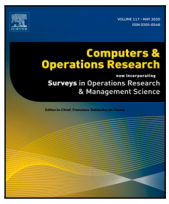
Computers & Operations Research 183 (2025) 107153

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



An effective aggregation heuristic for Capacitated Facility Location Problems with many demand points

R.J.W. Buijs [a,b,*], R.D. van der Mei [a,b], E.R. Dugundji [a,c], S. Bhulai [a,b]

[a] Centrum Wiskunde en Informatica, Sceince Park 123, 1098 XG Amsterdam, Netherlands
[b] Vrije Universiteit, De Boelelaan 1105, 1081 HV, Amsterdam, Netherlands
[c] Massachussetts Institute of Technology, CTL, 1 Amherst St, Cambridge, MA 02142, USA

ARTICLE INFO

Keywords:
Capacitated Facility Location Problem
Aggregation
Matheuristic
Reverse logistics

ABSTRACT

In location analysis, the effects of demand aggregation have been the subject of many studies. This body of literature is mainly focused on p-median and p-center problems. Relatively few papers in the literature on aggregation explicitly concern the Capacitated Facility Location Problem **(CFLP)**. Our work examines the beneficial use of aggregation in the context of the CFLP. We focus on problems where there are significantly more demand points than potential facility locations, since this is where aggregation is most applicable in reducing complexity. We examine ways to obtain an aggregation at a fixed resolution, that is likely to perform well for a given instance of the problem. These aggregation techniques will form the core of a broader algorithmic framework, which contributes to the literature concerning heuristics for CFLPs. Our core aggregation method is based on applying k-means clustering in $\mathbb{R}^m$, where m is the number of potential facilities. The space in which we apply the clustering is constructed by applying a transformation to the normalized distance matrix corresponding to the original CFLP problem. The aim of applying the transformation is to magnify differences in distance where relevant, and to compress irrelevant differences in distance. We evaluate our heuristic method on larger instances based on a real-world problem in reverse logistics. The results are encouraging and indicate that our method is capable of outperforming an intuitive benchmark aggregation method. We find that choosing the right hyperparameters and starting with a good initialization help our method perform better.

1. Introduction

Location problems emerge in many contexts, such as supply chain design (da Gama, 2022), power systems (Lee, 2022) and public service planning (Celik Turkoglu and Erol Genevois, 2020). As the availability of detailed data increases, these problems may become very large and hard to solve. Aggregation has been extensively studied as a means of data manipulation to reduce problem complexity for discrete facility location problems. The motivation for using a demand aggregation method to solve these problems is to be able to obtain near-optimal solutions to these larger instances relatively fast, while using an intuitive and straightforward technique. By producing a smaller problem that is actually solved to optimality, a sense of accuracy based on the level of fine-grainedness of this smaller problem is also provided. The fact that there is no clear, unambiguous answer to the question how to best obtain an aggregated problem from the original problem, has led to aggregation being a topic of research in location science for decades.

Demand aggregation is especially relevant when the resolution of customer data is very detailed. A good example of this is locating public facilities to which an entire population or demographic may require access. In these settings, the locational problems at hand are often p-median and p-center problems. The choice between using a p-median or a p-center objective may depend on the locational goal of the modeler (Bach, 1981), but both models require the selection of p points as the locations for (identical) facilities. The topic of demand aggregation is thus mainly discussed in scientific literature for these two problem classes.

However, locational problems with many demand points are not limited to p-center or p-median settings: In contexts such as waste management, or warehouse location for (very) large customer bases, the Capacitated Facility Location Problem (CFLP) is more applicable. Although these cases may be more limited compared to those for p-median and p-center models, they still provide an interesting motivation for studying CFLPs with many demand points as well, despite the CFLP not being the focus of most of the aggregation-related literature.

* Corresponding author at: Centrum Wiskunde en Informatica, Sceince Park 123, 1098 XG Amsterdam, Netherlands.
E-mail addresses: rjwb@cwi.nl (R.J.W. Buijs), mei@cwi.nl (R.D. van der Mei), elenna_d@mit.edu (E.R. Dugundji), s.bhulai@vu.nl (S. Bhulai).

https://doi.org/10.1016/j.cor.2025.107153
Received 31 March 2024; Received in revised form 22 May 2025; Accepted 26 May 2025
Available online 18 June 2025

We remark here that there exist other reasons and settings for which it makes sense to consider aggregation, most notably the lack of accurately specific data. We will, however, focus specifically on the setting where a large location problem with many data points has to be simplified in order to obtain a good solution within a reasonable amount of time. Note that as we will not attempt to resolve any of the complexity arising from having a large number of facilities, we restrict our approach to problem settings where the number of demand points is much larger than the number of potential facilities to choose from. This ensures that complexity can be sufficiently reduced by focusing on the demand points only. Bearing this in mind, the **contribution of this paper is as follows:**

- We will study aggregation in the context of Capacitated Facility Location Problems **(CFLP)**, a problem that is addressed relatively little in aggregation-related literature;
- We will focus on minimizing the actual optimality gap between the original problem and the aggregated version, thus allowing problem-specific knowledge to be used in our aggregation schemes instead of applying a general, error bound-based scheme to each encountered problem instance, and demonstrate the effectiveness of this approach;
- We will place the presented aggregation techniques in a broader algorithmic framework, adding to the body of existing heuristics for large capacitated facility location and network design problems.

The remainder of the paper is structured as follows: We will go over the CFLP in Section 2, followed by a discussion of related literature in Section 3. In Section 4, we will discuss the motivation and theoretical background behind our methodology, after which we will present our core aggregation framework. In Section 5, we argue how to approach the topic of parameter selection within our aggregation paradigm. Consequently, we evaluate our methods by applying them to various instances of the CFLP and discuss our results in Section 7. In Section 8, we provide additional analysis to assess the potential of our method for application to other location problems. Finally, we present our conclusions in Section 9.

2. The capacitated facility location problem

The CFLP is a location problem where one may open any number out of m potential facilities to fulfill the demand arising at n demand points. The intent is to minimize the costs associated with having open facilities and transporting goods from the facility to the customer. This problem is NP-hard (Cornuejols et al., 1990). The CFLP can be described in the form of a Mixed Integer Linear Program (MILP) P as can be seen in Eqs. (1) to (5). The objective (Eq. (1)) shows that we are minimizing facility cost and transportation cost over a set of facility opening variables $\boldsymbol{y} \in \{0,1\}^m$ and demand allocation variables $\boldsymbol{x} \in \mathbb{R}_+^{m \times n}$. The decision variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are constrained to meet demands exactly (Eq. (2)) while not exceeding facility capacity (Eq. (3)). In the broadest sense, the parameters d_{ij} describe the cost associated with fulfilling demand from demand point j by facility i. For this paper, we assume that d_{ij} is finite for all i and j. This means that there are no absolute restrictions on which demand point is served by which facility. Moreover, we assume that these parameters represent a transportation cost and that they are proportional to some distance in the real world.

$$\min_{\boldsymbol{x},\boldsymbol{y}} z = \sum_{i\in[m]} \sum_{j\in[n]} D_j d_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i \tag{1}$$

subject to

$$\sum_{i\in[m]} x_{ij} = 1 \qquad \forall\, j \in [n] \tag{2}$$

$$\sum_{j\in[n]} x_{ij} D_j \le Q_i y_i \qquad \forall\, i \in [m] \tag{3}$$

$$0 \le x_{ij} \le 1 \qquad \forall\, i \in [m],\ j \in [n] \tag{4}$$

$$y_i \in \{0,1\} \qquad \forall\, i \in [m] \tag{5}$$

The CFLP can be solved to optimality by applying a solution algorithm tailored to solving MILPs, but because of its NP-hardness, finding optimal solutions for large instances of the CFLP is not computationally tractable.

3. Related literature

In this section, we will review the related literature. Our review focuses on three subdomains: Analysis of aggregation errors, aggregation-based solution techniques and algorithms and heuristics for solving the CFLP. We would like to remind the reader that most of the works on aggregation in location modeling that we review here also cover p-median and (to a lesser extent) p-center problems. For a more detailed description and analysis of these problems, we refer the reader to the chapters on p-median and p-center problems by Marín and Pelegrín (2019) and Çalık et al. (2019), respectively.

3.1. Analysis of aggregation errors

One of the first papers that addresses the error induced by aggregating all demand in a single area into a single point is the work of Hillsman and Rhoda (1978). They introduce three types of aggregation errors that emerge when aggregated data is used to solve location problems. Two of these, called source A and source B errors, are caused by approximating the weighted distance of a facility to all of the demand in an area by the distance of a single representative point. They also introduce the source C error, which arises from the possibility that for some demand points, the nearest facility changes by aggregating. Current and Schilling (1987) proposed a ethod to eliminate source A and B errors in these problems, which ensures that the optimal objective of the aggregated problem is bounded below by the optimal objective of the original problem. Later, a method of eliminating source C error when the disaggregated demand is not known exactly was proposed by Hodgson and Neuman (1993). However, minimizing aggregation-induced errors is far from a trivial problem. In fact, Francis and Lowe (1992) have shown that, for p-median and p-center problems, the problem of obtaining an aggregation that is generally optimal (minimizing the worst-case error, which does not allow for the exploitation of any structural properties of the problem instance) is of the same type as the original location problem. This is referred to as the *aggregation paradox*.

For examples of work relating to aggregation error analysis in specific contexts, we refer the reader to Goodchild (1979) (analysis of the effect of aggregation error on the problem solution), Bach (1981) (aggregation and distance measures), Erkut and Bozkaya (1999) (error analysis of different aggregation methods, among which planar k-means clustering), and Hodgson and Hewko (2003) (analysis of the trade-off between aggregation and surrogation error). An extensive overview of the mathematical aspects of aggregation and the different characterizations that can be used for errors induced by this practice in location modeling, as well as a literature survey on this topic, can be found in Francis et al. (2009).

3.2. Aggregation-based solution techniques for location problems

The fact that the task of finding the 'best' aggregation in general is at least as hard as the original problem, does not mean that aggregation cannot or should not be used as a tool to solve large or complex location problems. In fact, many aggregation-based methods have been proposed to solve different location problems. We again remark that the p-median problem was the context in which most of the methods that we will review here have been developed. Such methods can be developed with different aims in mind. One approach is to aim for

a minimal worst-case performance. Examples include Francis et al. (1996), Andersson et al. (1998), and the multi-pattern tiling algorithms introduced by Qi and Shen (2010), which are based on the honeycomb heuristic of Papadimitriou (1981). There are also works in which the proposed aggregation method is more practically driven, and where the proposed method has some desirable traits that suit the context in which it is applied, instead of a possible worst-case application. Examples include Avella et al. (2012) and Irawan and Salhi (2013). In both of these works, aggregation is applied in the context of a broader algorithmic framework. Another more recent example is the quadtree-based method by Salhi and Irawan (2015). In this work, the authors arrive at an aggregation from the base that is best for a known (sub)set of potential facilities, acknowledging that finding an optimal general aggregation is not possible due to the aggregation paradox. Another study that has this algorithmic nature is that of Cebecauer and Buzna (2017). They have developed an iterative aggregation algorithm, in which the aggregation is updated at each iteration in order to minimize error. The method has been applied to p-median problems of different scales using data from Slovakia. For a more extensive review of aggregation-based methods for several location problems, we refer the reader to the survey by Irawan and Salhi (2015).

3.3. Algorithms and heuristics for solving the Capacitated Facility Location Problem (CFLP)

In the abundance of aggregation-related literature, relatively few works explicitly state the CFLP. To the best of our knowledge, the work on aggregation-based methods for this problem is limited to the paper by Sankaran (2007). In this paper, an aggregation heuristic is proposed that is based on iteratively collapsing individual demand points into a single point, while keeping the increase in a derived error bound below a certain level, until this is not possible anymore. In this approach, the maximal error has to be specified beforehand, and the number of aggregation zones required (and thus the computational complexity of the resulting problem) are an outcome of the iterative process. Despite the fact that aggregation has not been used as a prominent solution technique for CFLPs, the problem has been approached from many angles. We will review the most important directions for the heuristic methods that have been proposed in the literature here. It is known that methods based on Lagrangian Relaxation can provide lower bounds, for instance, of the CFLP, while also yielding approximate solutions (see, e.g., Marín and Pelegrín, 1999). Among the popular solution methods, we also find metaheuristic approaches. Basu et al. (2014) give a good overview of how these approaches are used for solving various types of location problems, among which the CFLP. Liao and Guo (2008) use a heuristic based on an adaptation of k-means clustering directly to find good solutions to capacitated facility location problems.

Guastaroba and Speranza (2012) showed that kernel search, a matheuristic, could reproduce optimal results for CFLPs with a large number of potential facilities, or find new good approximate solutions. This method could be considered a reduction-based method as well: The kernel search algorithm aims to find good solutions by sequentially solving a problem containing only a subset of the original variables. Another matheuristic based on the premise of size reduction was presented for the related p-median problem by Stefanello et al. (2014b). Besides these, a few other methods that are based on reducing the model and using an easier counterpart in the solution technique have been proposed: In their branch & bound algorithm, Nauss (1978) use a condition derived by Akinc and Khumawala (1977) that allows some facilities to be fixed open without changing the optimum, which comes down to reducing the number of binary decision variables in the (sub)problem. Avella et al. (2009) made use of a constructed core problem, which contained only a subset of facility and allocation variables, selected based on Lagrangian reduced cost. Finally, Jain and Vazirani (2001) presented a constant-factor approximation algorithm for the metric Soft Capacitated Facility Location Problem, a variant of the metric CFLP where each facility can be opened multiple times. Their approach reduces the problem to an uncapacitated facility location problem, for which better constant-factor approximation algorithms had already been established.

The contribution by Jain and Vazirani (2001) also contributes to another line of work, which is dedicated to developing approximation algorithms for the CFLP. For an overview of approximation algorithms for the CFLP and related problems, we refer the reader to Aardal et al. (2015). More recently, An et al. (2017) were the first to develop a formulation of the CFLP that has a constant integrality gap. They used this formulation and LP-rounding to devise a constant-factor approximation algorithm.

4. Methodology

In this section, we will address the technical background of aggregation in the context of the CFLP. We analyze the effects of aggregation on the solutions of the CFLP. We provide an instance-specific upper bound for the optimality error that is induced by applying aggregation. Finally, we outline our core aggregation framework, which is based on our analysis.

4.1. Theoretical background

When aggregating demand in the context of location models, we usually map the set of demand points to a new, much smaller set of demand points in order to reduce the size and complexity of a problem: The aggregated location problem can be modeled using a smaller MILP $\tilde{P}$. A valid and straightforward approach here would be to aim for minimizing the weighted distance from demand points $j \in [n]$ to their mapping, $M(j) \in K$, $|K| < n$. Francis et al. (2000) have shown that this weighted distance is, in fact, an error bound for a class of location models called SAND models. A way to avoid underestimation of the objective function by $\tilde{P}$ (using Current and Schilling, 1987's approach to mitigate source A and B errors) is to assign a set of weighted average distances to each aggregated demand point (ADP). Rather than treating the ADP as some point with a physical location, it can be viewed as an aggregation over the distances of the points that are mapped to it. Minimizing the difference in distance between demand points (DPs) and their ADPs, instead of minimizing the distance of a DP to some physical ADP location, allows for the use of aggregation in problems where distances are non-Euclidean. Our simplified problem $\tilde{P}$ is equivalent to the original problem where $\boldsymbol{d}$ is replaced by $\tilde{\boldsymbol{d}}$. Here we define $\tilde{d}_{ij}$ as the determined distance between $M(j)$ and facility i. We denote this problem by $P_{\tilde{\boldsymbol{d}}}$. We can infer about how we should construct $\tilde{P}$ by comparing P and $P_{\tilde{\boldsymbol{d}}}$, as the latter is equivalent to $\tilde{P}$. We will first introduce some notation. All of the notation that will be introduced refers to the objective z (Eq. (1)), and is thus subject to (2) to (5).

Firstly, we can express the objective value of problems P and $P_{\tilde{\boldsymbol{d}}}$ (where the distances are given by $\boldsymbol{d}$ and $\tilde{\boldsymbol{d}}$, respectively) as a function of $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$z_{\boldsymbol{d}}(\boldsymbol{x},\boldsymbol{y}) := \sum_{i\in[m]}\sum_{j\in[n]} D_j d_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i, \tag{6}$$

$$z_{\tilde{\boldsymbol{d}}}(\boldsymbol{x},\boldsymbol{y}) := \sum_{i\in[m]}\sum_{j\in[n]} D_j \tilde{d}_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i. \tag{7}$$

Furthermore, we introduce the shorthand notation $z_{\boldsymbol{d}}(\boldsymbol{y})$ and $z_{\tilde{\boldsymbol{d}}}(\boldsymbol{y})$ that express the objective of P and $P_{\tilde{\boldsymbol{d}}}$ only in terms of the binary facility location vector $\boldsymbol{y}$, where we choose for $\boldsymbol{x}$ the allocation that minimizes transportation cost given $\boldsymbol{y}$. Finding this allocation and the corresponding objective can be achieved by solving an LP with respect to $\boldsymbol{x}$.

$$z_{\boldsymbol{d}}(\boldsymbol{y}) := \min_{\boldsymbol{x}} \sum_{i\in[m]}\sum_{j\in[n]} D_j d_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i, \tag{8}$$

$$z_{\tilde{\boldsymbol{d}}}(\boldsymbol{y}) := \min_{\boldsymbol{x}} \sum_{i\in[m]}\sum_{j\in[n]} D_j \tilde{d}_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i. \tag{9}$$

Along these lines, we also define $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y})$ and $\boldsymbol{x}^*_{\tilde{\boldsymbol{d}}}(\boldsymbol{y})$ denoting that allocation $\boldsymbol{x}$, that minimizes transportation cost given $\boldsymbol{y}$.

$$\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}) := \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i\in[m]}\sum_{j\in[n]} D_j d_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i, \tag{10}$$

$$\boldsymbol{x}^*_{\tilde{\boldsymbol{d}}}(\boldsymbol{y}) := \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i\in[m]}\sum_{j\in[n]} D_j \tilde{d}_{ij} x_{ij} + \sum_{i\in[m]} c_i y_i. \tag{11}$$

We will also introduce a specific notation for the optimal locational decision in the problems P and $P_{\tilde{\boldsymbol{d}}}$.

$$\boldsymbol{y}^*_{\boldsymbol{d}} := \underset{\boldsymbol{y}}{\operatorname{argmin}}\, z_{\boldsymbol{d}}(\boldsymbol{y}), \tag{12}$$

$$\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}} := \underset{\boldsymbol{y}}{\operatorname{argmin}}\, z_{\tilde{\boldsymbol{d}}}(\boldsymbol{y}). \tag{13}$$

Finally, we introduce notations for three different flavors of optimal objectives: $z^*_{\boldsymbol{d}}$ for the objective of optimal location decision $\boldsymbol{y}^*_{\boldsymbol{d}}$ in the original problem, $\tilde{z}^*_{\tilde{\boldsymbol{d}}}$ for the objective of aggregate-optimal location decision $\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}$ in the aggregated problem, and finally $z^*_{\tilde{\boldsymbol{d}}}$ for the objective of the aggregate-optimal location decision in the original problem.

$$z^*_{\boldsymbol{d}} := z_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}}), \tag{14}$$

$$\tilde{z}^*_{\tilde{\boldsymbol{d}}} := z_{\tilde{\boldsymbol{d}}}(\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}) \tag{15}$$

$$z^*_{\tilde{\boldsymbol{d}}} := z_{\boldsymbol{d}}(\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}). \tag{16}$$

Since we are interested in the performance of the aggregate-optimal solution under the actual locational distribution of the demand, we aim to minimize the problem-specific optimality gap $z^*_{\tilde{\boldsymbol{d}}} - z^*_{\boldsymbol{d}}$. This optimality gap is always nonnegative due to the way $\tilde{\boldsymbol{d}}$ is constructed.

