure 4 `close greedy reassign`

```
Input: an instance of model (3); a facility to close $j'$; an assignment $[x]$ of $I$ to $S$.
Output: status $f \in$ {feasible, infeasible} for the given inputs; if feasible: solution $[x]$, utilization $[u]$, and objective
    function value $\overline{z}_S$; if infeasible: $[x] \leftarrow 0, [u] \leftarrow 0, \overline{z}_S \leftarrow +\infty$.
 1: Initialize: $I' \leftarrow \{i \in I : x_{ij'} = 1\}$; $x_{ij'} \leftarrow 0, \forall i \in I'$; $S \leftarrow S \setminus \{j'\}$; $R_j \leftarrow C_j - \sum_{j \in I} U_i P_{ij} x_{ij}, \forall j \in S$.
 2: Lines 2 - 13 from Algorithm 1.
```

Algorithm 2 `close greedy improved`

```
Input: an instance of model (1); methods $m, m'$ for user assignment defined in Procedure 3; scalar $n_c \leq |J|$.
Output: status $f \in$ {feasible, infeasible} for the given inputs; if feasible: $[x], [y], \overline{z}$.
 1: Initialize: $S \leftarrow J$; $J' \leftarrow J$; $\delta_j \leftarrow 0, \forall j \in J$; $[f, x, u, \overline{z}] \leftarrow$ user assignment$(I, S, m)$.
 2: while $|S| > B$ do
 3:     $[f^*, x^*, u^*, z^*, j^*] \leftarrow$ [infeasible$, 0, 0, +\infty,$ "none"].
 4:     for $j' \in J'$ do
 5:         $[f', x', u', z'] \leftarrow$ greedy reassign$(I, S, j', x)$.
 6:         $\delta_{j'} \leftarrow z' - \overline{z}$.
 7:         if $f' =$ feasible and $z' < z^*$, $[f^*, x^*, u^*, z^*, j^*] \leftarrow [f', x', u', z', j']$.
 8:     if $f^* =$ infeasible, return $f \leftarrow$ infeasible; "heuristic failed".
 9:     $S \leftarrow S \setminus \{j^*\}$; $[f, x, u, \overline{z}] \leftarrow [f^*, x^*, u^*, z^*]$.
10:     $J' \leftarrow \{j \in S :$ indices of smallest $n_c$ values of $\delta_j\} \subseteq J$.
11: $[f', x', u', z'] \leftarrow$ user assignment$(I, S, m')$.
12: if $f' =$ feasible and $z' < \overline{z}$, $[f, x, u, \overline{z}] \leftarrow [f', x', u', z']$.
13: $y_j = 1, \forall j \in S$; else, $y_j = 0$.
14: return $f$; $[x]$; $[y]$; $\overline{z}$.
```

In the third version, we consider a different way to choose the candidate pool of facilities, J'. So far, we determine these facilities as the ones with the lowest utilization. However, since the `BFLP` balances both access and utilization, this idea might not be the best choice. We now determine the candidate pool by considering facilities that were good candidates to close in previous iterations but were not chosen. Algorithm 2 summarizes this scheme that we describe next.

As in `close greedy basic`, we initialize with all the available facilities, J, and solve the `BUAP` (line 1). Now, let δ_j denote the change in the objective function value that closing facility j brings to the objective function of model (3) (line 6); we compute the objective function value in line 5 via step 2 as described in Procedure 4. Here, $\overline{z}, z'$ denote the objective function value for model (3) before and after closing facility j, respectively. At each iteration, we then determine the n_c facilities among those that are still open that had the smallest values of δ_j in the last iteration it was calculated. The reasoning behind our choice of small values of δ_j is that we seek to keep the increase in the objective function of the minimization problem small. If the `BUAP` is solved to optimality, then $\delta_j \geq 0$ since the the objective function value cannot decrease when a facility is closed (this follows from Proposition S3 in Appendix A.1). However, since we do not solve the `BUAP` to optimality, it is possible that $\delta_j < 0$ as well. We note that δ values are updated only for facilities within the set J' at each iteration. Hence, when closing the first facility, we also initialize δ, by considering J' to be the entire set of facilities. In Appendix B.2, we provide an additional discussion on the δ parameter.

We further note that the for loops over the set J' can be implemented in parallel, saving potentially significant computational effort. Swapping between several different methods in the while loop, rather than staying with a single method m, is another suggestion for improvement. A computationally cheap local search heuristic, similar to what we describe in Section 4.4, could also be implemented within the algorithms. Finally, we note that using `greedy reassign` is expected to decrease the quality of the assignments when used over a lot of iterations. To counter this effect, one could recompute the complete assignment after every few iterations; however, in our computational experiments this enhancement did not lead to any significant improvement.

4.3 The open greedy algorithm

Our second scheme is motivated by `ADD`-styled algorithms and seeks to sequentially open facilities rather than closing them as in the `DROP`-styled algorithms. As mentioned in Section 4.1, a key difference in these two algorithms is how infeasible iterations are handled; infeasibility arises when some users are left unassigned to any facility due to capacity restrictions. Since we seek to use the improvements discussed for `close greedy` directly now, we need a method for adapting an assignment when a facility is opened, analogous to Procedure 4. This procedure accomplishes two tasks: for a previously infeasible assignment, we assign the unassigned users to the newly opened facility; and for a previously feasible assignment, we reassign appropriate users to the newly opened facility.

Procedure 5 `greedy reassign open`

Input: an instance of model (3); a facility to open $j^* \in J \setminus S$; a (incomplete/complete) assignment, $[x]$; the depth d of the local search reassignment.

Output: status $f \in \{\texttt{feasible}, \texttt{infeasible}\}$ for the given inputs; solution $[x]$, utilization $[u]$, and objective function value $\overline{z}_S$.

```
1: Initialize: $S \leftarrow S \cup \{j^*\}$; $I' \leftarrow \{i \in I : x_{ij} = 0, \forall j \in J\}$; $R_j \leftarrow C_j - \sum_{j \in I} U_i P_{ij} x_{ij}, \forall j \in S$.
2: Lines 2-12 from Procedure 1 with line 7 replaced by "break to line 3 in this algorithm".
3: $J' \leftarrow \{j^*\}$; $k \leftarrow 0$.
4: while $k < d$ do
5:     if $J' = \emptyset$, break to line 18.
6:     $J'_k = \emptyset$.
7:     for $j' \in J'$ do
8:         $I'' \leftarrow \texttt{sort}(\{i \in I : U_i P_{ij'} \leq R_j, x_{ij'} = 0\}, P_{ij'}, \text{descending})$.
9:         for $i \in I''$ do
10:            if $i \in I'$ and $U_i P_{ij'} \leq R_{j'}$
11:                $x_{ij'} \leftarrow 1$; $R_{j'} \leftarrow R_{j'} - U_i P_{ij'}$; $I' \leftarrow I' \setminus \{i\}$.
12:            else
13:                $j'' \leftarrow \arg\max_{j \in J} \{x_{ij}\}$
14:                if if reassignment better$(i, j', j'', R_j) = \texttt{True}$
15:                    $x_{ij''} \leftarrow 0$; $x_{ij'} \leftarrow 1$; $R_{j''} \leftarrow R_{j''} + U_i P_{ij''}$; $R_{j'} \leftarrow R_{j'} - U_i P_{ij'}$.
16:                    $J'_k \leftarrow J'_k \cup \{j''\}$.
17:     $J' \leftarrow J'_k$; $k \leftarrow k + 1$.
18: if $I' = \emptyset$, $f \leftarrow \texttt{feasible}$; else $f \leftarrow \texttt{infeasible}$.
19: return $f$; $[x]$; $u_j \leftarrow \frac{\sum_{i \in I} W_{ij} x_{ij}}{C_j}, \forall j \in J$; $\overline{z}_S \leftarrow \sum_{j \in J} C_j (1 - u_j)^2$.
```

Procedure 5 executes both these tasks, and we summarize it next. Here, we begin with a possibly incomplete assignment of users to the available set of facilities, S, plus a facility j^* that we consider opening. This incomplete assignment could arise from a previous iteration of the `open greedy` algorithm, or from a previous call to `greedy reassign open`. The set I' is the set of users that are currently unassigned; this set is empty if we begin with a complete (or, feasible) assignment. Procedure 5 then determines a (potentially, still incomplete) assignment to the $S \cup \{j'\}$ available facilities. If I' is not empty, we use parts of Procedure 1 to greedily assign these users to the available facilities. Afterwards, or if I' is empty, we seek to compute a better assignment. The set $J' \subseteq J$ denotes a candidate pool of facilities we seek to assign users to. The rest of the algorithm conducts a local search to compute the users assigned to j^*, plus a potential reassignment of users to the S facilities due to the corresponding increase in their available capacity. We explain this search below.

For each facility $j' \in J'$, we determine the set of users, I'', that are candidates for its assignment. We do so in line 8 accounting for the facility's capacity and prioritizing by the preferences of users. We distinguish two cases for each of these candidate users: those that are currently unassigned (line 10), and those that are currently assigned (line 12). The former case exists only for incomplete assignments; here, we simply check whether the facility has sufficient capacity for the user (line 10), assign the user, update the facility's remaining capacity, and remove the user from set of unassigned users (line 11). This improves the objective function since previously unassigned users are now assigned. The latter case is for complete assignments; here, we determine if a reassignment of the user from its existing facility j'' to j' is beneficial (line 14). We do so via the `if reassignment better` procedure defined in Procedure S1 in Appendix B.1. If so, we conduct the reassignment, update the remaining capacities of the two facilities involved (line 15), and include the facility j'' to the candidate pool of facilities J'_k considered in the next iteration. After considering all the facilities in J', we go to the next iteration. If no reassignments are made in the previous iteration, we terminate (line 5). Otherwise, the next iteration of the outer while loop probes deeper within the set of facilities in J' in search of an even better assignment. In this sense, the parameter d determines the "depth" of the local search. Thus, $d = 1$ indicates that we only seek to reassign users to j^*, while larger values of d indicate that we additionally reassign users to all the facilities that lost their users in the immediately previous iteration. Despite this probing, the final assignment could still be incomplete; we check this in line 18. Even if only an incomplete assignment is computed, we still return it as it is beneficial in an algorithm that uses this subroutine. Line 19 returns the output.

To summarize, Procedure 5 combines the `greedy assign` procedure with a local search. This local search is necessary for `ADD`-styled algorithms because, unlike `DROP`-styled algorithms, a natural candidate set of users to assign to the new facilities is unavailable. If the input assignment is incomplete, then such a set of users is readily available; however, even then a local search improves on an input assignment. Next, we describe an algorithm to solve the `BFLP`, based on the `ADD` procedure that uses Procedure 5 as a subroutine.

Algorithm 3 `open greedy`

```
Input: an instance of model (1); methods $m, m'$ for Oracle user assignment defined in Procedure 3; the number of facilities
    to consider at each iteration, $n_c \leq |J|$; a depth $d$ for Procedure 5; a scalar $n_f \leq |J|$ for number of iterations after which
    to recompute assignment.
Output: status $f \in \{$feasible, infeasible$\}$ for the given inputs; if feasible: $[x]$, $[y]$, $\overline{z}$.
 1: Initialize: $S \leftarrow \emptyset$; $J' \leftarrow J$; $\delta_j \leftarrow 0, \forall j \in J$; $[f, x, u, \overline{z}] \leftarrow [$infeasible$, 0, 0, \sum_{j \in J} C_j]$.
 2: while $|S| < B$ do
 3:     $[f^*, x^*, u^*, z^*, j^*] \leftarrow [$infeasible$, 0, 0, +\infty, \text{“}none''\text{]}$.
 4:     for $j' \in J'$ do
 5:         $[f', x', u', z'] \leftarrow$ greedy reassign open$(I, S, j', x, d)$.
 6:         if $(f' =$ feasible or $f' = f)$ and $z' < z^*$, $[f^*, x^*, u^*, z^*, j^*] \leftarrow [f', x', u', z', j']$.
 7:         $\delta_{j'} \leftarrow z' - \overline{z}$.
 8:     $S \leftarrow S \cup \{j^*\}$; $[f, x, u, \overline{z}] \leftarrow [f^*, x^*, u^*, z^*]$.
 9:     $J' \leftarrow \{j \in J \setminus S$ : indices of smallest $n_c$ values of $\delta_j\} \subseteq J$.
10:     if $|S| \equiv 0 \pmod{n_f}$
11:         $[f', x', u', z'] \leftarrow$ user assignment$(I, S, m)$.
12:         if $\left(f' =\right.$ feasible and $\left.z' < \overline{z}\right)$, $[f, x, u, \overline{z}] \leftarrow [f', x', u', z']$.
13: $[f', x', u', z'] \leftarrow$ user assignment$(I, S, m')$.
14: if $f' =$ feasible and $z' < \overline{z}$, $[f, x, u, \overline{z}] \leftarrow [f', x', u', z']$.
15: $y_j = 1, \forall j \in S$; else, $y_j = 0$.
16: return $f$; $[x]$; $[y]$; $\overline{z}$.
```

Algorithm 3 summarizes this scheme. We initialize the set S of open facilities as empty, and the candidate pool of facilities to consider at each iteration, J', as J (line 1). Since the algorithm begins with no open facilities, the initial few iterations are infeasible. We thus allow infeasible solutions, until a feasible solution is available from the previous iterations (line 6). We update the parameter δ_j whenever we recompute the change in objective of opening facility j (line 7). We further update the set of open facilities to include the best facility to open, j^*, (line 8), while we update the candidate pool of facilities for the next iteration with those that were the best to open in previous iterations based on values of δ_j (line 9). As we mention in Section 4.2, we recompute assignments from scratch every n_f iterations in lines 10-12, if needed. Finally, once all the facilities are open, we run a final assignment method, m', that potentially improves the constructed assignment (lines 13-14).

As with `close greedy`, possible improvements include considering a certain number of random facilities and parallelizing the `for` loop. We discuss the performance of Algorithm 3 in Section 6.3.2.

4.4 `BFLP` local search

In this section, we combine two local search algorithms based on `ADD` and `DROP` as discussed in [23] into a single local search algorithm. The central idea of this algorithm is to open and close a facility at each iteration, where at least one of these is done in the best possible way. If this interchange leads to a better solution, it is accepted as the current solution. Algorithm 4 presents this scheme that we summarize below; we present computational results in Section 6.3.3. The algorithm relies on two procedures whose details we reserve for Appendix B.2: (i) `initialize change` (Procedure S5) computes the change in the objective function value, δ_j, if facility j is closed or opened, and (ii) `choose fac based on change` (Procedure S6) chooses randomly the best facility to either open or close.

Algorithm 4 considers two cases whether j' is currently open (lines 5-13) or closed (lines 14-22). Consider the first case; the second follows analogously. First, we compute the assignment for when j' is closed and update $\delta_{j'}$ correspondingly (line 6). Then, we choose a candidate pool of n_c facilities, J'', to consider opening from the set of closed facilities, based on the smallest δ_j values (line 7). We then consider each facility in J'' and find the one to open that leads to the smallest objective function value, based on j' being closed and the corresponding assignment $[x']$ (lines 8). If the objective function improves, we update the set of open facilities, S, by closing j' and opening j^* and update the assignment (line 12). Additionally, we update J' to include all facilities again and negate the values of δ for the two facilities considered (line 13). We update J' as after performing a change facilities that we considered opening or closing previously might lead to an improvement if we consider them again. We update δ values since the facilities have swapped from being open to being closed (or vice versa), and the change in objective function of undoing this is exactly the opposite. We recompute the assignment from scratch after exiting the while loop (line 23), and update the assignment if required (line 24). Finally, the resulting set of open facilities S, the assignment x and the objective function value is returned (line 26).