An upper bound (UB) to this gap is provided by $z_{\tilde{\boldsymbol{d}}}(\boldsymbol{y}^*_{\boldsymbol{d}}) - z_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})$. This UB coincides with the optimality gap when the UB is 0. This value can theoretically be achieved if the number of ADPs is large enough. The UB reduces to

$$\sum_{i\in[m]}\sum_{j\in[n]} D_j(\tilde{d}_{ij} - d_{ij})\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}. \tag{17}$$

This is equivalent to an intermediate result given by Sankaran (2007), who uses this result to further derive an upper bound that is problem-specific, but does not depend on the optimal solution $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}$ of the original problem.

4.2. Core aggregation framework

A schematic overview of our core aggregation method is given in Fig. 1. Our method is centered around shifts in the distance matrix domain (rather than directly minimizing distances in the Cartesian plane). It consists of the following five steps:

1. Application of decay function (Section 4.2.2)
2. Clustering (Section 4.2.3)
3. Computing aggregated distance vectors (Section 4.2.4)
4. Solving the aggregated MILP (Section 4.2.4)
5. Solving the allocation problem (Section 4.2.5)

In the remainder of this section, we will first discuss why we propose looking at differences in the distance matrix instead of looking at differences in planar location, after which we introduce our core method in a stepwise manner. Fig. 3(a) shows an example of a CFLP instance. This small instance, consisting of three facilities and thirteen demand points (with unit demand), will be used to demonstrate our methodology in a stepwise manner.

4.2.1. On the domain of the distance matrix space and minimizing shifts

In our example setting, each demand point j is described using its Cartesian coordinates, and the distance $d_{jj'}$ between two points i and j in this plane is given by the Euclidean distance formula. As discussed, an aggregation procedure could be based on the premise of minimizing the weighted sum of these distances between the original demand points j and their respective aggregated mappings j'. However, since our UB (17) is defined as a sum of differences between distances rather than the distances to some representative point, our method focuses on the shift in these distances instead. We denote the total demand at ADP k by $D_k = \sum_{j:M(j)=k} D_j$. Note that by means of construction, for a single ADP k, $d_{ik} = \frac{\sum_{j:M(j)=k} D_j d_{ij}}{D_k}$. Also, because of the equivalence relation between the smaller MILP $\tilde{P}$ and the surrogate MILP $P_{\tilde{\boldsymbol{d}}}$, we have that $\tilde{d}_{ij} = d_{ik} \forall j : M(j) = k, \forall i$. For any given ADP k, the UB contribution of distance shifts that we can attribute to this ADP is equal to

$$\sum_{j:M(j)=k}\sum_{i\in[m]} D_j(\tilde{d}_{ij} - d_{ij})\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}, \tag{18}$$

where $\tilde{d}_{ij} = d_{ik}$. We observe that the contribution will be equal to zero if, for all $i \in [m]$, the term $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}$ consists of only zeros or ones for the indices $j : M(j) = k$. We can show that the contribution of any ADP k to the UB is always nonnegative. Moreover, let H_k be the set of facilities for which the condition mentioned above does not hold:

$$H_k = \{i \in [m] : 0 < \sum_{j:M(j)=k} \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} < |\{j : M(j) = k\}|\}. \tag{19}$$

Proposition 1. *Given an ADP k, the upper bound for the total error that can be attributed to this ADP is equal to*

$$\frac{1}{2D_k}\sum_{i\in H_k}\sum_{j:M(j)=k}\sum_{j':M(j')=k} D_j D_{j'}(\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'})(d_{ij'} - d_{ij}) \tag{20}$$

Note that for larger ADPs, the summation in Eq. (20) will contain many zero terms. We can be more precise by enumerating only the terms corresponding to tuples (i, j, j') for which $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}$ is nonzero. Let us first define this set of tuples G_k for a given ADP k:

$$G_k = \{(i, j, j') : i \in H_k, j, j' \in [n] | M(j) = k, M(j') = k, \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'} \neq 0\} \tag{21}$$

Eq. (20) can now be simplified as follows:

$$\frac{1}{2D_k}\sum_{(i,j,j')\in G_k} D_j D_{j'}(\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'})(d_{ij'} - d_{ij}) \tag{22}$$

Without loss of generality, let us assume that for every tuple (j, j') there are exactly two corresponding elements in G_k: (i, j, j') and (i', j, j').

Proposition 2. *Suppose that for every tuple (j, j') there are exactly two corresponding elements in G_k: (i, j, j') and (i', j, j'). If (i, j, j') and (i', j, j') are two elements of G_k, the contribution to the error UB corresponding to these two elements is proportional to and increasing in $|(d_{i'j} - d_{i'j'}) - (d_{ij} - d_{ij'})|$, D_j and $D_{j'}$.*

The proofs for Propositions 1 and 2 can be found in the Appendix A.

Proposition 3. *The pairwise error-bound contribution presented in Proposition 2 is bounded below by 0 and above by $|(d_{i'j} - d_{i'j'})| + |(d_{ij} - d_{ij'})|$, and both bounds are tight.*

Proof. The lower bound trivially follows from the fact that the contribution is an absolute value. The upper bound trivially follows when both terms in the expression have different signs. An example of tight bounds in Euclidean space is when both demand points and facilities are on a straight line: If the facilities are on the same end of the line compared to the demand points, the expression attains value 0, and if both facilities are on opposite ends with the demand points in between, the upper bound is attained. □

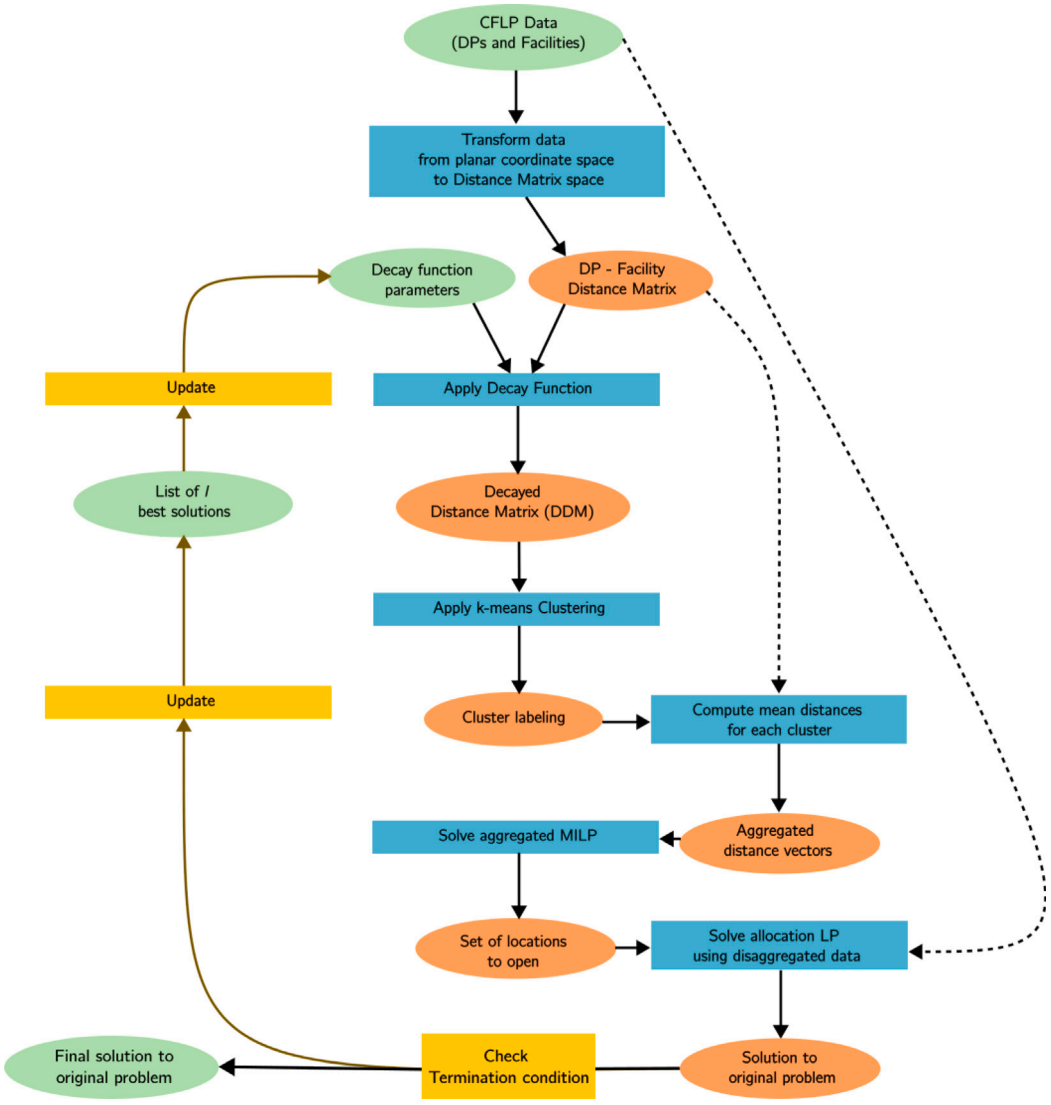


Fig. 1. A schematic overview of our core aggregation method.

By substituting the upper bound from Proposition 3 into Eq. (22), we obtain the following weaker, but more practical upper bound on the introduced error for ADP k:

$$\frac{1}{2D_k} \sum_{(i,j,j') \in G_k} D_j D_{j'} |d_{ij'} - d_{ij}| \tag{23}$$

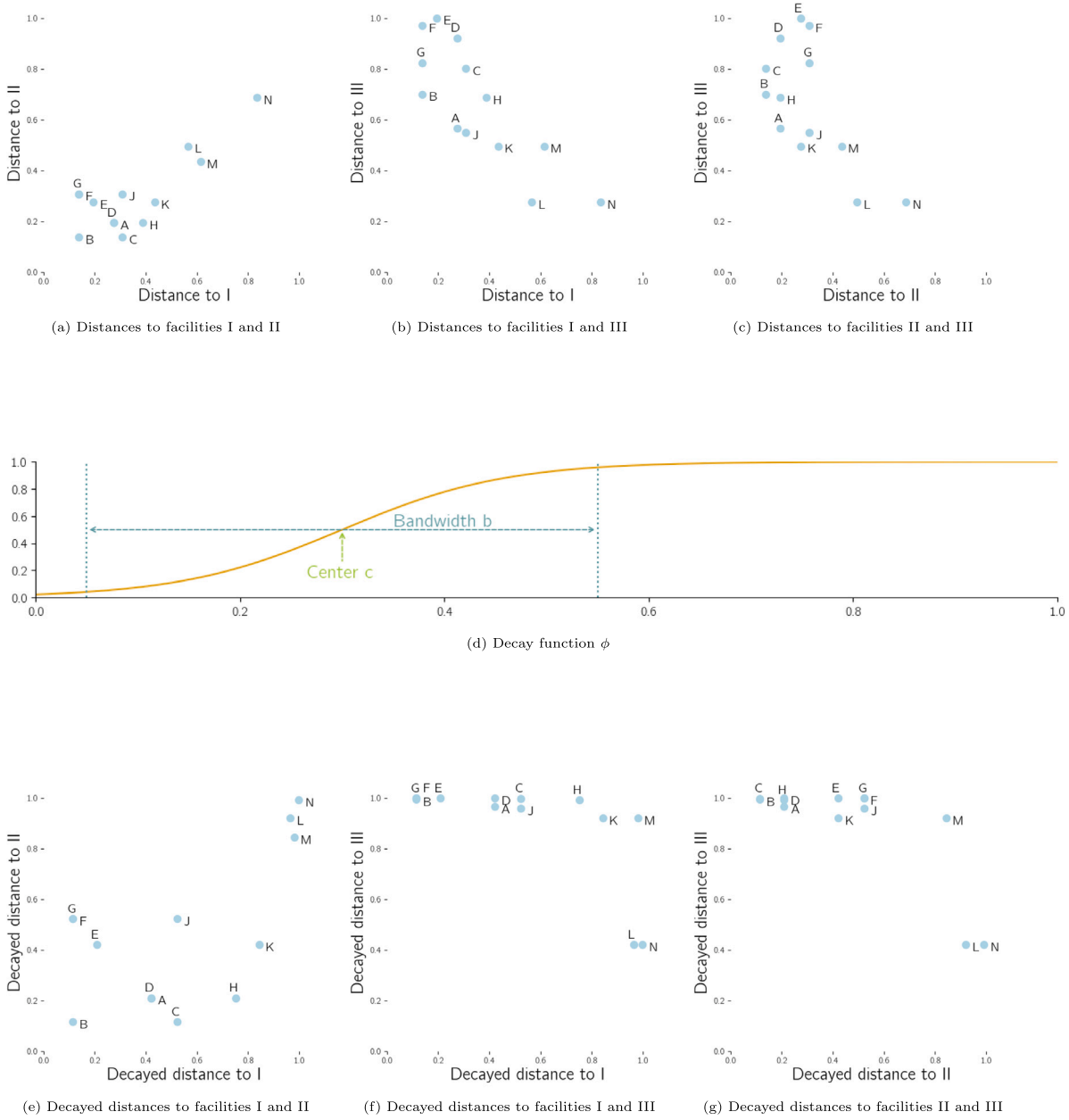
Moreover, we remark that there generally is a negative relation between $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}$ and d_{ij}. More specifically, one can expect that the higher the biggest deviation of d_{ij} from $\tilde{d}_{ij}$ in an ADP, the more likely it is that $H_k \neq \emptyset$. This should be avoided as much as possible, as these are the ADPs that contribute to the UB. When attempting to find an aggregation for which the mentioned UB is low, this would give an incentive to put more weight on penalizing the largest weighted differences $D_j D_{j'} |d_{ij} - d_{ij'}|$ in distance.

Given this information, we can still view this as a task of minimizing the distance between points, albeit not in the original $\mathbb{R}^2$ space of the locations depicted in Fig. 3(a), but rather in the $\mathbb{R}^m$ space where each point is characterized by its (normalized) distance d_{ij} to each of the m potential facilities. Figs. 2(a) to 2(c) show the two-dimensional slices of the location of the demand points from Fig. 3(a) in their three-dimensional distance matrix space. Note that in many instances of CFLP, the distances used between demand points and facilities are not necessarily Euclidean, nor are the locations always given in some coordinate system. The distance between each demand point and facility is always specified, however, so working in the distance matrix space is generally possible for every possible CFLP instance.

4.2.2. Application of decay function

We have seen in the previous section that minimizing distance shifts will help to ensure that the UB attains lower values. However, our previous analysis also suggests that only specific shifts near the 'allocation boundary' of the optimal facilities $\boldsymbol{y}^*_{\boldsymbol{d}}$ are really of interest for minimization. As we do not know $\boldsymbol{y}^*_{\boldsymbol{d}}$, $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y})$ is unknown, and we cannot minimize these specific shifts directly. We can, however, apply a transformation to make sure that some shifts are prioritized (i.e., penalized heavier) rather than treating all distance shifts equally. The motivation for doing this is similar to the idea underlying variable elimination in size reduction heuristics, such as the one Stefanello et al. (2014b) introduced to solve the capacitated p-median problem, which is very similar to the CFLP.

The choice for a good transformation is not obvious. One could consider a general elementwise transformation mapping each distance in the normalized distance matrix to a new value on the same domain ($\phi : [0,1] \rightarrow [0,1]$). An alternative approach would be to use a parameterized function that may be uniquely defined for each (i,j) pair, and applied elementwise to each distance matrix entry ($\{\phi_{i,j} : [0,1] \rightarrow \mathbb{R}^+; i \in [m], j \in [n]\}$). In our illustrative example, we apply a



(a) Distances to facilities I and II (b) Distances to facilities I and III (c) Distances to facilities II and III

(d) Decay function ϕ

(e) Decayed distances to facilities I and II (f) Decayed distances to facilities I and III (g) Decayed distances to facilities II and III

Fig. 2. 2-dimensional slices of the normalized distance matrix representation of the demand points from the example of Fig. 3(a), before and after application of decay function ϕ.

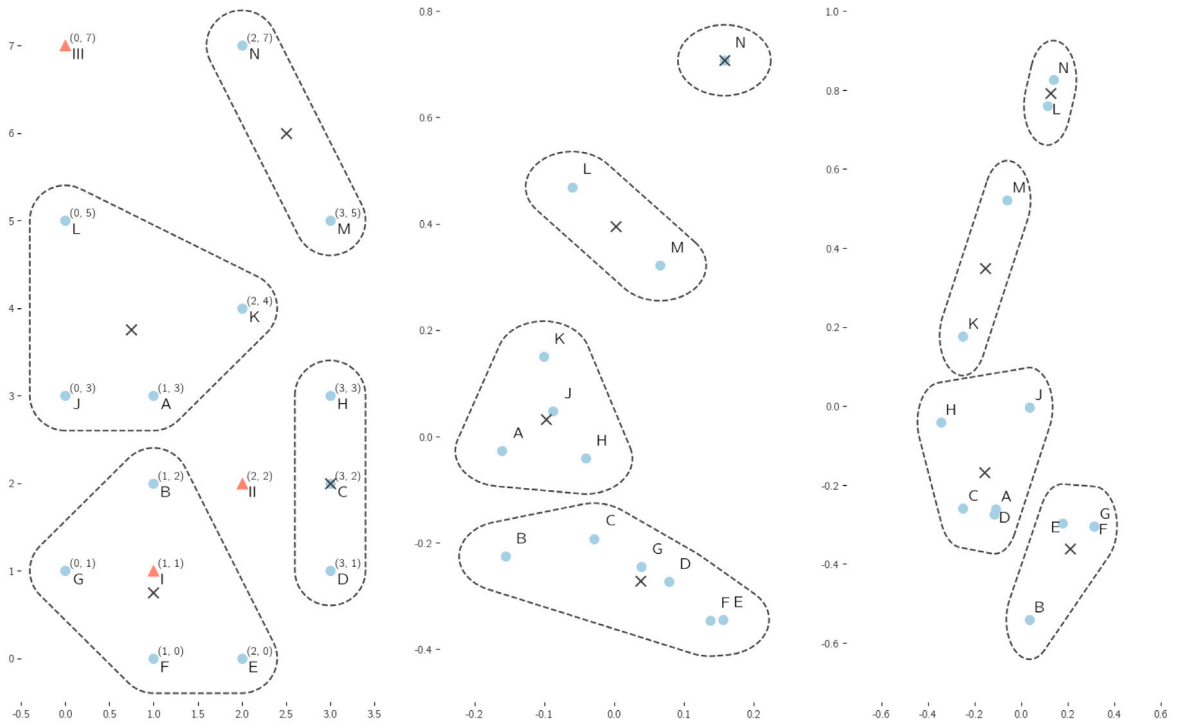
single transformation ϕ that allows us to account far less for distance shifts occurring outside of a certain 'critical' domain. For example, we are not too concerned if a distance is shifted from 0.7 to 0.95 (e.g., because we expect that the corresponding allocation variable x_{ij} will quite confidently be 0 in both cases), but a shift from 0.15 to 0.25 might be very undesirable (e.g., because we expect that the allocation variable x_{ij} takes on both values 0 and 1 in this region). To this end, we propose a function of the following form:

$$\phi(x) = \frac{e^{(x-\alpha)\beta}}{1 + e^{(x-\alpha)\beta}}. \tag{24}$$

Here, α and β can be chosen such that all differences outside a certain bandwidth will 'decay' to a small value. An example of such a function, with a bandwidth of 0.5, centered around 0.3, is depicted in Fig. 2(d). The resulting mapping of the points from Figs. 2(a) to 2(c) is shown in Figs. 2(e) to 2(g). After applying this mapping to the normalized distance matrix, we obtain a Decayed Distance Matrix (DDM).