Our computational results in Section 6.3.3 show that Algorithm 4 significantly improves poor feasible solutions. However, good feasible solutions achieved by `open greedy` and `close greedy` improve minimally before stalling the algorithm. Possible extensions to prevent this include beginning with a low value of n_c and increasing it every few iterations, or random perturbation to escape a local optimum, see e.g. [12,3,4].

Algorithm 4 `BFLP local search`

Input: an instance of model (1); a feasible solution to the model (assignment $[x]$, the set of open facilities S with $y_j = 1$, the objective function value $\overline{z}$); method m' for Oracle `user assignment` defined in Procedure 3; the number of facilities to consider each iteration, $n_c \leq |J|$; a depth d for Procedure 5; an iteration limit l for the main `while` loop.

Output: status $f \in \{$`feasible`, `infeasible`$\}$; if `feasible`, $[x], [y], \overline{z}$.

```
1: Initialize: $\delta \leftarrow$ initialize change$(x, S, \overline{z}, d)$; $J' \leftarrow J$; $k \leftarrow 0$.
2: while $|J'| > 0$ and $k < l$ do
3:     $j' \leftarrow$ choose fac based on change$(J, J', S, \delta)$; $J' \leftarrow J' \setminus \{j'\}$; $k \leftarrow k + 1$.
4:     $[f^*, x^*, u^*, z^*, j^*] \leftarrow [$infeasible$, 0, 0, +\infty,$ “none”$]$.
5:     if $j' \in S$
6:         $[f', x', u', z'] \leftarrow$ greedy reassign$(I, S, j', x)$; $\delta_{j'} \leftarrow z' - \overline{z}$ .
7:         $J'' \leftarrow \{j \in J \setminus S$ : indices of smallest $n_c$ values in $\delta_j\}$.
8:         for $j'' \in J''$ do
9:             $[f'', x'', u'', z''] \leftarrow$ greedy reassign open$(I, S \setminus \{j'\}, j'', x', d)$; $\delta_{j''} \leftarrow z'' - \overline{z}$.
10:            if $f'' =$ feasible and $z'' < z^*$, $[f^*, x^*, u^*, z^*, j^*] \leftarrow [f'', x'', u'', z'', j'']$.
11:        if $f^* =$ feasible and $z^* < \overline{z}$
12:            $S \leftarrow (S \cup \{j^*\}) \setminus \{j'\}$; $[f, x, u, \overline{z}] \leftarrow [f^*, x^*, u^*, z^*]$.
13:            $J' \leftarrow J$; $\delta_{j'} \leftarrow -\delta_{j'}$; $\delta_{j^*} \leftarrow -\delta_{j^*}$.
14:    else
15:        $[f', x', u', z'] \leftarrow$ greedy reassign open$(I, S, j', x, d)$; $\delta_{j'} \leftarrow z' - \overline{z}$.
16:        $J'' \leftarrow \{j \in S$ : indices of smallest $n_c$ values in $\delta_j\}$.
17:        for $j'' \in J''$ do
18:            $[f'', x'', u'', z''] \leftarrow$ greedy reassign$(I, S \cup \{j'\}, j'', x')$; $\delta_{j''} \leftarrow z'' - \overline{z}$.
19:            if $f'' =$ feasible and $z'' < z^*$ $[f^*, x^*, u^*, z^*, j^*] \leftarrow [f'', x'', u'', z'', j'']$.
20:        if $f^* =$ feasible and $z^* < \overline{z}$
21:            $S \leftarrow (S \cup \{j'\}) \setminus \{j^*\}$; $[f, x, u, \overline{z}] \leftarrow [f^*, x^*, u^*, z^*]$.
22:            $J' \leftarrow J$; $\delta_{j'} \leftarrow -\delta_{j'}$; $\delta_{j^*} \leftarrow -\delta_{j^*}$.
23: $[f', x', u', z'] \leftarrow$ user assignment$(I, S, m')$.
24: if $f' =$ feasible and $z' < \overline{z}$, $[f, x, u, \overline{z}] \leftarrow [f', x', u', z']$.
25: $y_j = 1, \forall j \in S$; else, $y_j = 0$.
26: return $f$; $[x]$; $[y]$; $\overline{z}$.
```

5 Data Sources and Estimation

In this section, we summarize the data we use for our computational experiments in Section 6. An instance of model (1) requires four parameters: $C_j, U_i, P_{i,j}, \forall i \in I, j \in J$, and B. We solve all instances by varying the budget parameter B in increments of 10% of $|J|$; we consider $B = |J| \times \{0.1, 0.2, \ldots, 0.9\}$. We refer to an *average* for an instance as that over these nine budgets. We then develop four classes of instances, two of which employ actual data from Bavaria while two are synthetically generated. Our first instance class is derived from the set of $|I| = 2{,}060$ users and $|J| = 1{,}394$ facilities in Bavaria; for details, we refer to the original data set described in [22]. We refer to this class as Instance I. In previous work, instances of this class are generally computationally intractable to solve naively. We create a second instance class that is smaller than the first using only a subset of users and facilities. Here, we choose Bavarian ZIP codes that begin with 90, 91 or 92. This is the region including Nuremberg, Fürth, Erlangen and the rural area around them, up to the eastern border of Bavaria. This instance includes $|I| = 368$ users and $|J| = 234$, and we refer to it as Instance II.

Heuristics to solve the `BUAP`, that we describe in Section 3, additionally require the set of open facilities, S, as an input. For our computational experiments for the `BUAP`, we thus derive four additional instance classes as follows. We first solve the `BFLP` naively—for both Instance I and Instance II—with a time limit of 20,000 seconds with $B = 0.3|J|$ and $B = 0.9|J|$. We then let S be the set of facilities opened in the best feasible solution of this MIP for these four instance classes. We refer to these four instances as Instance I-30, Instance I-90, Instance II-30 and Instance II-90, respectively. For the `BUAP`, Instance I-90 and Instance II-90 include a larger number of decision variables than Instance I-30 and Instance II-30, respectively. However, Instance I-30 and Instance II-30 have a lower available capacity per user which leads to a computationally more challenging model; e.g., the ratio $\sum_j C_j / \sum_i U_i$ is 0.71 and 1.01 for Instance I-30 and Instance I-90, respectively. Additionally, we create two instances of the of `SCUAP` with sufficient capacity by taking Instance I, but forcing $C_j = \sum_{i \in I} U_i P_{ij}, \forall j \in J$. We then follow the above procedure to create Instance I-S30 and Instance I-S30.

Next, we describe the construction of our two artificial data classes. We do so to further test the performance of our heuristics on new data, and consider these in Section 6.3.4 alone. We reserve details on the construction of this data set for Appendix C, and only summarize our estimation procedure here. We begin by randomly choosing n pairs of longitudes and latitudes to construct a “rectangle”. By varying the size of this rectangle, we encompass different subareas of Germany (potentially including all of Germany). We restrict U_i to uniformly random values within a range. We artificially place facilities near the geodesic coordinates of the users with a small Gaussian probability, and choose C_j again within a uniform range. Finally, we estimate the parameter P_{ij} using the exponential decay formula based on the distance between user and facility used in [22]. With this procedure, we create two instances : a large but sparse ($n = 5,000$) and a small but dense ($n = 1,500$) instance. We refer to these two instances as Instance III and Instance IV, respectively. The former instance includes a rectangle nearly the size of Germany with $|I| = 5{,}000$ users and $|J| = 2{,}497$ facilities. The latter instance has $|I| = 1{,}500$ users and $|J| = 425$ facilities with users having larger preferences for facilities than the first instance. The average number of facilities within a 5 kilometer radius of any user is 0.84 and 1.20, for Instance III and Instance IV, respectively; i.e., there are more facilities near each user in Instance IV.

6 Computational Results and Analysis

6.1 Setup

Next, we provide computational results for the heuristics discussed in Section 3 and Section 4. We begin in Section 6.2 with our results for the `BUAP`. We then present results for the `BFLP`: (i) in Section 6.3.1, we show that the two progressions of the `close greedy` algorithm indeed lead to improved solutions, (ii) in Section 6.3.2, we present results for the `open greedy` algorithm, (iii) while in Section 6.3.3, we examine whether our `BFLP` local search heuristic improves the previously obtained results. Finally, in Section 6.3.4, we compare the different heuristics against each other and also against the algorithm developed in [21].