4.2.3. *Clustering*

Having obtained the DDM, we can apply a clustering algorithm to minimize the decayed distance shift errors. In principle, any weighted clustering algorithm could be applied to such a task, depending on the exact metric to minimize in terms of the decayed shift errors. In the context of constructing $\tilde{P}$ are looking to partition all m DPs into k ADP clusters, with $k < m$. We apply the greedy k-means ++ algorithm (Arthur et al., 2007) as implemented in the `scikit – learn` package in Python (Pedregosa et al., 2011) in this paper, using the demands of the DPs as weights. In addition to speed and scalability considerations, this algorithm fits the task at hand since its objective is to minimize the ℓ_2-norm of the distance shifts. Another option would be to perform a k-medians clustering, which punishes large decayed shifts less severely, but regardless of the question which objective is more desirable to minimize in our setting, the method is less computationally tractable for large datasets, and thus less well-fitted for our purpose. Using the



(a) planar k-means

(b) k-means on non-decayed normalized distance matrix, depicted using PCA on the distance matrix

(c) k-means on normalized distance matrix after applying decay function ϕ (Section 4.2.3), depicted using PCA on the DDM

Fig. 3. Results of different clustering methods.

obtained clustering, we have found a mapping $M : [n] \to K$ of demand points to ADPs, which can be used to obtain a reduced instance of the CFLP with only k demand points. The resulting clustering on our example instance is shown in Fig. 3(c), and we can note the differences compared to applying k-means directly in the plane, and to applying our approach without effectively applying a decay function (i.e., $\phi(x) = x$), which are shown in Figs. 3(a) and 3(b). We remark here that k-means clustering in the plane intends to minimize the weighted sum of squared Euclidean distances. In the absence of extreme outliers, the resulting aggregation will be one in which the sum of weighted distances from the DPs to the centroid of the ADP is also low.

4.2.4. Computing aggregated distance vectors and solving the aggregated MILP

Having obtained k ADPs to use as a substitute for our initial m demand points, the next step is to obtain a smaller instance of CFLP using the ADPs as demand points. We can compute the new demand at the ADPs using $D_k = \sum_{\{j:M(j)=k\}} D_j$. As described in Section 4.1, the distances d_{ik} between facility i and each ADP k can be represented in the aggregated variant of the MILP in Eqs. (1) to (5) using the weighted mean of the distances of the DPs that are mapped to k:

$$d_{ik} = \frac{\sum_{\{j:M(j)=k\}} D_j d_{ij}}{D_k}.$$

Substituting the ADPs for the original DPs in the MILP P presented in Section 2, yields a new MILP $\tilde{P}$, with (collapsed) variables $\tilde{\boldsymbol{x}} \in \mathbb{R}_+^{k \times m}$ and $\tilde{\boldsymbol{y}} \in \{0,1\}^m$, which can be solved using an exact MILP-solving algorithm to obtain the optimal solution $(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{y}}^*)$ to this problem.

4.2.5. Solving the allocation problem

After having solved the aggregated problem, we have obtained a pseudo-optimal facility opening vector $\tilde{\boldsymbol{y}}^*$, as well as an allocation $\tilde{\boldsymbol{x}}^*$ of ADP demand to the open facilities. The associated optimal objective value of $\tilde{P}$ is $\tilde{z}$. Consider $(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{y}}^*)$, the optimal solution of $\tilde{P}$. From this, it is easy to construct a corresponding solution to the original MILP P. This solution, the optimal solution of $P_{\tilde{\boldsymbol{d}}}$, is obtained as follows: $x^*_{\tilde{\boldsymbol{d}}ij} = \tilde{x}^*_{iM(j)}$ for each $j \in [n]$, and $\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}} = \tilde{\boldsymbol{y}}^*$. However, this solution is not always equal to the actual objective associated with $\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}$. If we decide to implement solution $\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}$, the associated transportation costs will be those that are optimal under the original distance matrix. This means that the associated allocation decision is $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}})$ and the actual objective for this solution is indeed $z^*_{\tilde{\boldsymbol{d}}}$ $(= z_{\boldsymbol{d}}(\boldsymbol{y}^*_{\tilde{\boldsymbol{d}}}))$.

5. Parameter selection

In order to get insight into how the core method performs and to evaluate its strengths and weaknesses, we performed some computational experiments on toy problems. In Fig. C.10, ten different geometric patterns were used as a testing environment for systematic experimentation, using three different values for the number of facilities n (where both n and k are small). For each instance used for these experiments, 2000 demand points were sampled in the unit plane. For some of the experiments, their locations were drawn from a (truncated) bivariate normal distribution with means of 0.5, variances of 0.09, and zero covariance, whereas for other experiments, their locations were sampled uniformly. The optimal solutions of these instances are known, since it is possible to solve them using an exact solver within a reasonable amount of time.

5.1. Universal ϕ: analysis of α and β

In the first setting that was tested, we applied our method with a parameterized function ϕ, where the parameters α and β are fixed. This means that we applied the same function to each distance matrix entry, independent of the demand point and facility to which the entry belongs. Note that the value of α corresponds with the center as indicated in Fig. 2(d), whereas β can be numerically estimated such that the function ϕ has a specific bandwidth, which we will indicate by b. We tested different combinations of fixed α and b on the test

Table 1
Realized mean $z_{\tilde{d}}(\mathbf{y}^*_{\tilde{d}})$ for different configurations (α, b) on each instance group, compared to the mean $\bar{z}^*_{\tilde{d}}$ for an aggregation obtained by applying planar k-means.

(α, b)	a	b	c	d	e	f	g	h	i	j
(0.3, 0.35)	**3339.01**	**2945.27**	4979.30	3310.10	3361.07	3464.50	4825.58	3342.25	3141.43	3998.15
(0.45, 0.7)	**3329.92**	2971.70	4908.35	3134.70	3154.37	**3360.29**	4656.69	3340.29	3110.07	**3886.14**
(0,0.7)	**3354.88**	**2951.01**	4960.85	3167.11	3270.38	3406.61	4783.13	3345.27	3097.28	3984.47
(0.2, 0.8)	**3338.03**	**2957.35**	4910.60	3115.51	3143.40	**3369.38**	4699.13	**3326.57**	3101.24	3907.59
(0.15, 0.2)	3518.57	3127.03	5041.37	3766.55	3811.70	3851.68	5063.62	3613.80	3598.02	4446.64
(0.6, 0.3)	**3335.99**	3003.57	4970.17	3378.87	3407.12	**3380.05**	4733.32	3621.21	3201.75	3964.05
(0.5, 1.0)	**3331.32**	2967.39	4908.51	3132.64	3154.99	**3359.52**	4649.19	**3331.19**	3115.29	**3881.81**
Benchmark	3361.24	2966.81	4874.81	3096.04	3118.96	3388.63	4643.41	3332.20	3094.54	3895.63

beds where the demand point locations were uniformly distributed, and compared the resulting $z_{\tilde{d}}(\mathbf{y}^*_{\tilde{d}})$ to a benchmark based on planar k-means clustering. We set this benchmark to be $\bar{z}^*_{\tilde{d}}$, where $\tilde{d}$ in this case refers to the distances resulting from obtaining an aggregation mapping through planar k-means clustering. These values are reported in Table 1.

The insights that can be derived from these results are as expected: For some of the instance groups, we do find a configuration that beats the benchmark in terms of the upper bound, but for other groups this does not seem to be possible. We can explain the cause of these differences by looking at how our method generally tends to aggregate points in two different instance groups, (a) and (g), when the parameters are fixed to a single value. This is depicted in Fig. 4: In Fig. 4(a), the boundaries of the (green) aggregation zone are parallel to the allocation boundaries of the facilities (indicated by the dashed lines). In Fig. 4(b), on the other hand, multiple allocation boundaries of the facilities cut through the aggregation zone.

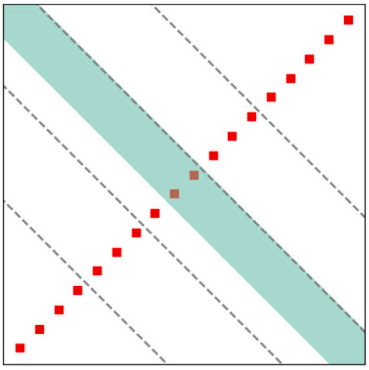
Our example in Fig. 4 demonstrates why applying a single function ϕ universally to the entire distance matrix can, in some cases, be very restrictive in finding good aggregations. The points within an aggregation zone will have similar distances to the potential facilities. If some points are far from all facilities, this will result in them being grouped together, even though it is unlikely that these points will actually be served from the same facility. In the example, we would likely benefit from parameterization on demand point — so that DPs with very different distance vectors can still be aggregated together, provided that their ranking of the distances to the potential facilities is relatively similar. One could also imagine instances where parameterization on facility would be very desirable, for example, if an instance contains facilities with very bad cost-to-capacity ratios or if there is a lot of variation in capacity (and thus likely in service reach) between potential facilities.

5.2. ϕ parameterized on demand point and facility: proposed method for α and γ

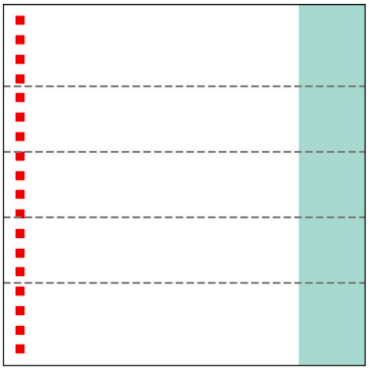
We will now discuss a way to address the shortcomings described in Section 5.1, by parameterizing α_{ij} on demand point and facility, and introducing a parameter γ_i, that amplifies the decayed distances for certain facilities:

$$\phi_{ij}(x) = \gamma_i \frac{e^{(x-\alpha_{ij})\beta}}{1 + e^{(x-\alpha_{ij})\beta}}.$$

We propose to use the decay function in synthesis with an approach that tries to construct the allocation boundaries of the optimal facilities. By this, we mean to identify those points j that are most likely to be 'close' (with respect to some distance, for example, Euclidean distance if the Cartesian coordinates of the demand points are known) to points j' such that $\text{argmax}_i(x_{ij}) \neq \text{argmax}_i(x_{ij'})$. More precisely, in our approach, we intend to estimate α_{ij} such that α_{ij} is roughly equal to the distance of facility i to its allocation boundary in the direction of demand point j. To estimate this for every demand point j, we can take some solution of the problem, for which we expect that the allocation boundary of a facility is in some way representative of the allocation boundary that this facility would have in the optimal solution (if the facility is opened in the optimal solution). To determine where the allocation boundary lies in the selected solution, we look at the distance $d_{ij'}$ of facility i to the point $j' \in \{j' \in [n] | \text{argmax}_i(x_{ij}) \neq \text{argmax}_i(x_{ij'})\}$ that is nearest to j.



(a) An example of a favorable region to be clustered together by an aggregation algorithm



(b) An example of an unfavorable region to be clustered together by an aggregation algorithm

Fig. 4. Examples of a favorable and unfavorable region to be clustered together by an aggregation algorithm.

It is worthwhile to note that, in some cases, taking the nearest point on the other side of the boundary might lead to an undesirable estimation of how far the boundary is. It is even possible to get an estimate α_{ij} that is lower than d_{ij} for a point that is within the allocation boundary. Also, in some cases, there are multiple close points on the other side of the boundary that would yield very different α_{ij} estimates. Thus, it makes sense to guide the process by penalizing points that lie further from the facility (in case of inward estimation) or that are closer to the facility (in case of outward estimation) when selecting the nearest point at the other side of the boundary. Adding the penalty term

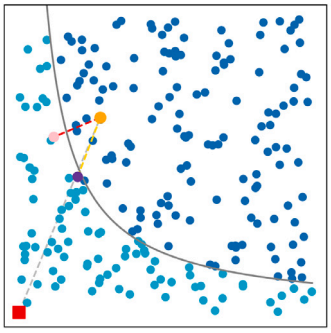
$\pm\omega d_{ij}$ will ensure that the point on the other side of the boundary that is found lies in the same direction as demand point j (with reference to the facility). This concept is shown in Figs. 5(a) and 5(b). Another case where boundary misestimation can happen is at peripheral points: a point j in the corner of the plane that is allocated to facility i may perceive that its closest neighbors that are not allocated to facility i are, in fact, all closer to the facility than j itself, leading to underestimating α_{ij}. This can lead to the formation of 'peripheral ADPs', ADPs that contain points from all the corners of the plane. These ADPs can introduce a significant amount of error. To mitigate this, we can add some surrogate points that are not allocated to any facility, at the bounds of the plane, just for aiding the estimation process (the points are not present at any moment when optimizing the problem). To each of those surrogate points, we can allocate (high) values θ_{ij}. α_{ij} will then be set to the value of θ_{ij}, if such a surrogate point is hit. Example values could be $\theta_{ij} = 1$ for all surrogate points and facilities, or to set θ_{ij} equal to twice the distance between the facility i and the surrogate point j. Fig. 5(c) shows an example of the estimation procedure with surrogate points.

We suspect that the approach will work better when the distances d_{ij} are approximately Euclidean, and it will likely work extremely poorly if distances do not satisfy the triangle inequality or if they are completely random. (Since, in that case, there is barely any exploiting of the distance structure that can be done). Of course, we can only apply this procedure to those facilities for which $y_i = 1$ in the selected solution. If we encounter a solution that has $y_i = 0$, we cannot infer anything about the possible location of the allocation boundary of facility i based on that solution. It is possible to get an estimate of the allocation boundary for every i by first solving m LP relaxations, where the constraint $y_i = 1$ is added to LP relaxation i. This however comes with a significant computational overhead (scaling with m). Therefore, we instead opt for a warm-up period together with an on-the-fly exploration mechanism in our matheuristic, which enables the estimation of the allocation boundary for relevant (promising) facilities. This mechanism will be discussed further in Section 6.

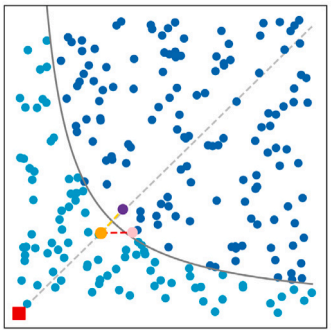
6. Heuristic aggregation algorithm

Fig. 1 shows the main idea of how the core methodology can be used within a matheuristic algorithm to solve large CFLP instances. By applying aggregation iteratively, we can obtain more accurate information of where the allocation boundaries may be in the optimal solution. This information is acquired by inspecting solutions that were already found during a previous iteration. The locations of the allocation boundaries of the open facilities in those solutions provide (more) reasonable estimates for the allocation boundaries in the optimal solution. Furthermore, by keeping track of a ranked list of solutions found by the algorithm, we obtain additional ways to steer the algorithm towards exploration in the beginning, and exploitation in later stages. The parameter γ_i in the aggregation method can for example be based on the number of solutions of good quality in which facility i is opened. Our complete matheuristic can be found in Algorithm 1.

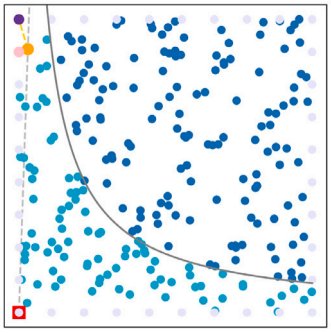
The algorithm consists of 5 phases. Phase 0 is an initialization phase, which is only carried out at the start of the heuristic. Phases 1–3 describe the core aggregation method. Phase 4 is the phase in which the parameters are updated, after which the heuristic will iterate through Phases 1–4 until the termination criterion is met. We will now briefly elaborate on each of the phases. A more detailed explanation of Phase 1–3 can be found in Section 4.



(a) Inward estimation: We penalize points that are further from the facility, to avoid overestimating α_{ij}. In this case, the purple point, which is nearer to the grey dashed line, is preferred over the pink point, even though the pink point is nearer to j.



(b) Outward estimation: We penalize points that are closer to the facility, to avoid underestimating α_{ij}. In this case, the purple point, which is nearer to the grey dashed line, is preferred over the pink point, even though the pink point is nearer to j.



(c) Corner estimation: We add surrogate reference points to avoid underestimating α_{ij}. In this case, the selected purple point is not in the original set of points on the other side of the boundary, which does not reflect the actual distance of the allocation boundary in the direction of j well.

Fig. 5. Illustration of the selection process of a near point to determine α_{ij}, which should approximate the distance of facility i to its allocation boundary in the direction of point j.

Algorithm 1 Iterative aggregation matheuristic

Output: A best found solution $(\boldsymbol{x}, \boldsymbol{y})$ and a corresponding aggregation mapping M and problem $\tilde{P}$ for which $\boldsymbol{y}$ is the optimal locational decision.

1: $t \leftarrow 0$
2: TERMINATION_CRITERION_MET $\leftarrow$ False
3: **Execute** Phase 0:
4: $\quad\rightarrow$ **Pass** $\alpha^{(INIT)}$,$b^{(INIT)}$, $\gamma^{(INIT)}$, ω,
5: $\quad\quad$ $|S_{sample}|$, $|S_{surr}|$,
6: $\quad\rightarrow$ **Obtain** $\alpha_{ij}^{(1)}$, $\gamma_i^{(1)}$, $\beta_i^{(1)}$
7: List of solutions $L \leftarrow \emptyset$

$t \leftarrow 1$
while TERMINATION_CRITERION_MET = False **do**
  Execute Phase 1:
    → **Pass** P, $\alpha_{ij}^{(t)}$, $\gamma_i^{(t)}$, $\beta_i^{(t)}$, K
    → **Obtain** Mapping $M^{(t)} : [n] \rightarrow K$
  Execute Phase 2:
    → **Pass** P, $M^{(t)}$, $\{\boldsymbol{y}^{(t')} : t' \in [t-1]\}$
    → **Obtain** $(\tilde{\boldsymbol{x}}^{*(t)}, \boldsymbol{y}^{(t)})$
  Execute Phase 3
    → **Pass** P, $\boldsymbol{y}^{(t)}$
    → **Obtain** $\boldsymbol{x}^{(t)}$, $z^{(t)}$
  $L \leftarrow L \cup (\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}, z^{(t)})$
  $t_{best} \leftarrow \text{argmin}_{t' \in [t]} z^{(t')}$
  if $t_{best} \leq t - T_{improve} \| t = T_{max}$ **then**
    TERMINATION_CRITERION_MET $\leftarrow$ True
  else
    Execute Phase 4
      → **Pass** t, L, $(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}, z^{(t)})$, $\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{b}^{(t)}$, $\boldsymbol{\gamma}^{(t)}$
  end if
  $t \leftarrow t + 1$
end while
return $(\boldsymbol{x}^{(t_{best})}, \boldsymbol{y}^{(t_{best})}, z^{(t_{best})})$, $M^{(t_{best})}$

6.1. Phase 0: Initialization

Before entering the main loop of the heuristic, $\boldsymbol{\alpha}^{(1)}$, $\boldsymbol{\beta}^{(1)}$ and $\boldsymbol{\gamma}^{(1)}$ need to be initialized. Furthermore, some preliminaries are needed to be able to determine the allocation boundaries that are needed for $\boldsymbol{\alpha}$ estimation in Phase 4. For this procedure, we require the distances from each demand point to other demand points.