We perform all computational experiments on the DelftBlue [5] system with 1 core on an Intel XEON E5-6248R 24C 3.0GHz processor with Pyomo version 6.4.2 and Gurobi version 9.5.2. We run all models with the default setting of the Gurobi parameters, apart from setting `NodeMethod` to 2 when solving the `BFLP` MIP. Our code and data is available at `https://github.com/Malena205/heuristics_quadratic_facility_location`. In what follows, we compare the solutions of our heuristics to the best feasible solution achieved by naively solving the MIP. We do so in two ways: (i) by allowing the MIP to run for 20,000 seconds, and (ii) by running it for exactly the same time as that taken by the heuristic (which is significantly lesser). For fairness of comparison, in the latter case, we do include the time taken to build the optimization model, while the 20,000 seconds are only the time for running the actual optimization, not including the time to build the model. Then, we report results for the quantity $\frac{obj_{MIP} - obj_h}{obj_{MIP}}$, and denote it by Δ_{MIP} and Δ_S for the two above-mentioned cases, respectively; here, obj_{MIP} and obj_h

denote the objective function values obtained naively and using a heuristic, respectively. Then, positive values of Δ denote that the heuristic outperforms the naive solver. We use a 1% tolerance for $|\Delta|$; i.e., we consider all values of $|\Delta| \leq 0.01$ as zero and denote these with a dash (-).

In the implementation of `relaxation rounding`, we reduce the number of x variables by considering only $n_r < |S|$ facilities to determine a user's most preferred facilities; i.e., we reduce the $|I||S|$ combinations to $|I|n_r$. We do so since a suboptimal fractional solution for the `BUAP` that is obtained fast serves our purpose. The choice of n_r is arbitrary — fewer facilities are required if they are sufficiently spread out. In our computational experiments, we find $n_r = 20$ performs well; if we use `relaxation rounding` as the final assignment method, m', we use $n_r = 50$. If the corresponding models are infeasible, we rebuild with a new set of 20 preferred facilities. This idea is different from that proposed in [22], where a cut-off for the preference is used. Specifically, in [22] only (i, j) pairs where $P_{ij} \geq 0.2$ are considered and, further, constraint (1d) is relaxed to an inequality. This requires a post-processing model, as some users are left unassigned [22]. Our approach obviates this need, and requires neither the post-processing model nor the relaxation to an inequality. However, we include these cutoffs in the implementation of the local search heuristics, since it ensures a greater likelihood of obtaining improved solutions and significantly reduces computational effort, see, e.g., [20].

6.2 Analysis: Heuristics for the `BUAP`

We now discuss our computational results for the `greedy assign` and `relaxation rounding` procedures, that we mention in Section 3, to solve the `BUAP`; further, we compare both the heuristics with and without the two local search additions of Section 3.3 (`reassign` and `swap`). Table 1 presents results for both the `SCUAP` (first two columns) and `BUAP` (last four columns). The first observation we make is that all our heuristic methods succeed in finding feasible solutions no more than 1.3% worse than those obtained naively; further, the heuristics achieve the same or better solution in 18 of the 36 cases we present in Table 1. Importantly, the heuristics take no more than 12 seconds — while the naive solution method can still fail to achieve the corresponding objective function value even in 20,000 seconds.

Table 1: Results of the different `BUAP` heuristics on different instances. A "-" indicates a gap of zero within our considered tolerance. Times are rounded up to the nearest integer. Δ_{MIP} is defined in Section 6.1. Within 20,000 seconds, Instance I-30, Instance I-90, and Instance I-S90 are solved to a MIP gap of 0.02%, 0.35%, and 90.9%, respectively, while all other instances are solved optimally. `relaxation rounding` is run with $n_r = 20$. For details, see section 6.2.

		Instance I-S30		Instance I-S90		Instance I-30		Instance I-90		Instance II-30		Instance II-90	
Heuristic	**Local search**	**Run time [s]**	Δ_{MIP} [%]	**Run time [s]**	Δ_{MIP} [%]	**Run time [s]**	Δ_{MIP} [%]	**Run time [s]**	Δ_{MIP} [%]	**Run time [s]**	Δ_{MIP} [%]	**Run time [s]**	Δ_{MIP} [%]
`relaxation rounding`	none	9	-	12	6.55	12	-0.28	11	-0.09	7	-	3	-
	`reassign`	9	-	12	6.55	12	-0.09	11	-0.05	7	-	3	-
	`swap`	9	-	12	6.55	12	-0.26	11	-0.05	7	-	3	-
`greedy assign`	none	1	-	1	6.55	1	-1.31	1	-1.13	1	-1.05	1	-0.71
	`reassign`	1	-	2	6.55	1	-0.25	2	-0.17	1	-0.21	1	-0.03
	`swap`	1	-	2	6.55	1	-1.12	2	-0.99	1	-1.04	1	-0.68

Within the considered time limit, we naively obtain an optimal solution in 6,567, 15, and 31 seconds for Instance I-S30, Instance II-30, and Instance II-90, respectively, including the time to build the models. For Instance I-S90, Instance I-30, and Instance I-90 we terminate with an optimality gap of 90.9%, 0.02%, and 0.35%, respectively. We first discuss the two `SCUAP` instances. The small Instance I-S30 is solved to optimality both naively and by each of the six heuristics; however, the larger Instance I-S90 is challenging to solve naively. Here, the naive solver spends its entire quota of 20,000 seconds on the root node itself, terminating with an optimality gap of 91%. In contrast, the heuristics take only a few seconds and still achieve solutions over 6.6% better than those obtained naively. For the other four columns, `relaxation rounding` achieves nearly the same objective function value as that obtained naively on 6 of the 12 cases. In contrast, `greedy assign` fails to do so for even a single case. However, invoking these procedures multiple calls could still slow algorithms for the BFLP, e.g., a single run of `relaxation rounding` takes about 10 seconds for the larger four instances. Interestingly, run times for `relaxation rounding` are marginally larger for smaller budgets than the larger ones, suggesting more reassignments are required to make the solution feasible after rounding. Further, run times of `relaxation rounding` are five to ten times larger than `greedy assign`, although the former achieves better solutions. These observations suggest invoking `greedy assign` multiple times for user assignment within algorithms that solve the BFLP, while using `relaxation rounding` only as the final user assignment method in our algorithms. This empirical observation our proposal for different choices of m and m' in Section 4, and we follow this in the forthcoming sections. Finally, comparing the local search heuristics, `local search reassign` demonstrates the most value. Δ values for it are at least as good (i.e., large) as both no local search or `swap` for all of the eight cases with comparable run times. In contrast, `local search swap` does not lead to any significant improvements.

We now summarize our recommendations for which heuristics to use for the `BUAP`. Since we call these procedures several times, our aim here is to achieve solutions with very little computational effort. Our results suggest significant value in using the local search additional to the heuristics, as it is fast and significantly improves the previously obtained solution. Comparing our two local search schemes, we find reassigning users to be superior than swapping them both in terms of improving the objective function value and the run times. Overall, we find the `greedy assign` with `local search reassign` to be the best performer. In contrast, if run times are a lesser concern we suggest using `relaxation rounding` with `local search reassign` since it leads to results identical or better than the MIP for 4 out of the 6 instances while `greedy assign` with `local search reassign` only achieves this on 2 out of the 6 instances.