For big problems, using all demand points as reference points for the boundary estimation will weigh heavily on memory and computational time, so that we are required to take a sample of the demand points to use as a reference. The sample needs to be big enough to accurately obtain estimates of the allocation boundary, yet small enough to ensure computational tractability. In our experiments, when $n > 2500$ we take a sample of size 2500. If the Cartesian coordinates of the demand points are known, the Euclidean distance between the demand points can be used (possibly as a surrogate). If this is not the case, but the triangle inequality holds, we can use a lower bound based on the distances between the demand points and the facilities, as described in Line 30 of Phase 0.

As described in Section 5.2, surrogate points are added to the reference points at the boundary of the plane. This is again relatively straightforward if the Cartesian coordinates of the demand points are known, since they can be linearly placed along a bounding box. If the coordinates are not known, such a bounding box can be artificially estimated by constructing its main diagonals based on the estimated distances between the demand points. The surrogate points can then be added by means of an affine combination of the distances of the corner points of the artificial box. The distances to these points are increased by a small padding factor of 1.15 to account for the fact that the bounding box and distances between demand points are estimated artificially.

Phase 0 Initialization

Input: $\alpha^{(INIT)}$ $b^{(INIT)}$, $\gamma^{(INIT)}$, fixed number of reference demand points to be sampled ($|S_{sample}|$), fixed number of surrogate demand points to be added ($|S_{surr}|$, should be a multiple of 4).

Output: $\alpha_{ij}^{(1)}$ for all facilities $i \in [m]$ and demand points $j \in [n]$, $\gamma_i^{(1)}$ and $\beta_i^{(1)}$ for all facilities $i \in [m]$, set of reference demand points S_{sample}, set of surrogate demand points S_{surr}

1:
2: **function** ADD_NON-CARTESIAN_SURROGATES( )
3: CORNER1, CORNER2 $\leftarrow \text{argmax}_{(j,j') \in [n] \times S_{sample}} d_{jj'}$
▷ We estimate corner points for a bounding box: The first diagonal axis if this box spans the largest distance between two individual demand points (One demand point and one point in the sample);
4: **for all** $j' \in S_{sample}$ **do**
5: $d_{\text{DIAGONAL1}j'} \leftarrow \sqrt{d_{\text{CORNER1}j'}} + \sqrt{d_{\text{CORNER2}j'}}$
6: **end for**
7: CORNER3 $\leftarrow \text{argmax}_{j' \in S_{sample}} d_{\text{DIAGONAL1}j'}$
▷ From all points in S_{sample}, the point which is furthest from the two existing corner points should be the third corner point. We use the square root to disincentivize choosing a point that is very far from one corner point but not from the other.
8: CORNER4 $\leftarrow \text{argmax}_{j' \in S_{sample}} d_{\text{DIAGONAL1}j'} + d_{\text{CORNER3}j'}$
▷ The same principle applies for the fourth corner point, which should be the point that is furthest from the third corner point and the diagonal formed by corner points 1 and 2.
9: $\ell \leftarrow$ A vector of length $\frac{1}{4}|S_{surr}|$ whose elements are spaced linearly between 0 and 1
10: **for all** $i \in [n]$ **do**
11: **for all** $j \in \{1, ..., \frac{1}{4}|S_{surr}|\}$ **do**
12: $d_{ij} \leftarrow 1.15(\ell_j d_{i\text{CORNER1}} + (1-\ell_j) d_{i\text{CORNER3}})$
13: $j' \leftarrow j + \frac{1}{4}|S_{surr}|$
14: $d_{ij'} \leftarrow 1.15(\ell_j d_{i\text{CORNER2}} + (1-\ell_j) d_{i\text{CORNER3}})$
15: $j'' \leftarrow j + \frac{1}{2}|S_{surr}|$
16: $d_{ij''} \leftarrow 1.15(\ell_j d_{i\text{CORNER2}} + (1-\ell_j) d_{i\text{CORNER4}})$
17: $j''' \leftarrow j + \frac{3}{4}|S_{surr}|$
18: $d_{ij'''} \leftarrow 1.15(\ell_j d_{i\text{CORNER1}} + (1-\ell_j) d_{i\text{CORNER4}})$
19: **end for**
20: **end for**
▷ This sets the distances from the demand points to the surrogate points as affine combinations of corner points of the artificial bounding box.
21: **end function**
22: $S_{sample} \leftarrow$ sample of $|S_{sample}|$ demand points from $[n]$ ▷ The sample of reference demand points can be the entire set $[n]$, if n is small enough.
23: **if** CARTESIAN_COORDINATES_KNOWN = True **then**
24: $S_{surr} \leftarrow$ a set of $|S_{surr}|$ surrogate demand points, composed of 4 sets of $\frac{1}{4}|S_{surr}|$ points linearly spaced along each bound of the plane.
25: **for all** (j, j') in $[n] \times (S_{sample} \cup S_{surr})$ **do**
26: $d_{jj'} \leftarrow$ Euclidean distance between j and j'
27: **end for**
28: **else**
29: **for all** (j, j') in $[n] \times S_{sample}$ **do**
30: $d_{jj'} \leftarrow max_{i \in [m]} |d_{ij'} - d_{ij}|$
▷ If the triangle inequality holds, this is a lower bound for the actual distance between j and j'.
31: **end for**
32: ADD_NON-CARTESIAN_SURROGATES( )
33: **end if**
34: **for all** $(i, j) \in [m] \times S_{surr}$ **do**
35: $\theta_{ij} \leftarrow 2d_{ij}$
36: **end for**
37: **function** BETA_FROM_BANDWIDTH(b)
38: Solve for β: $b = 2\frac{log(\beta + \sqrt{\beta(\beta-2)})}{\beta}$
▷ To compute which β corresponds to a certain value of b, we compute the points where the slope of the base transformation is equal to $\frac{1}{2}$. β is chosen such that all points with a slope bigger than $\frac{1}{2}$ are in the band with width b. Solving the above equation is done numerically and approximately.
39: **end function**
40: **for all** $(i, j) \in [m] \times [n]$ **do**
41: $\alpha_{ij}^{(1)} \leftarrow \alpha^{(INIT)}$
42: **end for**
43: **for all** $i \in [m]$ **do**

44: $b_i^{(1)} \leftarrow b^{(INIT)}$
45: $\beta_i^{(1)} \leftarrow$ BETA_FROM_BANDWIDTH$(b_i^{(1)})$
46: $\gamma_i^{(1)} \leftarrow \gamma^{(INIT)}$
47: **end for**
48: $t \leftarrow 1$
49: $L \leftarrow \emptyset$
50: $p_{explore} \leftarrow 1$

6.2. Phases 1 (decay function and clustering), 2 (obtaining and solving reduced problem), and 3 (allocation problem and solution to P)

Phases 1, 2 and 3 encompass the application of the different steps of the core method, as described in Section 4. In Phase 2, some additional constraints are added in most cases:

- In line 11, constraints are added that eliminate each solution that has already been seen by the algorithm.
- In line 16, a constraint *may be* added to ensure that the algorithm will arrive at a solution that opens at least one, and at most ξ_{max} facilities for which the allocation boundary has not yet been estimated. This mechanism also ensures the exploration of solutions that are not composed of the same facilities as the solutions that have already been found. During an initial warm-up period, the algorithm will explore with probability 1, and during the rest of the iterations it will explore at a lower rate.

Phase 1 Decay function and clustering

Input: Problem P, $\alpha_{ij}^{(t)}$ for all facilities $i \in [m]$ and demand points $j \in [n]$, $\gamma_i^{(t)}$ and $\beta_i^{(t)}$ for all facilities $i \in [m]$, set of ADPs K
Output: Mapping $M^{(t)} : [n] \rightarrow K$
1: **for all** $(i, j) \in [m] \times [n]$ **do**
2: $\phi_{ij}^{(t)} \leftarrow \gamma_i^{(t)} \frac{e^{\beta_i^{(t)}(x-\alpha_{ij}^{(t)})}}{1+e^{\beta_i^{(t)}(x-\alpha_{ij}^{(t)})}}$
3: **end for**
4: **for all** $j \in [n]$ **do**
5: $\boldsymbol{v}_j^{(t)} \leftarrow (\phi_{1j}^{(t)}(d_{1j}), ..., \phi_{mj}^{(t)}(d_{mj}))$
6: **end for**
7: Apply k-means ++ clustering to partition the n vectors $\boldsymbol{v}_j^{(t)}$ into $|K|$ clusters, and obtain the corresponding mapping $M^{(t)} : [n] \rightarrow K$

Phase 2 Obtaining and solving reduced problem

Input: Problem P, Mapping $M^{(t)}$, all previously found solutions for $\boldsymbol{y}$: $\{\boldsymbol{y}^{(t')} : t' \in [t-1]\}$
Output: $(\tilde{\boldsymbol{x}}^{*(t)}, \tilde{\boldsymbol{y}}^{*(t)})$, the optimal solution of the reduced problem $\tilde{P}$
1: **for all** $k \in K$ **do**
2: $D_k^{(t)} \leftarrow \sum_{\{j:M^{(t)}(j)=k\}} D_j$
3: **for all** $i \in [m]$ **do**
4: $d_{ik} = \frac{\sum_{\{j:M^{(t)}(j)=k\}} D_j d_{ij}}{D_k}$.
5: **end for**
6: **end for**
7: Obtain $\tilde{P}^{(t)}$, a CFLP with demand points K and facilities $[m]$ and demand volumes and distances as computed above.
8: **for all** $t' \in [t-1]$ **do**
9: Let $\psi_i^{(t')} = \begin{cases} 1 & \text{if } y_i^{(t')} = 1 \\ -1 & \text{Otherwise} \end{cases}$
10: Add the following constraint to $\tilde{P}^{(t)}$:
11: $\sum_{i\in[m]} \psi_i^{(t')} y_i \leq \sum_{i\in[m]} y_i^{(t')} - 1$
▷ Note that $\psi_i^{(t')}$ and $y_i^{(t')}$ are parameters, not variables
12: **end for**
13: $S_{unexplored} = \{i : y_i^{(t')} = 0 \; \forall \; t' \in [t-1]\}$
14: **if** $S_{unexplored} \neq \emptyset$ **then**
15: With probability $p_{explore}$, add the following constraint to $\tilde{P}^{(t)}$:
16: $\sum_{i\in S_{unexplored}} y_i \geq \xi$
17: Where $\xi \sim U[1, \xi_{max}]$
18: **end if**
19: Solve $\tilde{P}^{(t)}$ using an exact solver.
20: $(\tilde{\boldsymbol{x}}^{*(t)}, \boldsymbol{y}^{(t)}) \leftarrow$ The optimal solution of $\tilde{P}^{(t)}$

Phase 3 Allocation problem and solution to P

Input: Problem P and solution $\boldsymbol{y}^{(t)}$ of reduced problem
Output: $\boldsymbol{x}^{(t)}$, $z^{(t)}$
1: Solve P exactly with the values for $\boldsymbol{y}$ fixed to $\tilde{\boldsymbol{y}}^{*(t)}$
2: $\boldsymbol{x}^{(t)} \leftarrow \boldsymbol{x}_{\boldsymbol{d}}^{*}(\boldsymbol{y}_{\boldsymbol{d}}^{*})$ and $z^{(t)} \leftarrow z_{\boldsymbol{d}}^{*}$.

6.3. Phase 4: Update parameters

After completing Phase 3, we update the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ and re-iterate Phases 1–3 with the updated parameters. The updating rules are different for the warm-up period and the rest of the algorithm: The warm-up period is used mainly to explore the solution space and to estimate the allocation boundaries of the facilities, whereas during the remainder of the algorithm we aim to capitalize on the information that was gained before.

During the warmup period, we explore different solutions by keeping the entries of $\boldsymbol{\alpha}$ equal to 0, and applying different bandwidth values, starting with a large value and then slowly decreasing. In fact, the shape of the transformation function ϕ in this setting is not sigmoidal, but it is instead strictly increasing and concave. This transformation suits a general situation, enabling exploration without any information being available on the allocation boundaries. The parameter γ_i is kept constant for all facilities during the warmup period, as we want to explore the solution space as much as possible while having no information yet about which facilities are 'promising'. During the warmup period, we do gain more information on the allocation boundaries of the facilities. We do already update these, and store the corresponding α-values in $\tilde{\boldsymbol{\alpha}}$, which will be used by the algorithm after the warmup period has ended.

The complete matheuristic is shaped by how the update rules used after the warm-up period are specified. Different update mechanisms may be better suited to different situations. As we want the algorithm to find decent solutions in relatively few iterations, we opt for relatively large update steps for $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ after the warmup period has finished. Especially the value for the bandwidth parameter will have a different impact on the solution progression depending on the characteristics of the instance. It is clear however that starting with a high bandwidth value and decreasing it as the allocation boundary becomes more precise and the solutions obtain better quality, is a good strategy. We thus opt for a set of update rules where $\boldsymbol{\alpha}$ can change relatively quickly, while slowly decreasing the bandwidths, as well as decreasing γ_i for the facilities that do not show up in any of the best solutions found so far. The update rules we used can be found below: $\boldsymbol{\alpha}$ is a weighted average of all the allocation boundaries estimated before, favoring more recent observations. If for some facility i, all $\alpha_{ij}^{(t)}$ are 0 for all j, we do not smooth the values corresponding to this facility, and the newly obtained values derived from the allocation boundary are used at time $t+1$. $\boldsymbol{b}$ will slowly decline over time for all facilities. $\boldsymbol{\gamma}$ is updated based on a list of κ best solutions: For facilities that are not opened in any of these solutions, it will decay towards 0, and for facilities that are opened in multiple solutions in this list, it will grow to 1. The more solutions in the list contain a facility, the faster its respective γ will go to 1.

After finishing Phase 4, Phases 1–3 will be repeated, unless one of the termination criteria is met. The algorithm terminates if the maximum number of iterations T_{max} is reached, or if the best solution has not changed over the past $T_{improve}$ iterations.

Phase 4 Update parameters

Input: Current solution $(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}, z^{(t)})$, Set of previous solutions L, current parameters $\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{\beta}^{(t)}$, $\boldsymbol{\gamma}^{(t)}$

function SET_ALPHA_VALUES($\boldsymbol{x}$, i, t)
  $A_i \leftarrow \{j | j \in [n] : \text{argmax}_{i' \in [m]}\, x_{i'j} = i\}$
  for all $j \in A_i$ **do**
    $j' \leftarrow \text{argmin}_{j'' \in S \setminus A_i}\, d_{jj''} + \omega d_{ij''}$
    if $j' \in S_{surr}$ **then**
      $\tilde{\alpha}^{(t)} ij \leftarrow \theta ij'$
    else
      $\tilde{\alpha}^{(t)} ij \leftarrow d ij'$
    end if
  end for
  for all $j \in [n] \setminus A_i$ **do**
    $j' \leftarrow \text{argmin}_{j'' \in A_i \cap S}\, d_{jj''} - \omega d_{ij''}$
    $\tilde{\alpha}^{(t)} ij \leftarrow d ij'$
  end for
end function

function UPDATE_ALPHA(L, t, $\boldsymbol{\alpha}^{(t)}$)
  for all $i \in [m] : y_i^{(t)} = 1$ **do**
    SET_ALPHA_VALUES($\boldsymbol{x}^{(t)}$, i, $t+1$)
  end for
  for all $i \in [m] : y_i^{(t)} = 0$ **do**
    for all $j \in [n]$ **do**
      $\tilde{\alpha} ij^{(t+1)} \leftarrow \tilde{\alpha} ij^{(t)}$
    end for
  end for
  for all $i \in [m]$ **do**
    if $\sum_{j \in [n]} \tilde{\alpha}_{ij}^{(t)} \neq 0$ **then**
      for all $j \in [n]$ **do**
        $\tilde{\alpha} ij^{(t+1)} \leftarrow (1-\eta_1)\tilde{\alpha} ij^{(t)} + \eta_1 \tilde{\alpha}_{ij}^{(t+1)}$
      end for
    end if
    ▷ We only apply smoothing when we already have established an initial α estimate for facility i
  end for
  $\tilde{\boldsymbol{\alpha}}^{(t+1)} \leftarrow (1-\eta_1)\tilde{\boldsymbol{\alpha}}^{(t)} + \eta_1 \tilde{\boldsymbol{\alpha}}^{(t+1)}$
  if $t < T_{warmup}$ **then**
    $\boldsymbol{\alpha}^{(t+1)} \leftarrow \boldsymbol{\alpha}^{(t)}$
  else
    $\boldsymbol{\alpha}^{(t+1)} \leftarrow \tilde{\boldsymbol{\alpha}}^{(t+1)}$
  end if
end function

function UPDATE_BETA(L, $\boldsymbol{b}^{(t)}$)
  $\boldsymbol{b}^{(t+1)} \leftarrow (1-\eta_2)\boldsymbol{b}^{(t)}$
  for all $i \in [m]$ **do**
    $\beta_i^{(t+1)} \leftarrow$ BETA_FROM_BANDWIDTH($b_i^{(t+1)}$)
  end for
end function

function UPDATE_GAMMA(L, t, $\boldsymbol{\gamma}^{(t)}$)
  if $t < T_{warmup}$ **then**
    $\boldsymbol{\gamma}^{(t+1)} \leftarrow \boldsymbol{\gamma}^{(t)}$
  else
    $L_{top} \leftarrow \emptyset$
    $\kappa \leftarrow 1$
    while $\kappa \leq \min(t, \kappa_{max})$ **do**
      $L_{top} \leftarrow L_{top} \cup \{\text{argmin}_{(\boldsymbol{x},\boldsymbol{y},z) \in L \setminus L_{top}}\, z\}$
      $\kappa \leftarrow \kappa + 1$
    end while
    $\boldsymbol{\gamma}^{(t+1)} \leftarrow (1-\eta_3)\boldsymbol{\gamma}^{(t)}$
    for all $i | \exists (\boldsymbol{x}, \boldsymbol{y}, z) \in L_{top} : y_i = 1$ **do**
      $\gamma^{(t+1)} i \leftarrow \gamma_i^{(t+1)} - (\sum (\boldsymbol{x}, \boldsymbol{y}, z) \in L_{top} y_i - 1)\eta_3 \gamma_i^{(t)}$
      $\gamma^{(t+1)} i \leftarrow \gamma_i^{(t+1)} + \sum (\boldsymbol{x}, \boldsymbol{y}, z) \in L_{top} y_i \eta_3$
    end for
  end if
end function
UPDATE_ALPHA(L , t, $\boldsymbol{\alpha}^{(t)}$)
UPDATE_BETA(L , $\boldsymbol{b}^{(t)}$)
UPDATE_GAMMA(L ,t, $\boldsymbol{\gamma}^{(t)}$)
if $t \geq T_{warmup}$ **then**
  $p_{explore} \leftarrow 0.2$
end if

7. Computational experiments

In order to assess the effectivity and versatility of our method, we test its performance on a broad range of instances. We compare its performance to the benchmark method *(planar k-means)*. In this section, we will present a detailed description and motivation for the set up of our experiments, as well as the experimental results on each of the instance sets.

7.1. Instance sets

Firstly, we have applied our method to a set of relatively small CFLP instances from the literature, whose optimal solutions are known. The resulting optimality gaps can provide insight into the general performance that can be expected of our heuristic when solving instances of the CFLP. The instances we have used are the following:

- Very small instances labeled `cap41` - `cap134` from the OR library (Beasley, 1988).
- Instances `capa` - `capc` from the same library, which are slightly larger than the previous set. In addition to this, we created instances using the same generation process in order to be able to compare the performance of our method on instances where the Cartesian coordinates are not known to similar instances where they are known. We relabeled the instances `capa` - `capc` to `cap201` - `cap224`, and labeled the generated instances `cap231` - `cap284`.
- 5 sets of instances as proposed by Klose and Görtz (2007) and Görtz and Klose (2012), where $m \times n = 1000 \times 1000$, 1500×300, 1500×600, 500×100 and 500×200.

Besides these instances from the literature, we also generated some new instances on which we compared our method to the benchmark approach. In addition to the instances in Fig. C.10, that were mentioned in Section 5, we generated different instances from a data set inspired by a location problem for waste treatment centers in reverse logistics. This was done in order to evaluate our method on real-life location problems with many demand points. The demand points represent small and large future construction and demolition sites which will produce insulation waste over a time horizon spanning multiple decades. In this section, we elaborate on these instances.

The instances and underlying data are partly based on a real-life location application for locating treatment plants for insulation waste from construction and demolition. Estimates of the volumes of insulation material in buildings are made for a region of around 2 million people around Amsterdam, the Netherlands. The region contains both very dense urban areas and more rural areas. The treatment plants, for which we have to make the locational decision, have to be built from scratch. The plants need to have enough capacity to process all the insulation material currently present in buildings.

Table 2
Demand point configurations in instances.

	Specifics	Total volume (m^3)	Number of DPs	Volume of largest DP	Number of instances tested
0	Euclidean distances, realistic amount collected from pre-sorting locations	4,227,074	57,854	46,786	200
1	Euclidean distances, more waste collected from pre-sorting locations	4,227,286	36,635	62,807	160
3	Euclidean distances, more waste present in non-residential buildings, (much) more waste collected from pre-sorting locations	8,243,537	22,540	261,293	70
5	Euclidean distances, realistic amount collected from pre-sorting locations	4,227,074	57,854	46,786	200
6	Road distances, more waste collected from pre-sorting locations	4,227,286	36,635	62,807	160
8	Road distances, more waste present in non-residential buildings, (much) more waste collected from pre-sorting locations	8,243,537	22,540	261,293	70

Some points with very high demand are present in the data: These represent existing collection and pre-sorting facilities which will first collect the waste of the smallest among the construction and demolition sites: The waste then needs to be transported from this collection station to the treatment facilities. The amounts of waste accumulating in these locations will thus be much larger than the amounts at a single large construction site.

We generated 3 different sets of demand points and volumes, using different parameters for the amount of material in buildings. We also changed the threshold value below which waste from a potential construction site is not collected from that site, but rather from its nearest pre-sorting facility. The characteristics of the 3 sets of demand points are shown in Table 2. We have tested our algorithm both for Euclidean and over the road distances, resulting in 6 different settings for the DPs. The instances where we make use of over the road distances were generated with help of openrouteservice.org.

As for the potential locations and parameters of the treatment plants, we generated sets of 10 parameters for cost and capacity. Some of these parameters are more realistic than others. We however also test our method on parameter sets that are less realistic to the original data context in order to infer how well it will perform in problems which do have different characteristics than ours.

For each of the parameter sets, we generate the costs from one of ten pre-defined cost distributions, and fix all capacities to one of ten different values. The number of potential locations in our instances ranges from 42 to 125. The instances which have 42 potential locations, consist of those locations where building a treatment plant would actually be possible. However, in order to test the behavior of our method on instances where m grows larger, we also decided to generate instances with more locational optionality. For this, we incrementally add industrial zones to the set of potential locations to generate instances ten with different values of n. We also tried settings with more than 125 potential locations, but from this problem size onward, we are facing computational limits at which CFLP problems can no longer be addressed by methods solely addressing demand point reduction. The complexity arising from the number of locations needs to be reduced as well in order to (heuristically) obtain good solutions for these problems. All parameters used to generate our instances can be found in Table 3.[1] The costs associated with the facilities are not directly expressed in a monetary unit, but one cost unit is equal to the cost of moving 1 m^3 of material over a distance of one meter. Fig. C.9 shows two example instances.

[1] For a small subset of the possible combinations, it is immediately clear that no feasible solution exists: The instances resulting from these combinations are discarded.

Table 3
Facility configurations in instances.

(a) Number of potential facilities m		(b) Cost distribution		(c) Capacity	
	m		Distribution		Capacity
0	42	0	$U(2150, 2350)$	0	142,500
1	60	1	$U(2000, 2500)$	1	170,000
2	80	2	$U(1000, 4000)$	2	200,000
3	100	3	Fixed: 2250	3	220,000
4	125	4	$U(3150, 3350)$	4	250,000
		5	$U(3000, 3500)$	5	300,000
		6	Fixed: 3250	6	375,000
		7	$U(1150, 1350)$	7	450,000
		8	$U(1000, 1500)$	8	600,000
		9	Fixed: 1250	9	900,000

7.2. Experimental setup

We have applied our method, as well as the benchmark method to each of the instance sets described before. We use slightly different configurations for each set, based on the complexity of the problems and the computational load needed to store the required data during the solution process. We ran our experiments using different input values for the number of ADPs, again differing per instance set. For each instance, we ran the benchmark method 40 times and for the instances from the literature we also ran our matheuristic multiple times. This is because the results of both methods depend on the random initialization. For the 860 newly generated instances, we only ran our matheuristic once. Because of its iterative nature, running the heuristic once many instances still gives an adequate impression of its performance. In order to enhance the solution process of the reduced problem in Phase 2, we add the following known valid inequalities to the reduced problem when solving more complex instances:

$$x_{ij} \leq y_i \; \forall \; i \in [m], j \in [n] \tag{25}$$

The other hyperparameters we used for these experiments can be found in Table 4. We used 16 cores for instances that were computationally less heavy, and for some heavier instances we used 32 cores, based on memory requirements. For some runs, we limited the number of threads the solver could use to limit memory use. A detailed overview of the characteristics of the test instances and experimental setup for these instances can be found in Appendix B. For all instances, we limited the MILP solve time for the reduced problem to 20 min per iteration (for both our method and the benchmark method), using the best found solution instead of the optimum if this limit was reached. We limited the total runtime of our algorithm to 9000 s. Our experiments were conducted on a computing cluster, on 2.4 GHz CPU nodes, with 2 GB of available RAM per core. The matheuristic and benchmark method were implemented in Python `3.10.4` and the reduced and allocation problems were solved using Gurobi `10.0.1`.

Table 4
Hyperparameter configuration.

	Phase 0
$\lvert S_{sample}\rvert$	2500
$\lvert S_{surr}\rvert$	500
θ_{ij}	2 times the Euclidean distance between i and j for the Euclidean real-life instances, 2.8 times the Euclidean distance between i and j for all other instances.
$\alpha^{(INIT)}$	0
$b^{(INIT)}$	1
$\gamma^{(INIT)}$	1
ω	$\frac{4}{5}$ (Euclidean real-life); $\frac{4}{7}$ (Others)
	Phase 2
ξ_{max}	5
	Phase 4
η_1	0.35
η_2	0.1
η_3	0.18
κ_{max}	6
$T_{improve}$	25
T_{max}	40
T_{warmup}	6 (OR-Library) or 10 (Other instances)

7.3. Results

In the following, we will present and evaluate the performance of our heuristic on the different test instances. We will shed some light on optimality gaps for the instances for which the optimal values are known. We will also compare the performance of our method to the benchmark method, and assess the impact of the number of ADPs $|K|$ on the performance and running times of the method.

7.3.1. Beasley OR library instances

The CFLP instances from the OR library (Beasley, 1988) are very small, and thus not representative for the types of instances for which our method could be of use: Using state of the art solvers it is possible to solve these problems in little time. The instances do however constitute a good test set for evaluating our method and obtaining insight into its working. As the instances `cap41--cap134` have non-Euclidean transportation costs, but Cartesian coordinates are available (The 50 demand points represent US cities), a comparison to the benchmark method is possible and insightful. The slightly larger instances `cap201--cap284` can give insights into how effective our method is in the absence of Cartesian coordinates.

Table 5 shows the performance of our method on the OR library instances. This table states the mean optimality gaps for our method, as well as the best optimality gaps attained by the benchmark method, for different values of $|K|$, averaged over each instance group. A striking observation is that our method is often able to outperform the best benchmark run if $|K|$ is sufficiently large: Most of the gaps near 0 as $|K|$ grows larger. However, our method has problems dealing with smaller values of $|K|$, even more so than the benchmark method. We presume that this is due to the fact that the ADPs created by our method in this case will be more imbalanced, and a very large ADP that has to be served by multiple facilities will cause multiple facilities to be open in the center of this ADP, which is not optimal.

7.3.2. Klose & Görtz instances

The instances from Klose and Görtz (2007) and Görtz and Klose (2012) are larger than the OR library instances, and the distances in these instances are Euclidean. For evaluation of our methods, we chose four sets of instances which have $n > m$, and the biggest set of instances for which $m = n$, to evaluate how this would impact the performance.

The results of our method on these instances, and the best results of the benchmark method are shown in Table 6. Comparing the performance across the different instance sets, we can see that our method on average outperforms the best benchmark runs especially on those sets where $n > m$. The benchmark method is able to outperform our method on the instances where $m = n$, which is not a big surprise given that as m grows larger, our method will have more difficulty in creating good ADPs based on promising solutions, as there will be more facilities for which the method will not have seen solutions where these were opened. Another observation is that the optimality gaps for both methods will practically diminish as $|K|$ grows to 150. This is also expected as n is relatively small in these instances, which means setting $|K|$ to 150 results in producing small ADPs.

For these larger instances it also is insightful to compare running times. An overview of these is given in Table 7. This table reports the mean times per iteration for different values of $|K|$, as well as the mean time spent during initialization. We do remark here that the main driver for time spent during initialization for these instances is the time it takes to initialize the allocation problem. If multiple benchmark runs were done sequentially, this time would only be spent once. We separately measured how long it took Gurobi on average to set up this problem for a subset of our benchmark runs. The mean benchmark runtimes, which are also reported in Table 7, have been corrected by subtracting these averages. The problem setup times are also reported in the table, and are very comparable to the initialization times of our method, indicating that on these problems, the remainder of the initialization does not take a significant amount of time. In addition, Table 10 gives an insight into how many iterations were needed for the algorithm to reach its best value, and how many iterations were carried out before termination. Across the board, one iteration of our algorithm takes roughly an equal amount of time as one benchmark run, without the time it takes to initialize the allocation problem. Judging from Table 10, it is clear that our method is able to reach its best value in a relatively small number of iterations, which means that our approach is timewise competitive with running the benchmark method multiple times and keeping the best solution. For some of the larger instance sets, our algorithm needed significantly less time per iteration than the benchmark method. We suspect that a cause for this might be that the reduced problems that occur during the warmup period are easier to solve to optimality than the problems resulting from the benchmark method.

7.3.3. Real scale amsterdam region instances

Table 8 presents an overview of the average performance of our heuristic on the different groups of instances, compared to the best result in 40 benchmark runs. The runtimes involved in both approaches are reported in Table 9.

The results show that our method is able to outperform the best of 40 benchmark runs on average on most of the instance groups, and the outperformance is stronger on the non-Euclidean instances than on the Euclidean instances. An interesting observation is that on the non-Euclidean instances we observe that our method sometimes very clearly outperforms the best benchmark run, but on other instances the best benchmark run found a significantly better solution. The fact that this only holds for the best benchmark run likely indicates a larger variation in the objective values of the solutions given by the benchmark method: The benchmark method does not necessarily find the best solution every time nor is the objective of its reduced problems close to the true objective, but there is enough variation in the clustering outcomes for the best run to be quite competitive.

Based on earlier experimentation and insights from the results on the instances from the OR-library, we would expect our method to perform worse on instances where the number of open facilities in the optimal solution is (expected to be) higher than the number of ADPs. However, the results do not clearly show this effect. The converse is also true: Instances with a label ending in 9, which often require fewer facilities to be open, do not show a clear pattern of outperformance. A detailed visualization of the performance of each method on each instance is given in Appendix D.

Table 5
Average optimality gaps (%) of our heuristic (averages over 30 runs per instance) and the benchmark method (best of 40 runs) on different sets of instances from the OR-library (Beasley (1988)) and some instances generated using the same generation process. Note: The benchmark method requires the availability of Cartesian coordinates, which were not available for instances cap201 - cap224.

$\|K\|$	6		9		15		21	
	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
cap41-44	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
cap51	0.303	**0.211**	0.020	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
cap61-64	**0.748**	0.943	**0.104**	0.172	**0.005**	0.105	**0.000**	**0.000**
cap71-74	**0.692**	1.136	**0.093**	0.480	**0.000**	0.105	**0.000**	**0.000**
cap81-84	3.629	**2.237**	2.050	**1.152**	0.049	**0.029**	**0.000**	**0.000**
cap91-94	2.359	**1.869**	1.205	**0.677**	**0.063**	0.148	**0.000**	0.053
cap101-104	**2.190**	2.533	**1.045**	1.166	**0.065**	0.270	**0.000**	0.059
cap111-114	3.739	**2.940**	1.719	**0.600**	**0.052**	0.221	**0.000**	0.021
cap121-124	3.179	**2.117**	1.366	**0.772**	**0.058**	0.191	**0.000**	0.079
cap131-134	2.702	**2.561**	1.289	**1.005**	**0.075**	0.334	**0.000**	0.079
cap201-204	0.040	–	0.048	–	0.007	–	0.004	–
cap211-214	3.938	–	0.957	–	0.295	–	0.139	–
cap221-224	6.355	–	0.603	–	0.183	–	0.094	–
cap231-234	**0.281**	0.394	0.023	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
cap241-244	**0.248**	0.301	**0.002**	0.138	0.024	**0.000**	**0.000**	**0.000**
cap251-254	3.576	**2.457**	**1.104**	1.138	0.412	**0.193**	0.098	**0.048**
cap261-264	**21.255**	22.577	13.449	**13.181**	3.042	**2.889**	**0.602**	0.869
Average	3.372[a]	**3.179**[a]	–[b]	–[b]	**0.290**[a]	0.339[a]	–[b]	–[b]
$\|K\|$	9		21		35		70	
	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
cap271-274	**33.180**	34.426	**6.778**	7.052	**1.002**	1.540	**0.058**	0.379
cap281-284	**7.992**	8.388	**0.359**	0.619	**0.207**	0.282	**0.046**	0.137
Average	4.238[b]	**4.151**[b]	**0.514**[b]	0.582[b]	**0.604**[c]	0.911[c]	**0.052**[c]	0.258[c]

[a] Average over instances cap41 - cap134 and cap231 - cap264, for a fair comparison.
[b] Average over instances cap41 - cap134 and cap231 - cap284, for an overall and fair comparison.
[c] Average over instances cap271 - cap284.

Table 6
Optimality gaps (%) of our heuristic on different sets of instances from Klose and Görtz (2007) & Görtz and Klose (2012) for different values of $|K|$, averaged over 5 instances per set and 6 runs per instance, and the benchmark method (best of 40 runs, averaged per set).

$\|K\|$	20		75		150	
	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
1000 × 1000_5	0.983	**0.821**	0.214	**0.141**	0.038	**0.035**
1000 × 1000_10	0.967	**0.746**	0.152	**0.109**	0.058	**0.026**
1000 × 1000_15	0.896	**0.579**	0.104	**0.041**	0.052	**0.007**
1000 × 1000_20	0.609	**0.371**	0.094	**0.012**	0.026	**0.000**
1500 × 300_5	3.498	**3.016**	**0.135**	0.281	0.023	**0.012**
1500 × 300_10	2.109	**2.022**	**0.116**	0.186	0.020	**0.019**
1500 × 300_15	**2.509**	2.781	**0.064**	0.234	**0.001**	0.006
1500 × 300_20	**2.398**	3.096	**0.100**	0.287	**0.017**	0.033
1500 × 600_5	2.642	**2.353**	**0.288**[a]	0.408	0.056[a]	**0.050**
1500 × 600_10	4.237	**4.034**	0.257[a]	**0.243**	0.076[a]	**0.037**
1500 × 600_15	3.552	**3.007**	**0.056**	0.180	**0.030**	0.035
1500 × 600_20	2.035	**1.509**	**0.089**	0.270	**0.019**	0.023
500 × 100_3	0.414	**0.322**	**0.000**	**0.000**	**0.000**	**0.000**
500 × 100_5	**0.158**	0.205	**0.000**	**0.000**	**0.000**	**0.000**
500 × 100_10	**0.103**	0.202	0.003	**0.000**	**0.000**	**0.000**
500 × 200_3	0.603	**0.427**	0.007	**0.001**	**0.000**	**0.000**
500 × 200_5	0.826	**0.520**	0.032	**0.021**	0.001	**0.000**
500 × 200_10	**0.220**	0.325	0.002	**0.000**	**0.000**	**0.000**
Average	1.598	**1.463**	**0.095**	0.134	0.023	**0.016**

[a] These averages are computed over 3 runs instead of 6.

Finally, we have investigated the impact of the number of ADPs $|K|$ on the performance of our method. Fig. 6 depicts how the performance and run times are affected by $|K|$ for a selection of 10 instances. As described earlier, choosing $|K|$ too low will lead to a rather poor performance, and our method suffers from this more than the benchmark. For all instances, the gaps shrink as $|K|$ increases, and we see them flatline towards 0 around $|K| = 100$. As for the run times on this set of instances, we observe the effect of n being large in the duration of the update phase: For many of the instances this now constitutes the biggest share of a typical iteration.

For those instances in which the reduced problem can be solved relatively quickly even for large $|K|$ (represented by the light blue lines staying underneath the dark red line), the benchmark method should be preferred. For the instances where the reduced problem is more difficult to solve, our method is more competitive timewise, as the update phase is not the main driver of computation times for these instances.

We observe the same pattern in the reported runtimes for all instance groups in Table 9: The update phase takes a more or less constant time of around 15–25 s during each iteration. If the reduced problem solves quickly, the benchmark method is faster, but if the reduced

Table 7
Average runtimes per iteration for our method and average benchmark runtimes (corrected for problem setup time), for different values of $|K|$, and method initialization & benchmark problem setup times, on the instances by Klose and Görtz (2007) & Görtz and Klose (2012).