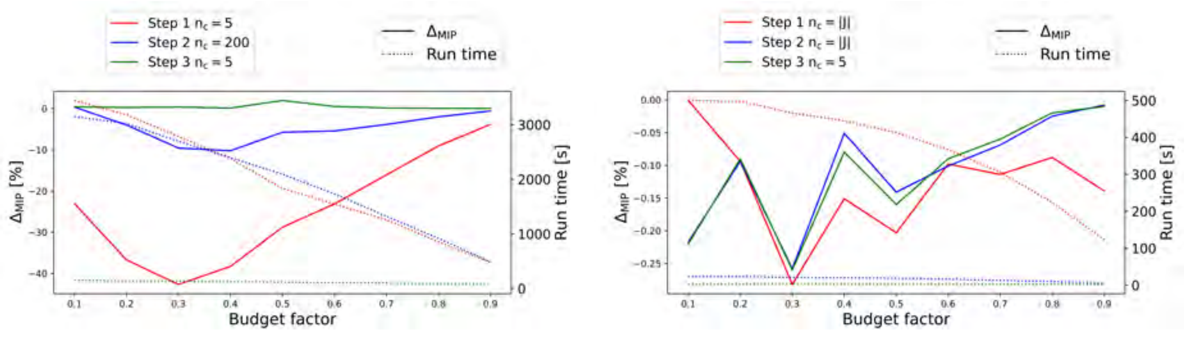
6.3 Results heuristics `BFLP`

In this section, we discuss the results of the heuristics we mention in Section 4 to solve the `BFLP`. We begin by studying the effect of the improvements made to `close greedy basic`, and determining sensible values of the n_c parameter for `close greedy`. Then, we present results for the `open greedy` algorithm by varying the n_c, n_f and d parameters. For the `BFLP local search` algorithm, we discuss how many iterations are needed and whether employing the δ_j parameter to choose facilities has value. We conclude with a summary and recommendations of the choice of the heuristics. Following our results from Section 6.2, we use m' =`relaxation rounding` with `local search reassign` as the final user assignment method all throughout.

6.3.1 Analysis: `close greedy`

We now present results for the `close greedy` algorithm as it progresses through its three versions. We provide detailed computational results in Table S1-Table S3 in Appendix D.1, while Figure 2 displays the improvements of the algorithm across its different versions. We use $n_r = 20$ for Step 1 and $n_r = 50$ for Step 2 and 3, and vary the parameters m and n_c.

We start by discussing the results of `close greedy basic`. On the smaller instance (Instance II), average values for Δ_{MIP} and Δ_S using m = `greedy assign` and $n_c = |J|$ are -0.13% and 0.23%, respectively. With a run time of (on average) only 371 seconds, the `close greedy` achieves nearly the same solution as that obtained naively — even without any of the two forthcoming versions. Within nearly this same time, using m = `relaxation rounding` and $n_c = 5$ instead performs relatively poorly: Δ_{MIP} and Δ_S values are on, average, at -19.72% and -19.24%, respectively. This provides further support from our conclusions drawn from Section 6.2 in favor of using m = `greedy assign` and m' = `relaxation rounding`. On the larger instance (Instance I), using $n_c = |J|$ is computationally prohibitive even with m = `greedy assign`; instead, we use $n_c = 5$ which provide average Δ_{MIP} and Δ_S values of -24.63% and -22.07%, respectively. The latter average is only taken over the budgets where the MIP solver finds a solution by the time the heuristic terminates, which is only the case for three budgets. The run time is 1,973 seconds on average, which suggests increasing n_c further is computationally prohibitive. As the Δ values in Table S1 of Appendix D.1 demonstrate, the first version (or, Step 1) of `close greedy` does not provide a strong competition to the naive solver.



(a) Instance I (b) Instance II

Fig. 2: Performance of the `close greedy` algorithm for its three steps. The initial and final assignment methods are m = `greedy assign` with `local search reassign` and m' = `relaxation rounding` with `local search reassign`, respectively. We use $n_r = 50$ for both Step 2 and Step 3; for Step 1, we do not use local search and use $n_r = 20$. For details, see Section 6.3.1.

Although increasing n_c is computationally demanding, it is one way to reduce the Δ. As we mention in Section 4.2, in the second version we reuse the previous iteration's assignment thereby speeding up the algorithm. Since this requires the method m to only be used once, from now onward we include `local search reassign` with m. Step 2 speeds up the algorithm; alternatively, this version can be viewed as being able to increase n_c for Instance I while keeping similar run times. For Instance I and Instance II being run with $n_c = 200$ and $n_c = |J|$, respectively, average Δ_{MIP} values are now -4.58% and -0.11%, respectively, and average run times are 1,979 and 20 seconds, respectively (Table S2a and Table S2b in Appendix D.1). Thus, for Instance II, average run time decrease by an order of magnitude from 372 seconds to 20 seconds while still being able to maintain $n_c = |J|$. This large decrease in run time is depicted in Figure 2b.

The third and final version instead allows us to decrease n_c while still improving the Δ values; see Table S2-Table S3 in Appendix D.1. As we mention in Section 4.2, this step changes the way we choose the n_c facilities to consider at each iteration. For Instance II, with $n_c = 5$ the algorithm only takes three seconds on average while having the same average Δ_{MIP} value as in the previous step. Increasing n_c to 50 increases the run time to only seven seconds (on average) while each of the nine Δ_{MIP} values are now the same as in the previous step. For Instance I, the results follow a similar trend: at both $n_c = 5$ and $n_c = 50$ an average Δ_{MIP} of 0.41% is achieved with average run times of 115 and 560 seconds, respectively. We further note that not even a single feasible solution is obtained naively in the entire time of completion of the heuristic.

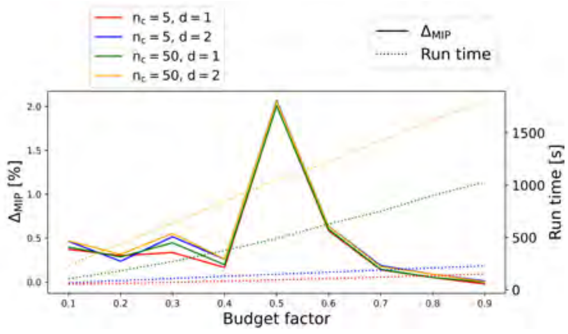
To conclude, both the additional steps to the algorithm we discuss in Section 4.2 significantly improve its performance. Not completely recomputing the assignment but instead adapting it leads to a large decrease in run time, while choosing which facilities to consider based on their performance in previous iterations leads to a large improvement in the objective function value achieved. With this latter improvement, even when only five facilities are considered at each iteration, the results are significantly improved and further increasing n_c only improves the solution quality marginally. These results suggest that our heuristics manage to select the "correct" facilities—even when our candidate pool of facilities shrinks to n_c as opposed to $|J|$—since the objective function values are at least those obtained naively.

6.3.2 Analysis: `open greedy`

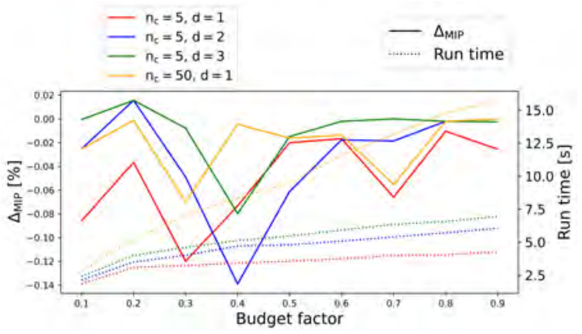
We now present results for the `open greedy` algorithm by varying its parameters; we again reserve detailed computational results for Appendix D.2. Similar to Section 6.3.1, we use $m =$ `greedy assign` with `local search reassign` and $m' =$ `relaxation rounding` with `local search reassign` for the initial and final assignment methods. There are three important parameters to consider: n_f, n_c and d. For most budget values, we observe no significant change (results not shown) by varying n_f that determines the frequency of recomputing assignments; thus, we do not recompute assignments from scratch. We consider $n_c = 5, 50$, and $|J|$ that determines the size of the candidate pool of facilities at each iteration.