$\|K\|$	20		75		150		Initialization	Problem setup
	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
1000 × 1000_5	9.00	4.28	31.80	32.65	86.09	84.40	17.00	16.25
1000 × 1000_10	8.05	3.79	27.65	38.85	81.00	116.40	17.17	16.32
1000 × 1000_15	7.59	3.62	25.31	26.74	65.24	51.48	17.06	16.25
1000 × 1000_20	7.34	3.32	20.54	20.54	47.02	40.23	17.00	16.36
1500 × 300_5	18.66	14.00	68.13	154.34	132.23	175.24	7.91	7.40
1500 × 300_10	6.61	3.32	23.10	53.07	56.04	85.16	7.98	7.42
1500 × 300_15	3.93	1.87	11.32	20.68	23.08	32.16	8.00	7.40
1500 × 300_20	3.64	1.83	9.76	15.96	20.46	29.20	7.96	7.44
1500 × 600_5	52.76	43.44	451.37[a]	346.36	665.55[a]	502.60	15.29	14.57
1500 × 600_10	169.38	78.09	717.48[a]	713.30	832.07[a]	746.70	15.30	14.63
1500 × 600_15	136.24	329.65	145.67	420.76	465.92	525.27	15.68	14.54
1500 × 600_20	20.24	6.27	47.16	129.17	119.40	226.99	15.59	14.70
500 × 100_3	1.40	0.78	3.85	3.30	8.82	7.18	1.01	0.91
500 × 100_5	1.35	1.03	4.58	4.38	10.96	8.99	1.00	0.91
500 × 100_10	1.07	0.75	3.44	2.28	8.38	4.42	1.01	0.90
500 × 200_3	2.34	1.23	8.56	6.11	22.54	14.92	1.91	1.81
500 × 200_5	4.82	3.27	15.08	14.93	36.48	32.95	1.92	1.76
500 × 200_10	2.94	2.84	14.19	14.00	47.88	33.39	1.91	1.79
Average $\frac{\text{Our method}}{\text{Benchmark}}$	1.812		0.947		1.142		1.071	

[a] These averages are computed over 3 runs instead of 6.

Table 8
Average gaps (%) between our objective found by method and the benchmark method (best of 40 runs) on different subsets of the Amsterdam instances.

Euclidean instances			Non-Euclidean instances		
$\|K\|$	20	100	$\|K\|$	20	100
0000–0039	−0.023	0.001	5000–5039	0.002	0.000
0070–0099	−0.023	−0.001	5070–5099	0.044	−0.004
0100–0139	−0.037	−0.001	5100–5139	−0.195	−0.001
0140–0169	−0.029	0.001	5140–5169	−0.242	−0.007
0240–0269	0.035	0.002	5240–5269	−0.072	0.001
0270–0299	−0.038	0.004	5270–5299	−0.298	−0.004
0350–0359	0.056	0.001	5350–5359	−0.146	0.001
0450–0459	0.111	0.002	5450–5459	−0.119	0.023
1040–1069	−0.004	−0.000	6040–6069	−0.017	−0.001
1070–1099	0.016	0.000	6070–6099	0.117[a]	−0.000
1100–1139	−0.010	0.002	6100–6139	−0.096	−0.002
1240–1269	0.020	0.001	6240–6269	−0.040	−0.001
1270–1299	−0.038	−0.001	6270–6299	−0.146	−0.005
3140–3169	−0.043	−0.000	8140–8169	−0.056	0.001
3200–3239	−0.046	−0.000	8200–8239	−0.207	−0.012
3270–3299	−0.151[a]	−0.002	8270–8299	−0.284	−0.010
Average	−0.021	0.000	Average	−0.109	−0.003

[a] For $|K| = 20$ on instances 3279 and 6079, an issue relating to solver precision caused our method to terminate before producing a final solution. These instances are not included in the averages.

problem is more difficult, our method is more competitive, as the MILP solution times are the main drivers for runtimes in that case.

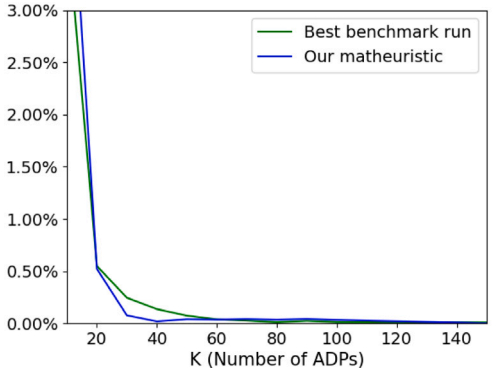
7.4. Upper bound: A posteriori analysis

As our method is based on trying to minimize Eq. (17), the results can be interpreted as being influenced by two factors: The ability of our method to indeed attain low values for this expression (which can only be computed a posteriori if the optimal solution is known), and the extent to which attaining low values here translates to finding better solutions to the location problem. In this section, we assess the ability of our method to effectively minimize the bound presented in Eq. (18). As the optimal solutions of the instances from the OR-library are known and they are very small, it is not hard to compute the upper bound for the error contribution of the ADPs (Eq. (18)) for our method and the benchmark. We have compared the mean best value of the total upper bound as presented in Eq. (17) for our method and the benchmark method on the OR library instances. The results can be found in Table 11. Clearly, our method is capable of minimizing the upper bound much more effectively than the benchmark method. We can however also conclude that this upper bound is not always a good indicator of the solution quality in practice. The fact that our method is able to minimize the upper bound more effectively does not necessarily mean that it will also find better solutions. While minimizing the upper bound, we may also at the same time be decreasing the objective value in the reduced problem for other good solutions, especially as the approach aims to learn from previously seen solutions. In the worst case, our method can be prone to overfitting to a set of solutions it has already seen. Furthermore, once our method has seen the optimal solution, it may find ADPs that suit the optimal solution very well, greatly decreasing the upper bound. In this case, low upper bounds observed in later iterations of the algorithm should be viewed with some caution. They may be less indicative of the ability of our algorithm to find the optimal solution while only having seen characteristics of good non-optimal solutions.

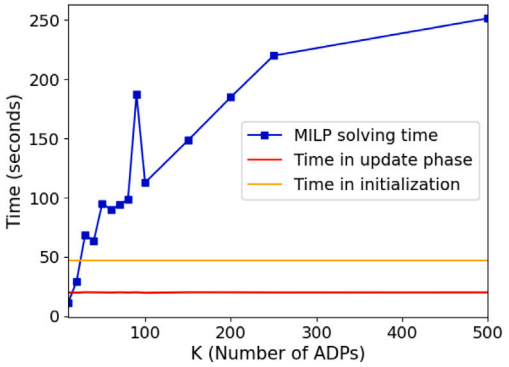
An example of the progression of the upper bound and the contribution of the individual ADPs to it for instance `cap101` from the OR library (using $K = 15$) can be found in Fig. 7. The left half of the figure shows the realization of the upper bound for the 40 benchmark runs, and the progression of the upper bound for our method , looking at the median of 30 runs. We observe that the upper bound decreases drastically after the warmup period ends and the updated $\boldsymbol{\alpha}$ values are being used. Another observation we could make is that in some of the runs for the OR-library instances, after around 20 iterations, the upper bound values would start increasing drastically again. This is caused by the creation of a single larger and imbalanced ADP, which likely has to do with the fact that the bandwidth has reached a very low value by that time. This is mainly a problem for lower values of K, and sometimes a lower value of the bandwidth can produce very good results. For numerical reasons, we do not let the bandwidth reach a value below 0.025, which probably will have some stabilizing effect; Only on very large instances with extremely narrow allocation boundaries would it be worthwhile considering altering this. Tuning the bandwidth is perhaps the most challenging aspect of our method. Especially in the presence of outliers that are served by different facilities in different good solutions, the allocation boundaries may fluctuate heavily, which could only be accounted for by tailoring the bandwidth.

Table 9
Average runtimes per iteration for our method average benchmark runtimes (corrected for problem setup time), for different values of $|K|$, and method initialization & benchmark problem setup times, on the Amsterdam instances.

$\|K\|$	20		100		Initialization	Problem setup
	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
Euclidean instances						
0000–0039	30.01	8.45	31.27	9.77	43.95	41.14
0070–0099	30.37	8.42	31.22	9.06	43.59	41.09
0100–0139	36.23	10.75	75.62	57.48	62.53	58.15
0140–0169	70.70	52.76	149.85	94.95	62.71	58.06
0240–0269	237.19	205.83	256.07	211.43	81.19	77.37
0270–0299	43.32	26.70	83.26	51.85	80.75	77.43
0350–0359	235.93	149.77	308.17	184.01	101.81	96.83
0450–0459	291.00	230.47	535.52	425.70	126.35	121.46
1040–1069	19.95	5.25	23.93	6.97	27.79	26.06
1070–1099	19.50	5.35	20.15	5.78	27.87	26.02
1100–1139	22.43	7.01	66.58	48.64	39.22	36.75
1240–1269	215.53	202.03	278.49	230.28	51.49	48.96
1270–1299	27.96	14.29	36.45	26.48	50.93	49.02
3140–3169	18.77	6.00	26.47	7.93	24.15	22.61
3200–3239	29.85	24.93	63.93	84.41	31.13	30.04
3270–3299	20.53	6.66	24.94	12.53	31.62	30.00
Average $\frac{\text{Our method}}{\text{Benchmark}}$	3.328		3.208		1.061	
Non-Euclidean instances						
5000–5039	31.17	9.32	31.65	9.74	44.16	41.24
5070–5099	31.63	9.17	31.74	9.41	43.80	41.22
5100–5139	34.59	10.60	37.21	14.52	61.54	58.42
5140–5169	36.52	11.77	68.45	36.23	61.87	58.35
5240–5269	52.04	33.48	136.44	85.59	80.63	77.21
5270–5299	40.20	12.36	45.39	15.78	81.16	77.26
5350–5359	67.37	41.89	179.96	132.49	102.38	97.05
5450–5459	94.26	70.83	290.54	179.35	127.42	121.41
6040–6069	19.88	5.78	20.75	6.53	27.81	25.96
6070–6099	20.02	5.89	20.29	6.12	27.67	25.99
6100–6139	22.21	6.70	24.30	8.43	39.02	36.83
6240–6269	44.12	31.34	129.14	48.22	61.34	48.80
6270–6299	25.98	7.69	34.92	17.02	51.36	48.81
8140–8169	18.86	6.46	20.58	8.37	24.08	22.52
8200–8239	21.34	7.10	29.47	19.15	31.53	29.99
8270–8299	20.45	6.70	22.43	8.55	31.39	30.07
Average $\frac{\text{Our method}}{\text{Benchmark}}$	3.191		3.184		1.064	



(a) Overview of performance of the matheuristic and the best of 40 benchmark runs for different values of $|K|$.



(b) Overview of runtimes of different phases in the matheuristic and of the benchmark method for different values of $|K|$.

Fig. 6. Overview of performance and run times for different values of $|K|$ compared to best known solution, averaged over a selection of 10 real scale instances with differing characteristics.

8. Capacitated *p*-median problem

To assess the applicability of our methodology for a wider range of related location problems, we also applied our matheuristic to a set of capacitated *p*-median problem (CPMP) instances from the literature (Lorena and Senne, 2003; Stefanello et al., 2014b; Gnägi and Baumann, 2021). This problem differs in some aspects from the CFLP, as can be seen by inspecting its formulation in Eqs. (26) – (31):

- Instead of assigning a cost to each opened facility in the objective, the number of facilities to open is fixed beforehand to p.
- The distance costs to minimize are not scaled by the demands of the individual demand points; These are only used when it comes to the capacity constraints.
- The problem does not allow multi sourcing: Each demand point has to be served by exactly one facility.

Table 10
Iteration at which best value was attained, and at which our algorithm was terminated, averaged over 5 instances per set and 6 runs each.

$\|K\|$	Best value at iteration			Stop at iteration		
	20	75	150	20	75	150
1000 × 1000_5	24.5	28.7	34.0	39.8	40.0	40.0
1000 × 1000_10	26.7	34.9	28.0	40.0	40.0	39.9
1000 × 1000_15	30.2	30.2	21.6	40.0	40.0	38.9
1000 × 1000_20	30.7	24.2	15.6	40.0	39.5	36.9
1500 × 300_5	6.9	21.7	11.7	30.6	38.5	34.7
1500 × 300_10	16.2	21.1	10.6	36.5	38.5	33.2
1500 × 300_15	18.9	19.1	13.1	37.1	39.2	36.4
1500 × 300_20	23.2	20.9	14.4	38.8	39.0	36.2
1500 × 600_5	5.1	17.9[a]	9.7[a]	29.1	22.8[a]	17.7[a]
1500 × 600_10	8.6	9.9[a]	5.4[a]	31.6	16.3[a]	14.0[a]
1500 × 600_15	8.0	20.7	11.1	31.6	39.0	26.2
1500 × 600_20	18.1	21.7	13.1	37.0	39.1	36.0
500 × 100_3	14.8	8.3	3.4	34.6	32.0	27.4
500 × 100_5	16.1	6.8	2.8	36.3	30.6	26.8
500 × 100_10	18.9	8.4	5.4	38.4	32.0	29.4
500 × 200_3	6.6	12.8	6.0	30.6	35.5	30.0
500 × 200_5	15.7	11.7	6.3	35.3	34.0	29.9
500 × 200_10	22.5	6.9	3.4	39.1	30.4	26.6

[a] These averages are computed over 3 runs instead of 6.

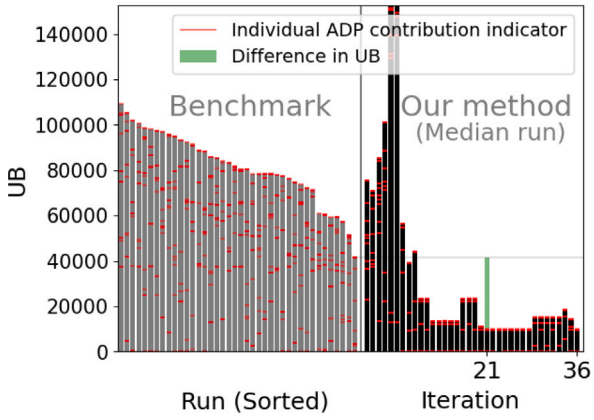


Fig. 7. Evaluation of the error bound from Eq. (18) on one of the instances from the OR library: The 40 benchmark runs are compared to the worst, median and best algorithm progression. Red marks indicate the contribution of different ADPs.

- In general, the instances of the CPMP that are proposed in the literature have the characteristic that the set of demand points and the set of potential facility locations coincide (And thus, $n = m > p$).

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{i\in[m]} \sum_{j\in[n]} d_{ij} x_{ij} \tag{26}$$

$$\sum_{i\in[m]} x_{ij} = 1 \;\forall\, j \in [n] \tag{27}$$

$$\sum_{j\in[n]} D_j x_{ij} \le Q_i y_i \;\forall\, i \in [m] \tag{28}$$

$$\sum_{i\in[m]} y_i = p \tag{29}$$

$$y_i \in \{0,1\} \;\forall\, i \in [m] \tag{30}$$

$$x_{ij} \in \{0,1\} \;\forall\, i \in [m], j \in [n] \tag{31}$$

Considering this, we made the following adjustments to our algorithm in order to apply it to instances of the CPMP:

- We limit the warmup period to 6 iterations;
- We choose a higher value for ξ_{max}, because the number of potential facilities is higher;
- In Phase 2, we solve the reduced problem as a problem where multi-sourcing is allowed (relax constraint (31)). This is because the ADP's summed demand can be bigger than the capacity of the facilities, rendering the problem infeasible. Relaxing this constraint means that the objective of the reduced problem no longer is an upper bound to the objective of the original problem, as constructing a feasible solution to the original problem from the reduced problem is no longer guaranteed;
- The allocation problem to solve after the facilities have been chosen, remains a binary program and is thus harder to solve than the LPs we are dealing with when applying the method to the CFLP. We therefore decided to limit the solution time of this binary program as well;
- The distances of the ADPs to the facilities are computed by taking the unweighted mean of the distances of the DPs that belong to it. We also apply unweighted k-means clustering for this reason;

For our experiments, we kept the remainder of the implementation intact, as the aim of these experiments is to investigate whether it could be worthwhile to consider a modified version of our algorithm, for application to different settings, and not to provide a different heuristic tailored to this problem. Due to the fact m and n are both high, our implementation becomes rather slow and memory and resource intensive: In terms of runtimes, our method's performance will not be comparable to state-of-the-art approaches for this problem, which are designed to avoid large computational overheads caused by massive distance matrices (see e.g. Gnägi and Baumann, 2021). To reduce these computation times, the method should be extended to aggregate the potential facilities as well. This is out of scope for this study, but we sketch an outline of how this could be achieved in Section 9.

For the smallest set of instances (`SJC1-SJC4b`, Lorena and Senne, 2003), we could keep the computational limits similar to the CFLP experiments: We used 32 cores and 10 solver threads and limited the solver time for the reduced problem to 20 minutes. We set ξ_{max} to 10, 15, and 25 for instances `SJC1`, `SJC2` and `SJC3a-4b`, respectively. We ran our heuristic 6 times on each of these instances for different values of $|K|$; The results for $|K| = 50, 75$ and 150 can be found in Table 12. Due to the relatively low number of solutions our method sees, it does not attain optimal values on these instances. However, the gaps attained are reasonably low, which suggests that our method can also be applied successfully to this problem, and a broader range of location problems in general.

Furthermore, we observed staggering gaps when selecting $|K| < p$. This is similar to our earlier observation for the CFLP. We conclude that our method, and (demand) aggregation in general is most exploitable in those problems where $\frac{n}{p}$ is higher. Furthermore, lower values of p also lead to a more compact allocation problem, which decreases the computational burden needed for obtaining the allocation. Keeping these observations in mind, we also ran our method on somewhat larger instances with appropriate values for $|K|$, that are nonetheless still considered small or medium-scale instances for the CPMP (Gnägi and Baumann, 2021). As our method would only produce qualitative solutions on the instances `fnl4461_0250` - `fnl4461_1000` for rather large values of $|K|$, and since we are not mainly focusing on runtime for this analysis, we relaxed our computational limits to ensure that 15 iterations of our algorithm could be completed, as a prerequisite to a meaningful analysis of its performance. For all the `fnl` instances, we ran our heuristic three times. We limited the solver time for the reduced problem to 20 min (which was almost always reached) and for the allocation problem to 10 min. Using 32 cores for most experiments: for one set of runs we had to use 64 cores instead, and for one set of runs, 2 out of 3 runs encountered an Out of Memory condition. The results of these experiments can be found in Table 13. The largest

Table 11
Mean best value of Eq. (17) for our method and the benchmark method on the OR library instances.

K	Our method	Benchmark	Our method	Benchmark	Our method	Benchmark
	6		15		21	
cap41-44	173,450.49	**172,713.21**	**18,822.01**	49,378.66	**7,787.17**	20,867.01
cap51	**106,775.73**	123,894.88	**5,336.64**	26,123.95	**60.63**	10,161.16
cap61-64	**86,169.81**	108,074.11	**17.96**	26,333.14	**0.00**	9,942.36
cap71-74	**70,893.01**	95,974.62	**0.00**	20,716.84	**0.00**	7,716.83
cap81-84	**227,577.83**	229,006.23	**11,230.77**	38,517.32	**1,046.00**	11,713.31
cap91-94	181,600.24	**162,034.11**	**2,465.45**	22,409.42	**0.00**	5,258.88
cap101-104	**139,087.79**	144,906.74	**2,465.45**	17,961.71	**0.00**	5,543.29
cap111-114	**212,782.99**	217,393.71	**13,050.62**	38,856.84	**235.77**	13,216.63
cap121-124	187,666.61	**170,021.47**	**2,465.45**	22,071.63	**0.00**	4,722.28
cap131-134	**136,316.31**	145,846.35	**2,465.45**	19,728.87	**0.00**	6,314.87

Table 12
Mean objectives, optimality gaps and runtimes of our algorithm on the SJC test instances of the capacitated p-median problem.

$\|K\|$	50			75			150		
Instance	Objective	Gap (%)	Time (s)	Objective	Gap (%)	Time (s)	Objective	Gap (%)	Time (s)
SJC1	17,374.65	0.495	26.58	17,381.57	0.536	50.90	–	–	–
SJC2	33,366.63	0.288	56.82	33,307.56	0.110	77.82	33,278.43	0.023	224.51
SJC3a	45,573.77	0.526	89.79	45,531.85	0.434	114.23	45,478.44	0.316	249.19
SJC3b	40,854.63	0.538	83.49	40,769.86	0.330	104.52	40,650.77	0.037	243.14
SJC4a	62,630.41	1.138	197.94	62,278.54	0.570	451.49	61,981.38	0.090	631.22
SJC4b	52,920.68	0.882	141.55	52,721.83	0.503	165.47	52,552.58	0.180	318.60

Table 13
Mean objectives, gaps to best reported solution and runtimes of our algorithm on the FNL and XMC test instances of the capacitated p-median problem.

Instance	$m = n$	p	$\|K\|$	Cores	Solver threads	Runs	Iters	Objective	Gap (%)	Time (s)
fnl4461_0020	4461	20	150	32	10	3	15	1,284,644.25	0.086	5,138.42
			450	32	10	3	15	1,283,663.31	0.010	18,472.56
fnl4461_0100	4461	100	150	32	10	3	15	552,962.95	0.739	19,899.61
			450	32	10	3	15	549,574.63	0.121	15,022.72
fnl4461_0250	4461	250	450	32	10	3	15	339,685.76	1.130	21,706.29
			750	64	10	3	15	337,895.06	0.597	22,391.37
fnl4461_0500	4461	500	750	32	10	3	15	228,784.92	1.835	23,107.44
			1250	32	10	1	15	227,059.38	1.067	24,344.54
fnl4461_1000	4461	1000	1250	32	10	3	15	150,433.67	3.134	26,622.65
XMC10500_100	10 500	100	150	192	10	3	15	178,909.18	−1.412	49,515.34
			450	192	10	3	15	177,379.75	−2.255	50,233.44

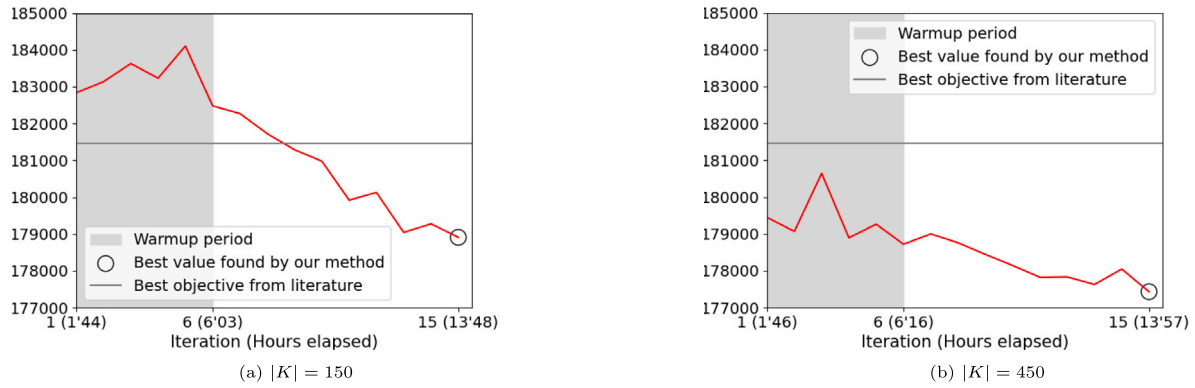
instance on which we ran our matheuristic is instance XMC10500_100. For this instance, we set the solver times for the reduced and allocation problems to 40 and 20 min, respectively. We were able to run our method using 192 cores with $|K| = 150$ and $|K| = 450$, and this instance marks the computational limit of applying our current method without significant additional problem-reducing mechanisms. The optimality gaps we obtained are competitive to those obtained by state-of-the-art methods (Gnägi and Baumann, 2021). As mentioned before, these methods are much more efficient in terms of runtime. Nonetheless, the fact that our method is able to find solutions that are closer to the optimal value, especially for those instances where p is low, suggests promising prospects if our method were to be applied in combination with other complexity reducing approaches. On instance XMC_10500_100, our method was able to find a new best solution compared to the one reported by Gnägi and Baumann, once more highlighting the potential of our method when $\frac{n}{p}$ is higher. Fig. 8 shows the progression of our heuristic on 2 runs with different values of $|K|$ for this instance. The progression indicated that even during the warmup period, qualitative solutions were found, and our method was able to improve on those during the remaining iterations.

9. Conclusions & future work

In this work, we investigated how aggregation techniques can be employed for solving instances of the CFLP with a large number of demand points. We have addressed how information on the locations and parameters of the original problem can be used to arrive at an aggregated problem whose objective function approaches that of the original problem in the neighborhood of the optimal solution. Our heuristic method relies heavily on inference about the optimal solution from previously seen good solutions. This means that our method, while having the potential to find better aggregations than the benchmark, is quite sensitive to the hyperparameter selection and the solutions seen during the warm-up period. Furthermore, our method works well under the assumption that the number of ADPs is comfortably higher than the number of facilities to be opened in the optimal solution, and under the assumption that the complexity emerging from the number of potential locations is not too big. If these conditions are not met, a good performance is not guaranteed.

As we have seen the importance of hyperparameter tuning, it would be an interesting topic for further research to scrutinize this aspect, both from a theoretical and an empirical angle. It would also be interesting to investigate whether our method can be adapted to solve more involved variants of the CFLP, such as CFLPs with multiple layers of facilities, or stochastic versions of the CFLP.

Finally, the method presented in this paper is mainly suited for application to instances of capacitated location problems for which there is a small number of facilities but a very large number of demand points. The behavior of the current method for instances with a very large number of potential facilities requires a large computational load, as clustering has to be done in m-dimensional space, and the



(a) $|K| = 150$ (b) $|K| = 450$

Fig. 8. Overview of the progression of our method on 2 runs ($|K| = 150$ and 450, respectively) for instance `XMC10500_100`.

reduced problem may still be hard due to the plethora of potential facilities that can be opened. Under certain restrictive assumptions, the proposed aggregation method could also be applied to aggregate potential facility locations. This would mainly be of interest for solving capacitated location problems where m is large, for example in settings where each demand point is also a potential facility location, such as the capacitated p-median problem. As mentioned before, Approaches aiming to solve very large instances of this type of problem fastly and efficiently have to be designed to avoid large computational overheads caused by massive distance matrices (Gnägi and Baumann, 2021).

By also aggregating the potential facility locations that correspond to the demand locations being aggregated, the complexity arising from a large number of potential facilities could be reduced. This effectively transforms an MILP with m binary variables into a reduced MILP with l integer variables. Similarly to the demand aggregation case, every facility could be associated with a vector in $\mathbb{R}^n$ of transformed distances to or from demand points. Based on these vectors, facilities can be clustered, such that facilities with similar vectors can be treated as a single bigger facility in a reduced model. This approach would mainly applicable to problems for which at least one of the following is true:

1. $c_i = C_1$, where C_1 is some constant, for all potential facilities i
2. $Q_i = C_2$, where C_2 is some constant, for all potential facilities i

The capacitated p-median problem fulfills (at least) the first condition. An outline for an approach would include the following steps:

1. Aggregate the points based on a limited set of promising facility locations using our method.
2. Aggregate the demand volumes for the ADPs.
3. For each ADP k, instantiate an aggregated facility with incremental (integer) capacity steps of size $\frac{\sum_{i:M(i)=k} Q_i}{|\{i:M(i)=k\}|}$, and associated cost per capacity step of $\frac{\sum_{i:M(i)=k} c_i}{|\{i:M(i)=k\}|}$.
4. Solve the reduced problem exactly.
5. Use the solution of the reduced problem to enforce constraints on the original problem that indicate how many facilities should be open in each cluster, and re-solve the original problem using these constraints.

We see two main issues that would need to be addressed first in order to make this approach viable. The first issue concerns the computation of the distances between the aggregated facilities and the demand points. Assuming we would apply this approach to problems where multi-sourcing is allowed, it is still unknown whether the property can be preserved that, given a solution to the reduced model, a solution with equal objective can be constructed to the original problem. This is due to the fact that the exact combination of facilities that will be opened within a cluster is not known in advance: Taking the weighted average distance over all facilities that are aggregated together will not strictly yield an upper bound for the transportation cost to any possible subset of these facilities. Secondly, the final step may still yield a computationally untractable problem, for which a further heuristic approach may need to be considered: As an example of this, applying this approach to a capacitated p-median problem and choosing $K = 1$ will again yield the original problem in the final step. The further fleshing out of this approach and delving deeper into the aforementioned issues is out of scope for this paper, but would be a very interesting topic for future research.

CRediT authorship contribution statement

R.J.W. Buijs: Writing – original draft, Visualization, Methodology, Investigation, Conceptualization. **R.D. van der Mei:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **E.R. Dugundji:** Writing – review & editing, Supervision, Conceptualization. **S. Bhulai:** Writing – review & editing, Supervision, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the revision process, the writers used a Github Copilot as an extension to the text editing software. The main purpose of the tool was to speed-up the formatting, but sometimes its textual suggestions were also used to improve the readability of the text. These suggestions have always been reviewed and approved by the authors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was carried out in context of the Circular Foam project, that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101036854. The authors would like to express their gratitude to two anonymous referees, whose remarks helped to greatly improve this paper. Finally, the authors would like to thank Berend Markhorst, Joris Slootweg, and Jesse Nagel for the insightful discussions during the different stages of this study.

Appendix A. Proofs of Propositions 1 and 2

A.1. Proof of Proposition 1

Proof. We recall Eq. (18); The contribution of an individual ADP is equal to $\sum_{j:M(j)=k}\sum_{i\in[m]} D_j(\tilde{d}_{ij}-d_{ij})\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}$. This equation can be written out as follows (knowing that $\tilde{d}_{ij}=d_{ik}:=\sum_{j':M(j')=k}\frac{D_{j'}d_{ij'}}{D_k}$):

$$\sum_{i\in H_k}\sum_{j:M(j)=k} D_j\left(\sum_{j':M(j')=k}\frac{D_{j'}d_{ij'}}{D_k}-d_{ij}\right)\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}$$

Multiplying d_{ij} by $1=\frac{D_k}{D_k}$ and using $D_k=\sum_{j':M(j')=k}D_{j'}$ gives

$$\sum_{i\in H_k}\sum_{j:M(j)=k} D_j\left(\frac{-\sum_{j':M(j')=k;j'\neq j}D_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij}+\sum_{j':M(j')=k;j'\neq j}\frac{D_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij'}\right)$$

Combining the summations and rearranging gives

$$\sum_{i\in H_k}\sum_{j:M(j)=k}\sum_{j':M(j')=k;j'\neq j}-\frac{D_jD_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij}+\frac{D_jD_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij'}$$

Writing the sum twice, reversing the order of j and j' gives

$$\frac{1}{2}\left[\sum_{i\in H_k}\sum_{j:M(j)=k}\sum_{j':M(j')=k;j'\neq j}-\frac{D_jD_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij}+\frac{D_jD_{j'}}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij'}\right.$$
$$\left.+\sum_{i\in H_k}\sum_{j':M(j')=k}\sum_{j:M(j)=k;j\neq j'}-\frac{D_{j'}D_j}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}d_{ij'}+\frac{D_{j'}D_j}{D_k}\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}d_{ij}\right]$$

Merging the two sums yields

$$\frac{1}{2D_k}\left[\sum_{i\in H_k}\sum_{j:M(j)=k}\sum_{j':M(j')=k;j'\neq j}D_jD_{j'}\left(-\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij}+\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}d_{ij'}-\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}d_{ij'}+\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}d_{ij}\right)\right]$$

By simplifying and acknowledging that $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}-\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}=0$ when $j=j'$, we arrive at

$$\frac{1}{2D_k}\sum_{i\in H_k}\sum_{j:M(j)=k}\sum_{j':M(j')=k}D_jD_{j'}(\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij}-\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'})(d_{ij'}-d_{ij})\quad\square$$

A.2. Proof of Proposition 2

Proposition 2. Suppose that for every tuple (j,j') there are exactly two corresponding elements in G_k: (i,j,j') and (i',j,j'). If (i,j,j') and (i',j,j') are two elements of G_k, the contribution to the error UB corresponding to these two elements is proportional to and increasing in $|(d_{i'j}-d_{i'j'})-(d_{ij}-d_{ij'})|$, D_j and $D_{j'}$.

Proof. Denote by r_i and $r_{i'}$ the remaining capacities of facilities i and i' if the optimal solution $\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})$ is applied at all points besides j and j', with the values x_{ij}, $x_{ij'}$, $x_{i'j}$, $x_{i'j'}$ still set to 0. Note that r_1 and r_2 must be positive, since we know by definition that x_{ij} and $x_{ij'}$ cannot both be 0, and the same holds for $x_{i'j}$ and $x_{i'j'}$. Also, $r_i+r_{i'}\geq D_j+D_{j'}$, since all demand must be fulfilled by facilities i and i'. We will prove the Proposition by making inference about the possible values of x_{ij}, $x_{ij'}$, $x_{i'j}$, and $x_{i'j'}$, in relation to d_{ij}, $d_{ij'}$, $d_{i'j}$, and $d_{i'j'}$. We distinguish three main cases:

Case 1: $d_{ij}-d_{i'j}<d_{ij'}-d_{i'j'}$
We can distinguish three scenarios that may have caused the tuples (i,j,j') and (i',j,j'): to be in G_k: The first possibility is that one facility is closer to both demand points, but it does not have enough remaining capacity to fulfill the complete demand of both. This can be either facility i or i'.

Scenario 1a: If facility i is closer to both demand points, both $d_{ij}-d_{i'j}$ and $d_{ij'}-d_{i'j'}$ are negative, but the difference in distance is bigger for demand point j. The solution that adds the lowest cost for $x_{ij}, x_{ij'}, x_{i'j}$ and $x_{i'j'}$ is to greedily allocate the remaining capacities according to

$$x_{ij}=\min\left\{\frac{r_i}{D_j},1\right\}$$

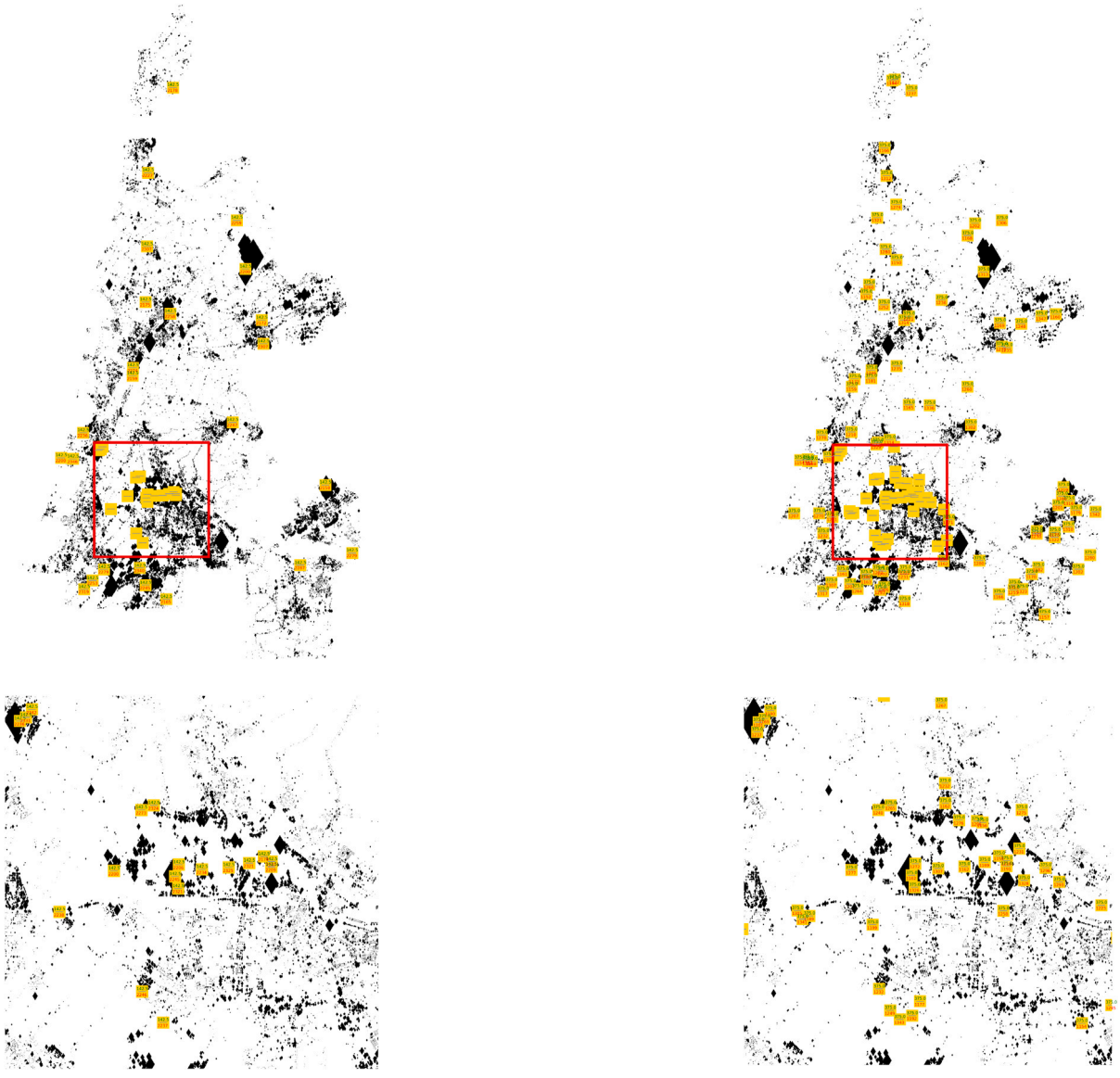
$$x_{i'j'}=\min\left\{\frac{D_j+D_{j'}-r_i}{D_{j'}},1\right\}$$

$$x_{ij'}=1-x_{i'j'}$$

$$x_{i'j}=1-x_{ij}$$

Table B.14
Test bed groupings and specifications.