Similar to our results for `close greedy`, varying n_c provides little improvement in the objective function values; see, Table S4b and Figure 3b for details. For Instance II, the average Δ_{MIP} values are -0.05%, -0.02%, and -0.02% with average run times of 3.5, 9.8 and 28.8 seconds for $n_c = 5, 50$ and $|J|$, respectively, when $d = 1$; i.e., the improvements are marginal. Thus, fixing $n_c = 5$, we vary the parameter d next. For $d = 1, 2, 3$, the average Δ_{MIP} values are -0.05%, -0.03% and -0.01% with corresponding run times of 3.5, 4.6 and 5.3 seconds, respectively. Similar observations follow on Instance I, except with larger run times, see Table S4a and Figure 3a. For $d = 1, 2, 3$, the corresponding Δ_{MIP} values are 0.44%, 0.50% and 0.50% with run times of 98, 149 and 195 seconds, respectively (for $n_c = 5$). Varying n_c from 5 to 50, again has little effect: Δ_{MIP} values improve by only 0.02, 0.01 and 0.02 percentage points for $d = 1, 2, 3$, respectively. We thus conclude that increasing d from 1 to 2 improves the results as expected — but by only a small magnitude; increasing d to 3 is not worth the extra computational effort.

Summarizing, our results follow a similar trend as `close greedy`. Varying the parameter values naturally affects the obtained solutions, however the heuristics are sufficiently capable of finding high quality feasible solutions without any major effort in tuning the parameters. In this sense, the algorithms are robust against parameter choices. Small value of n_c and $d = 2$ both achieve feasible solutions within half percent of those obtained naively in run times that are two orders of magnitude lesser.



(a) Instance I



(b) Instance II

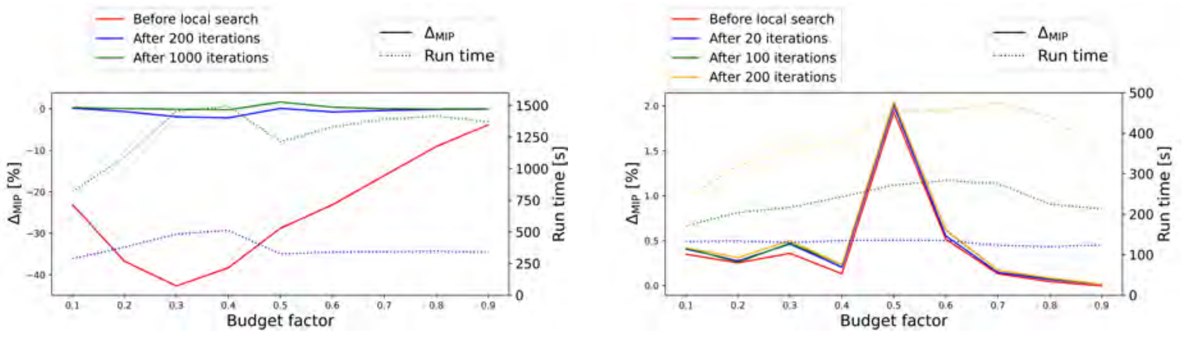
Fig. 3: Performance of the `open greedy` algorithm for choices of n_c and d. The initial and final assignment methods are m = `greedy assign` with `local search reassign` and m' = `relaxation rounding` with `local search reassign`, respectively. We use $n_r = 50$. For details, see Section 6.3.2.

6.3.3 Analysis: BFLP `local search`

We now present results for the `BFLP local search`. To determine the value of such an additional local search, we conduct two experiments. First, we check its performance when initialized with a poor quality feasible solution. For this, we consider the initial solution of `basic close greedy` with m = `greedy assign` on Instance I, which has an average Δ_{MIP} of -24.6%. We run local search with $n_c = 50$ and $d = 2$; higher values of d do not provide any benefit. In 200 iterations of Algorithm 4, the average Δ_{MIP} increases to -0.67% in an average of 373 seconds; while, after 1,000 iterations, the average Δ_{MIP} increases to 0.2% in an average of 1,287 seconds. This demonstrates the significant value of `BFLP local search` in improving feasible solutions. For more detailed results of this experiment, see the left half ("With δ") of Table S5 in Appendix D.3 and also Figure 4a. Second, we ascertain the value of employing the δ parameter; in Section 6.3.1, we demonstrated this value for `close greedy`. We compare our results against a scenario that does not use the δ parameter and instead randomly chooses facilities; see right half ("With random choices") of Table S5. Then, in 200 iterations of `BFLP local search`, the average Δ_{MIP} increases to only -8.14% (as opposed to -0.67% using δ). This experiment again demonstrates the importance of a tailored procedure such as that we employ in achieving high quality solutions.

Next, we consider the performance of `BFLP local search` when initialized with good output solutions from `close greedy` and `open greedy`. Our best feasible solution (until now) for Instance I is obtained through `open greedy` with $n_c = 50, d = 3$; the corresponding average Δ_{MIP} value is 0.52%. The local search is unable to improve upon this average Δ_{MIP} value further in 200 iterations with $n_c = 50$ and $d = 2$; see, Table S6 in Appendix D.3. With the same parameters for the local search, only a marginal improvement for `close greedy`'s best solution is achieved: starting with an average Δ_{MIP} of 0.41%, the local search increases this to 0.46%, 0.48%, and 0.49% after 20, 100, and 200 iterations respectively; see, Table S7a and Figure 4b. We observe that even this small improvement is made within the early few iterations and larger values of the depth parameter, d, and the parameter n_c do not create significant differences either (average Δ_{MIP} is also 0.46% after 20 iterations when using $n_c = 5$, $d = 1$ while taking an average of 87 seconds instead of the 130 seconds with $n_c = 50$, $d = 2$). Table S7b in Appendix D.3 provides additional details on the computations with $n_c = 5$, $d = 1$.

To summarize, small values of both n_c and d are sufficient for improvement via `BFLP local search`; further, only a few iterations are necessary. This also demonstrates that the solutions of `close greedy` and `open greedy` are already close to a local optimal. These results are in part due to the sophisticated choices of the δ parameters in all of our three heuristics.



(a) Initialized with solution from `close greedy` Step 1 (b) Initialized with solution from `close greedy` Step 3

Fig. 4: Performance of the `BFLP` local search with $n_c = 50$, $d = 2$ on Instance I. The final assignment method is $m' =$ `relaxation rounding` with `local search reassign`. We use $n_r = 50$. These run times are for the local search alone. For details, see Section 6.3.3

.

6.3.4 Analysis and Recommendations: Comparison of all heuristics

As we mention before, the aim of this work is to find good feasible solutions with reasonable computational effort. We now summarize our findings of Section 6.3.1-6.3.3 to provide recommendations on which algorithms are best suited for this purpose. To further support these recommendations, we also present results of how the heuristics perform with the chosen parameters on two additional instances, which are described in Section 5. Although our suggestions are based on the `BFLP`, we believe the general guidance extends to other, and, possibly, new FLPs as well.

We start by comparing all our presented algorithms—namely `close greedy` and `open greedy` with and without `BFLP local search` and the algorithm of [21]—to each other. In this sense, we externally validate how robust our algorithms are, while still using the parameters we identified to be the best in Section 6.3.1-6.3.3. Specifically, we run `open greedy` with $d = 2$ and $n_f = |J|$ (since recomputing assignments from scratch did not lead to any major improvements), `open greedy` and `close greedy` with $n_c = 5$, `BFLP local search` with $n_c = 5$, $d = 1$ for 100 iterations, and the algorithm of [21] until there is no change for 100 iterations. Table 2 provides this summary of these five algorithms.

First, all our considered heuristics perform better than the previously proposed method of [21]; across the 36 cases, `close greedy` with and without `BFLP local search` and `open greedy` with and without `BFLP local search` improve average Δ_{MIP} values by 3.13, 3.19, 3.16 and 3.21 percentage points. Thus, in what follows, we compare our heuristics against the naive solution method, as indicated by Δ_{MIP}. Our heuristics perform better than the naive solver for almost all cases of the more challenging Instance I and Instance III, especially at lower budgets; this is likely due to the naive solver struggling with the greater number of combinatorics involved see, panels "Instance I" and "Instance III" of Table 2.

Our results indicate no clear consensus between the performance of `open greedy` and `close greedy` across the four instances; this observation is different to that made in [1] where a greedy algorithm for the critical node problem based on dropping elements clearly outperforms a greedy algorithm that adds elements. On Instance I, Instance II, and Instance IV, `open greedy` is slightly better than `close greedy` with an average improvement in Δ_{MIP} of 0.09, 0.08, and 0.08 percentage points, respectively.

On Instance III, close greedy performs better by an average improvement of Δ_{MIP} of 0.15 percentage points. This is likely due to $n_c = 5$ not being sufficiently high for this very large instance for open greedy. By increasing n_c to 10 in open greedy, the Δ_{MIP} improves by an average of 0.15 compared to the open greedy solution with $n_c = 5$, making the average the same as for close greedy with $n_c = 5$. Increasing n_c further to 50 improves this average by less than 0.01 percentage points compared to the open greedy $n_c = 10$ solution, suggesting that $n_c = 10$ is sufficient for this large instance. Based on this observation, we suggest considering at least 0.5% of all the facilities in each iteration, i.e. $n_c \approx 0.005|J|$. However, we also observe worse performance when $n_c = 0.005|J| \approx 1$ is used on the smallest instance, Instance II. In particular, when using close greedy the average Δ_{MIP} on this instance is -0.34% when $n_c = 1$ and -0.11% when $n_c = 5$ while average run times are identical; see, details in Table S3b in Appendix D.1. Hence, we additionally recommend an absolute lower bound of $n_c \geq 5$.

The budget, B, provides an indicator of the choice between close greedy and open greedy. Run times are larger at lower budgets for close greedy and larger at larger budgets for open greedy. Thus, for a budget of more than half the total number of facilities, we recommend using close greedy; otherwise, we suggest using open greedy. There are no significant differences in the Δ_{MIP} between these two heuristics.

We find the parameter d is guided by how "dense" an instance is; i.e., the number of facilities a user prefers in some radius. For denser instances, we recommend increasing d to two, but not more. This is preferable than increasing n_c; as our results in Section 6.3.1 and Section 6.3.2 show increasing n_c beyond our recommended value increases the run time without any significant effect on Δ_{MIP}. Finally, recomputing assignments from scratch did not lead to any improvements in our experiments, and thus the parameter n_f is of no use.

The BFLP local search results in only marginal improvements as our results in Section 6.3.3 and Table 2 show. For Instance I to Instance IV, across both open greedy and close greedy, the BFLP local search results in an average improvement in Δ_{MIP} of only 0.03, 0.05, 0.07 and 0.07 percentage points, respectively. Further, running the local search beyond 100 iterations leads to very small or no improvement, and we do not recommend running it beyond this point.

Summarizing, our results demonstrate value on employing such tailored heuristics on the BFLP. This value is especially significant for larger and computationally challenging instances, such as those of the size of Germany or those with insufficient capacities to accommodate all users. We recall that our results do not include the time taken to build the model naively, which for the largest instance, Instance III, is itself about an hour. Despite this, the heuristics always achieve solutions comparable or better than those obtained naively in run times that are two or three orders of magnitude lesser.

7 Conclusion

We conclude with a summary of our main findings from this work. We perform a theoretical and computational study of two new problems for the discrete optimization community: the BFLP and the BUAP. We begin our work by showing that both these problems are $\mathcal{NP}$-complete. Motivated by this result, and also from computational evidence of the MIP solver struggling to solve large instances with limited capacity, we developed heuristics for both of these problems. Our heuristics for the BUAP were further inspired by use as subroutines for the heuristics of the BFLP, however they also have value within their own right. We show that our relaxation rounding heuristic outperforms the greedy algorithm developed in [21]. Additionally, we find that a local search that reassigns users performs significantly better than a local search that swaps assignments of users.

For the `BFLP` we developed two tailored algorithms, `close greedy` and `open greedy`, and one local search algorithm, the `BFLP local search`. Our algorithms are rooted in ideas prevalent in the FLP literature, especially those for solving the `CFLP` as described in [7]. However, adapting these to our problems requires several amendments leading them into an almost new form. We found two factors to be particularly relevant in improving the performance of the algorithms. The first important observation is that it suffices to adapt an assignment instead of completely recomputing it. In particular, if a single facility is closed (or, opened) it is sufficient to start with the original assignment and simply reassign users where this is necessary. The second idea that performs very well in our experiments is to choose a candidate set of facilities to close (or, open) based on a measure of how good they were to close (or, open) in a previous iteration. As instances become larger, it becomes intractable to consider all facilities at all steps of an algorithm. This choice of restricting the algorithm to only a few facilities worked exceedingly well in practice; even considering only 5 facilities on instances that have more than $1,000$ facilities shows merit.

Our heuristics outperform both the previously proposed greedy heuristic in [21] as well as the naive solver. Specifically, such computational challenges arise from either the size of the problem (e.g., facilities throughout Germany) or insufficient capacity. In this setting, our heuristics achieve comparable or better results in less than 15 minutes than what a naive solution method achieves in 5.5 hours. Importantly, the heuristics do not require detailed fine tuning of the parameters — typically, even the primitive version of our algorithms is sufficient to achieve good performance. For example, the local search algorithm developed for the `BFLP` only marginally improves upon the results achieved by both `close greedy` and `open greedy`; however, we show that the local search algorithm is especially good at quickly improving poor starting solutions.

Future work could consider extensions on at least two grounds. First, it could investigate in greater detail reasons that the same pool of candidate facilities at different iterations of `close greedy` and `open greedy` performs well. Here, a study of the δ parameter could be performed as well. Second, the performance of the heuristics on the generic generalized quadratic assignment problem and the corresponding facility location problem could be considered.

Table 2: Summary of results for different `BFLP` heuristics. All heuristics are run with $m =$ `greedy assign` with `local search reassign` and $m' =$`relaxation rounding` with `local search reassign` with $n_r = 50$. Positive Δ_{MIP} values indicate our heuristic performs better than the naive solver after 20,000 seconds. For all entries in this table, the naive solver has not even found a single feasible solution by the time these heuristics terminate. See Section 6.3.4 for details.

			Heuristic from [21]		`close greedy`		`close greedy and BFLP local search`		`open greedy`		`open greedy and BFLP local search`	
Instance	**Budget**	**MIP gap after 20K s [%]**	**Time [s]**	Δ_{MIP} **[%]**	**Time [s]**	Δ_{MIP} **[%]**	**Time [s]**	Δ_{MIP} **[%]**	**Time [s]**	Δ_{MIP} **[%]**	**Time [s]**	Δ_{MIP} **[%]**
Instance I	0.9	1.33	128	-1.34	75	-	149	0.02	229	0.01	311	0.02
	0.8	2.36	115	-1.64	84	0.04	160	0.07	210	0.09	283	0.10
	0.7	3.53	102	-1.81	102	0.13	171	0.15	189	0.19	268	0.19
	0.6	5.07	90	-1.59	106	0.51	179	0.56	169	0.63	247	0.64
	0.5	7.49	76	-0.42	117	1.94	228	2.00	148	2.07	215	2.07
	0.4	6.66	63	-2.28	122	0.11	234	0.20	129	0.26	202	0.26
	0.3	7.42	50	-2.11	137	0.35	214	0.43	108	0.52	176	0.53
	0.2	7.71	36	-2.07	143	0.24	231	0.28	89	0.24	172	0.28
	0.1	7.84	24	-1.40	151	0.39	260	0.42	68	0.46	120	0.46
Instance II	0.9	0.57	3	-0.86	3	-	10	-	6	-	10	-
	0.8	1.41	3	-1.07	3	-0.02	7	-0.02	6	-	9	-
	0.7	2.19	3	-1.34	3	-0.06	7	-0.04	5	-0.02	9	-
	0.6	3.05	2	-1.70	3	-0.09	6	-0.04	5	-0.02	8	-0.02
	0.5	3.53	2	-1.67	3	-0.16	8	-0.04	5	-0.06	9	-
	0.4	3.93	2	-1.99	3	-0.08	7	-0.05	5	-0.14	8	-
	0.3	4.12	1	-1.64	4	-0.26	7	-0.12	4	-0.05	7	-
	0.2	4.64	1	-2.83	3	-0.09	6	-0.08	3	0.02	6	0.02
	0.1	4.18	1	-1.47	2	-0.22	6	-0.03	2	-0.03	4	-0.03
Instance III	0.9	0.28	673	-1.58	343	-0.12	641	-0.06	1316	-0.06	1848	-0.06
	0.8	0.63	612	-2.30	397	-0.07	697	-0.04	1197	-0.04	1724	-0.04
	0.7	1.12	567	-3.31	462	-0.04	757	0.02	1077	-0.01	1583	0.03
	0.6	1.72	489	-4.10	502	-	791	0.03	965	-0.03	1275	0.03
	0.5	3.17	434	-3.62	544	0.75	828	0.78	862	0.68	1162	0.75
	0.4	4.46	359	-3.44	587	1.26	858	1.28	743	1.12	1027	1.22
	0.3	11.92	315	4.32	610	8.34	867	8.36	648	8.09	916	8.22
	0.2	25.36	215	19.00	657	21.79	899	21.80	521	21.35	770	21.59
	0.1	18.08	141	12.30	654	13.96	910	13.98	410	13.42	667	13.77
Instance IV	0.9	0.97	28	-7.45	18	-0.17	40	-0.13	62	-0.29	86	-0.25
	0.8	2.75	25	-7.75	17	-0.11	40	-0.09	60	-0.17	96	-0.13
	0.7	5.25	22	-8.01	18	-0.17	41	-0.05	58	-0.06	85	-
	0.6	8.03	19	-7.71	17	-0.17	41	-0.10	59	-0.15	87	-0.11
	0.5	11.44	17	-6.58	20	0.33	43	0.39	57	0.50	79	0.50
	0.4	15.48	14	-3.86	19	1.92	41	2.00	53	2.00	80	2.06
	0.3	16.61	12	-4.70	19	0.31	41	0.31	49	0.45	71	0.52
	0.2	18.26	10	-2.54	26	0.44	47	0.55	40	0.56	69	0.71
	0.1	13.39	8	0.17	32	1.49	50	1.67	25	1.77	55	1.81

Data availability

All our codes and data are publicly available at https://github.com/Malena205/heuristics_quadratic_facility_location.

Conflict of interest

The authors declare that they have no conflict of interest.

Acknowledgements We thank Christian Schmitt and Theresia van Essen for their comments on parts of this work. The second author was partially supported by the Research Investment and Support Building Sustainable and Green Futures program of the University of Southampton. The funding source had no involvement in study design, collection, analysis and interpretation of data; in the writing of the report; and in the decision to submit the article for publication.

References

Addis et al.(2016)Addis, Aringhieri, Grosso, and Hosteins. Bernardetta Addis, Roberto Aringhieri, Andrea Grosso, and Pierre Hosteins. Hybrid constructive heuristics for the critical node problem. *Annals of Operations Research*, 238 (1–2):637–649, February 2016. doi: 10.1007/s10479-016-2110-y.

Bayerisches Landesamt für Umwelt(2015). Bayerisches Landesamt für Umwelt, editor. *Wertstoffhof 2020 - Getrennthaltungsgebot und Novelle des ElektroG*, UmweltSpezial, April 2015. URL https://www.bestellen.bayern.de/application/eshop_app000009?SID=62794461.

Benlic and Hao(2013). Una Benlic and Jin-Kao Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, January 2013. doi: 10.1016/j.amc.2012.10.106.

Costa et al.(2022)Costa, Mei, and Zhang. Joao Guilherme Cavalcanti Costa, Yi Mei, and Mengjie Zhang. Guided local search with an adaptive neighbourhood size heuristic for large scale vehicle routing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 213–221, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392372. doi: 10.1145/3512290.3528865.

Delft High Performance Computing Centre (DHPC)(2022). Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

Feldman et al.(1966)Feldman, Lehrer, and Ray. EFATL Feldman, FA Lehrer, and TL Ray. Warehouse location under continuous economies of scale. *Management Science*, 12(9):670–684, May 1966. doi: 10.1287/mnsc.12.9.670.

Jacobsen(1983). Soren Kruse Jacobsen. Heuristics for the capacitated plant location model. *European Journal of Operational Research*, 12(3):253–261, March 1983. doi: 10.1016/0377-2217(83)90195-9.

Karp(1972). Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi: 10.1007/978-1-4684-2001-2_9.

Kelly et al.(1998)Kelly, Maulloo, and Tan. Frank P. Kelly, Aman K. Maulloo, and David Kim Hong Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998. doi: 10.1038/sj.jors.2600523.

Krarup and Pruzan(1983). Jakob Krarup and Peter Mark Pruzan. The simple plant location problem: Survey and synthesis. *European Journal of Operational Research*, 12(1):36–81, January 1983. doi: 10.1016/0377-2217(83)90181-9.

Kuehn and Hamburger(1963). Alfred A. Kuehn and Michael J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, July 1963. doi: 10.1287/mnsc.9.4.643.

Lourenço et al.(2003)Lourenço, Martin, and Stützle. Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353. Springer US, Boston, MA, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/0-306-48056-5_11.

Martello(1990). Silvano Martello. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley & Sons, USA, June 1990. ISBN 0471924202. doi: 10.2307/2583458.

Mateus et al.(2010)Mateus, Resende, and Silva. Geraldo R. Mateus, Mauricio G. Resende, and Ricardo M. Silva. Grasp with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics*, 17(5):527–565, September 2010. doi: 10.1007/s10732-010-9144-0.

McKendall and Li(2016). Alan McKendall and Chihui Li. A tabu search heuristic for a generalized quadratic assignment problem. *Journal of Industrial and Production Engineering*, 34(3):221–231, November 2016. doi: 10.1080/21681015.2016.1253620.

Montes de Oca et al.(2012)Montes de Oca, Ner, and Cotta. Marco A. Montes de Oca, Ferrante Ner, and CarlosEditor Cotta. *Local Search*, chapter 3, page 29–42. Springer-Verlag, 2012. doi: 10.1007/978-3-642-23247-3_3.

Öncan(2007). Temel Öncan. A survey of the generalized assignment problem and its applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141, August 2007. doi: 10.3138/infor.45.3.123.

Osman(1995). Ibrahim H. Osman. Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17(4):211–225, December 1995. doi: 10.1007/bf01720977.

Pinedo(2016). Michael L. Pinedo. *Scheduling Theory, Algorithms, and Systems*, chapter 5, pages 113–151. Springer International, 5 edition, 2016. doi: 10.1007/978-3-319-26580-3.

Risanger et al.(2021)Risanger, Singh, Morton, and Meyers. Simon Risanger, Bismark Singh, David Morton, and Lauren Ancel Meyers. Selecting pharmacies for covid-19 testing to ensure access. *Health Care Management Science*, 24(2):330–338, January 2021. doi: 10.1007/s10729-020-09538-w.

Schmitt(2021). Christian Schmitt. Balancing preferential access and fairness with an application to waste management: mathematical models, optimality conditions, and heuristics. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, August 2021. URL https://www.researchgate.net/publication/355984329_Balancing_preferential_access_and_fairness_with_an_application_to_waste_management_mathematical_models_optimality_conditions_and_heuristics. (Accessed 13 April 2023).

Schmitt and Singh(2024). Christian Schmitt and Bismark Singh. Quadratic optimization models for balancing preferential access and fairness: Formulations and optimality conditions. *INFORMS Journal on Computing*, February 2024. ISSN 1526-5528. doi: 10.1287/ijoc.2022.0308.

Sridharan(1995). R. Sridharan. The capacitated plant location problem. *European Journal of Operational Research*, 87(2):203–213, December 1995. doi: 10.1016/0377-2217(95)00042-o.

Wolfe(1959). Philip Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, July 1959. doi: 10.2307/1909468.