Test Bed	Instances	m	n	Set of values used for $\|K\|$	Facilities open (opt. sol.)	Cores used	Solver max threads	Benchmark runs	Heuristic runs	Valid inequalities?
OR-library	cap41–44	16	50	{6,9,12,15,18,21}	12–13	16	–	40	30	–
	cap51	16	50	{6,9,12,15,18,21}	8	16	–	40	30	–
	cap61–64	16	50	{6,9,12,15,18,21}	5–11	16	–	40	30	–
	cap71–74	16	50	{6,9,12,15,18,21}	4–11	16	–	40	30	–
	cap81–84	25	50	{6,9,12,15,18,21}	13–17	16	–	40	30	–
	cap91–94	25	50	{6,9,12,15,18,21}	7–15	16	–	40	30	–
	cap101–104	25	50	{6,9,12,15,18,21}	4–15	16	–	40	30	–
	cap111–114	50	50	{6,9,12,15,18,21}	13–17	16	–	40	30	–
	cap121–124	50	50	{6,9,12,15,18,21}	7–15	16	–	40	30	–
	cap131–134	50	50	{6,9,12,15,18,21}	4–15	16	–	40	30	–
	cap201–204 (capa)	100	1000	{6,9,12,15,18,21,35,70}	4–7	16	–	40	30	–
	cap211–214 (capb)	100	1000	{6,9,12,15,18,21,35,70}	7–11	16	–	40	30	–
	cap221–224 (capc)	100	1000	{6,9,12,15,18,21,35,70}	9–11	16	–	40	30	–
	cap231–234	100	1000	{6,9,12,15,18,21,35,70}	5–7	16	–	40	30	–
	cap241–244	100	1000	{6,9,12,15,18,21,35,70}	4–7	16	–	40	30	–
	cap251–254	100	1000	{6,9,12,15,18,21,35,70}	7–10	16	–	40	30	–
	cap261–264	100	1000	{6,9,12,15,18,21,35,70}	19	16	–	40	30	–
	cap271–274	100	1000	{6,9,12,15,18,21,35,70}	33	16	–	40	30	–
	cap281–284	100	1000	{6,9,12,15,18,21,35,70}	17	16	–	40	30	–
Görtz & Klose, Klose & Görtz	1000 × 1000_10	1000	1000	{20,75,150}	56–58	32	10	40	6	✓
	1000 × 1000_15	1000	1000	{20,75,150}	39–41	32	10	40	6	✓
	1000 × 1000_20	1000	1000	{20,75,150}	29–31	32	10	40	6	✓
	1000 × 1000_5	1000	1000	{20,75,150}	114–118	32	10	40	6	✓
	1500 × 300_10	300	1500	{20,75,150}	22–24	32	10	40	6	✓
	1500 × 300_15	300	1500	{20,75,150}	23–25	32	10	40	6	✓
	1500 × 300_20	300	1500	{20,75,150}	22–27	32	10	40	6	✓
	1500 × 300_5	300	1500	{20,75,150}	36–38	32	10	40	6	✓
	1500 × 600_10	600	1500	{20,75,150}	35–37	32	10	40	6	✓
	1500 × 600_15	600	1500	{20,75,150}	24–26	32	10	40	6	✓
	1500 × 600_20	600	1500	{20,75,150}	21–23	32	10	40	6	✓
	1500 × 600_5	600	1500	{20,75,150}	69–72	32	10	40	6	✓
	500 × 100_10	100	500	{20,75,150}	10–12	16	–	40	6	✓
	500 × 100_3	100	500	{20,75,150}	21–23	16	–	40	6	✓
	500 × 100_5	100	500	{20,75,150}	12–15	16	–	40	6	✓
	500 × 200_10	200	500	{20,75,150}	12–14	16	–	40	6	✓
	500 × 200_3	200	500	{20,75,150}	39–45	16	–	40	6	✓
	500 × 200_5	200	500	{20,75,150}	24–26	16	–	40	6	✓
Newly generated instances (Euclidean)	0000–0099	42	57 854	{20,100}	–	16	10	40	1	✓
	0100–0199	60	57 854	{20,100}	–	16	10	40	1	✓
	0200–0299	80	57 854	{20,100}	–	32	10	40	1	✓
	0350–0359	100	57 854	{20,100}	–	32	10	40	1	✓
	0450–0459	125	57 854	{20,100}	–	32	10	40	1	✓
	1000–1099	42	36 635	{20,100}	–	16	10	40	1	✓
	1100–1199	60	36 635	{20,100}	–	16	10	40	1	✓
	1200–1299	80	36 635	{20,100}	–	32	10	40	1	✓
	1350–1359	100	36 635	{20,100}	–	32	10	40	1	✓
	1450–1459	125	36 635	{20,100}	–	32	10	40	1	✓
	3100–3199	60	22 540	{20,100}	–	16	10	40	1	✓
	3200–3299	80	22 540	{20,100}	–	32	10	40	1	✓
	3350–3359	100	22 540	{20,100}	–	32	10	40	1	✓
	3450–3459	125	22 540	{20,100}	–	32	10	40	1	✓
Newly generated instances (over the road)	5000–5099	42	57 854	{20,100}	–	16	10	40	1	✓
	5100–5199	60	57 854	{20,100}	–	16	10	40	1	✓
	5200–5299	80	57 854	{20,100}	–	32	10	40	1	✓
	5350–5359	100	57 854	{20,100}	–	32	10	40	1	✓
	5450–5459	125	57 854	{20,100}	–	32	10	40	1	✓
	6000–6099	42	36 635	{20,100}	–	16	10	40	1	✓
	6100–6199	60	36 635	{20,100}	–	16	10	40	1	✓
	6200–6299	80	36 635	{20,100}	–	32	10	40	1	✓
	6350–6359	100	36 635	{20,100}	–	32	10	40	1	✓
	6450–6459	125	36 635	{20,100}	–	32	10	40	1	✓
	8100–8199	60	22 540	{20,100}	–	16	10	40	1	✓
	8200–8299	80	22 540	{20,100}	–	32	10	40	1	✓
	8350–8359	100	22 540	{20,100}	–	32	10	40	1	✓
	8450–8459	125	22 540	{20,100}	–	32	10	40	1	✓



(a) Instance 0000

(b) Instance 1476

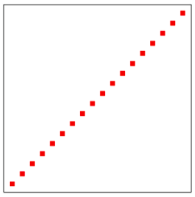
Fig. C.9. Overview of two of our generated instances. Facilities are annotated with cost and capacity. White diamonds represent demand points, with sizes proportional to the demand volumes.

Scenario 1b: If facility i' is closer to both demand points, both $d_{ij} - d_{i'j}$ and $d_{ij'} - d_{i'j'}$ are positive, but the difference in distance is bigger for demand point 2. The solution that adds the lowest cost for $x_{ij}, x_{ij'}, x_{i'j}$ and $x_{i'j'}$ is to greedily allocate the remaining capacities according to

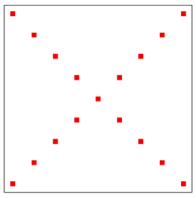
$$x_{ij} = \min\left\{\frac{D_j + D_{j'} - r_{i'}}{D_j}, 1\right\}$$

$$x_{i'j'} = \min\left\{\frac{r_{i'}}{D_{j'}}, 1\right\}$$

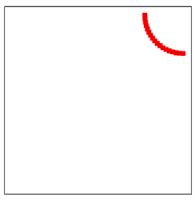
$$x_{ij'} = 1 - x_{i'j'}$$



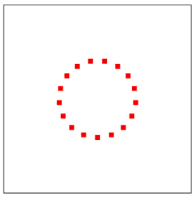
(a) A diagonal line of n points



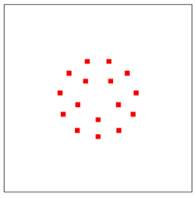
(b) A cross consisting of 2 diagonal lines containing $\frac{n+1}{2}$ and $\frac{n-1}{2}$ points, respectively



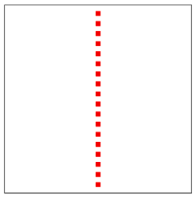
(c) A dense quarter-circle of n points in one of the corners of the plane



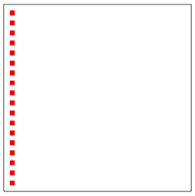
(d) A circle of n points in the center of the plane



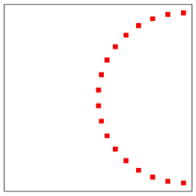
(e) 2 concentric circles of $\frac{2n}{3}$ and $\frac{n}{3}$ points in the center of the plane



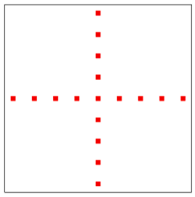
(f) A vertical line of n points in the center of the plane



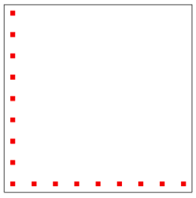
(g) A vertical line of n points at the edge of the plane



(h) A half circle of n points in one half of the plane



(i) A cross consisting of a vertical line of $\frac{n+1}{2}$ points and a horizontal line of $\frac{n-1}{2}$ points in the center of the plane



(j) An L-shape consisting of a vertical line of $\frac{n+1}{2}$ points and a horizontal line of $\frac{n-1}{2}$ points on the edges of the plane

Fig. C.10. 10 different positional patterns of potential facilities, serving as computational testing environments to get insights into parameterization of the core method.

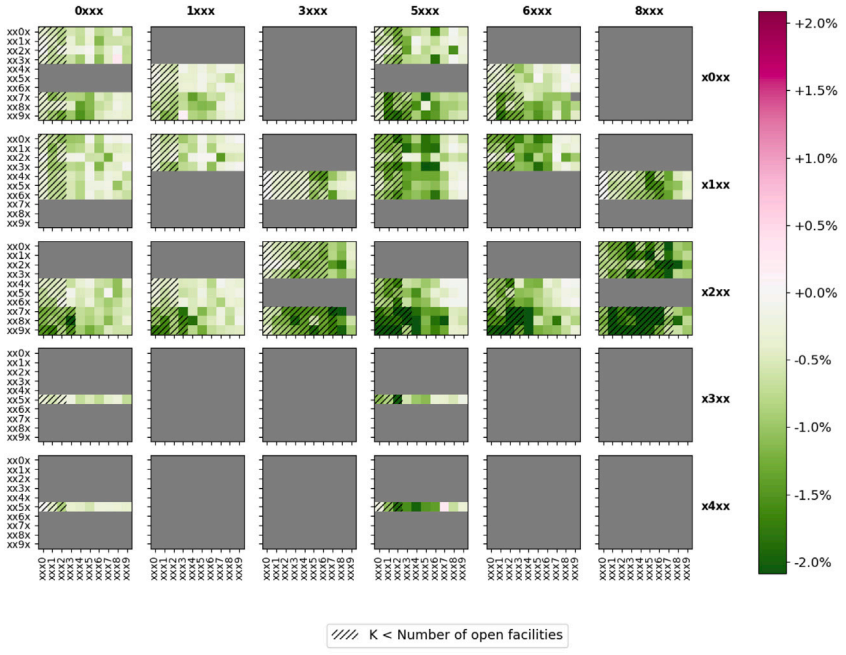
$$x_{i'j} = 1 - x_{ij}$$

Scenario 1c: The final possibility is that facility i is closer to demand point j, while facility i' is closer to demand point j'. In this case, the solution that adds the lowest cost for $x_{ij}, x_{ij'}, x_{i'j}$ and $x_{i'j'}$ is to greedily allocate the remaining capacities according to

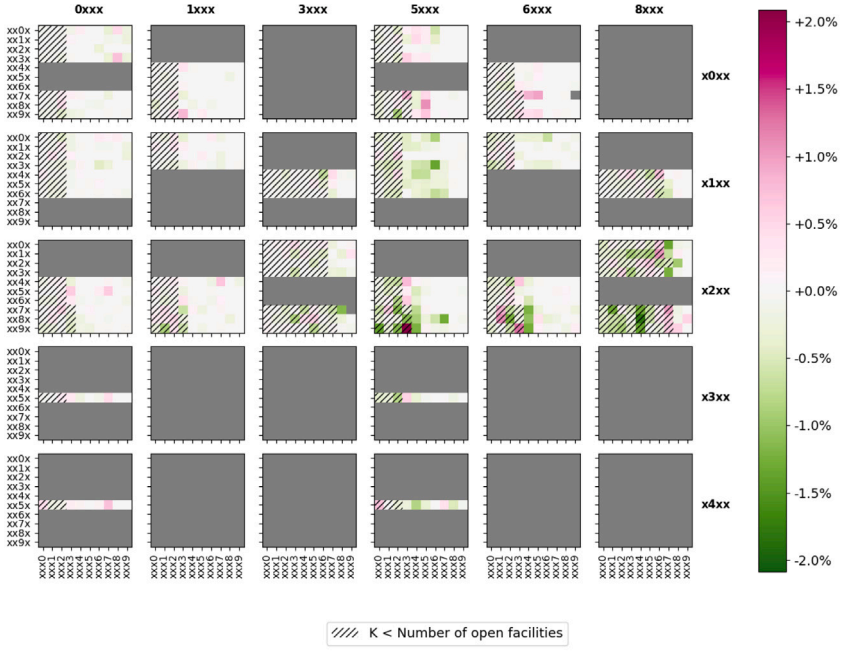
$$x_{ij} = \min\left\{\frac{r_i}{D_j}, 1\right\}$$

$$x_{i'j'} = \min\left\{\frac{r_{i'}}{D_{j'}}, 1\right\}$$

$$x_{ij'} = 1 - x_{i'j'}$$

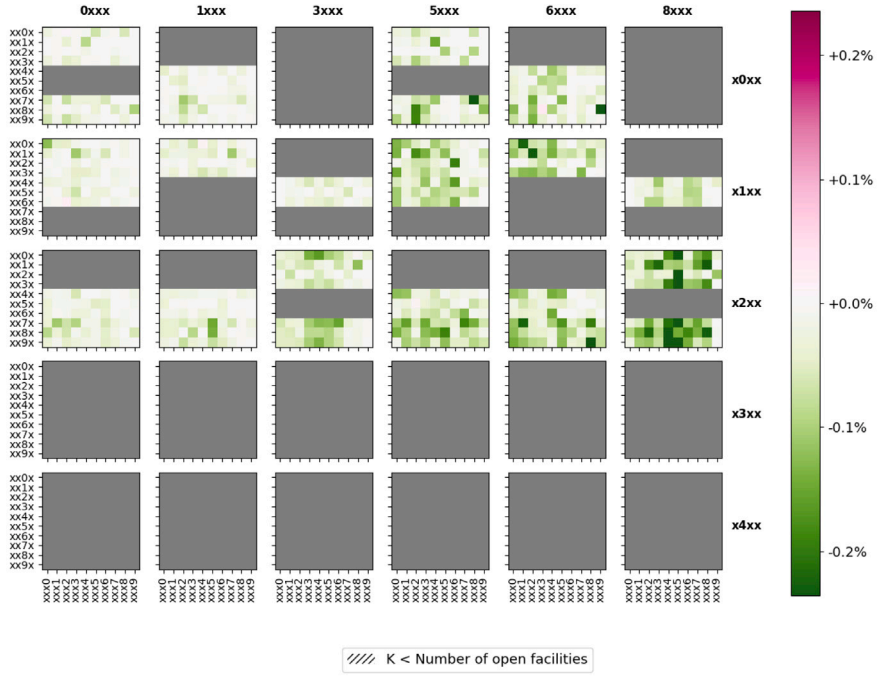


(a) Heuristic performance on Amsterdam instances compared to mean benchmark performance

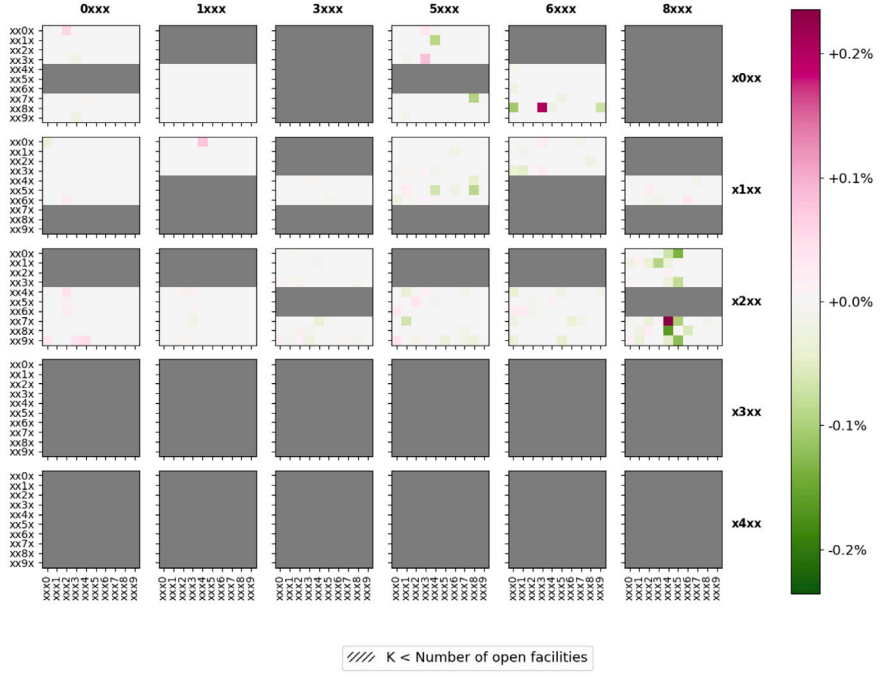


(b) Heuristic performance on Amsterdam instances compared to best benchmark performance

Fig. D.11. Breakdown of algorithm performance relative to benchmark objectives, setting $|K| = 20$. Negative values (in green) indicate instances where the heuristic attained a better objective than the relevant benchmark objective.



(a) Heuristic performance on Amsterdam instances compared to mean benchmark performance



(b) Heuristic performance on Amsterdam instances compared to best benchmark performance

Fig. D.12. Breakdown of algorithm performance relative to benchmark objectives, setting $|K| = 100$. Negative values (in green) indicate instances where the heuristic attained a better objective than the relevant benchmark objective.

$$x_{i'j} = 1 - x_{ij}$$

We observe that, in all scenarios, either x_{ij} or $x_{i'j'}$ (or both) must be equal to 1, and both must be strictly greater than 0. From this it follows that, in the optimal solution, $x_{ij} > x_{ij'}$:
We conclude that in this case $(\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}) > 0$ while also $(d_{i'j} - d_{ij} + d_{ij'} - d_{i'j'}) > 0$. This shows that, in this case, the error is positively proportional to $|(d_{i'j} - d_{ij} + d_{ij'} - d_{i'j'})| = |(d_{i'j} - d_{i'j'}) - (d_{ij} - d_{ij'})|$

Case 2: $d_{ij} - d_{i'j} > d_{ij'} - d_{i'j'}$
Similar to the previous case, we can distinguish three scenarios and deduct the optimal allocations:

Scenario 2a: Facility i is closer to both demand points:

$$x_{ij'} = \min\left\{\frac{r_i}{D_j}, 1\right\}$$

$$x_{i'j} = \min\left\{\frac{D_j + D_{j'} - r_i}{D_{j'}}, 1\right\}$$

$$x_{ij} = 1 - x_{i'j}$$

$$x_{i'j'} = 1 - x_{ij'}$$

Scenario 2b: Facility i' is closer to both demand points:

$$x_{ij'} = \min\left\{\frac{D_j + D_{j'} - r_{i'}}{D_j}, 1\right\}$$

$$x_{i'j} = \min\left\{\frac{r_{i'}}{D_{j'}}, 1\right\}$$

$$x_{ij} = 1 - x_{i'j}$$

$$x_{i'j'} = 1 - x_{ij'}$$

Scenario 2c: Facility i is closer to demand point j', facility i' is closer to demand point j.

$$x_{ij'} \geq \min\left\{\frac{r_i}{D_j}, 1\right\}$$

$$x_{i'j} \geq \min\left\{\frac{r_{i'}}{D_{j'}}, 1\right\}$$

$$x_{ij} = 1 - x_{i'j}$$

$$x_{i'j'} = 1 - x_{ij'}$$

Using the same argument as in the previous case, we may conclude that $(\boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij} - \boldsymbol{x}^*_{\boldsymbol{d}}(\boldsymbol{y}^*_{\boldsymbol{d}})_{ij'}) < 0$ while also $(d_{i'j} - d_{ij} + d_{ij'} - d_{i'j'}) < 0$. This shows that, in this case, the error is positively proportional to $|(d_{i'j} - d_{i'j'}) - (d_{ij} - d_{ij'})|$

Case 3: $d_{ij} - d_{i'j} = d_{ij'} - d_{i'j'}$
This case is trivial, since $(d_{i'j} - d_{ij} + d_{ij'} - d_{i'j'}) = 0$ and thus the error is positively proportional to $|(d_{i'j} - d_{i'j'}) - (d_{ij} - d_{ij'})|$ □

Appendix B. Table of instances and experiment details

See Table B.14.

Appendix C. Visualizations of generated instances

See Figs. C.9 and C.10.

Appendix D. Detailed visual overview of results on generated instances

See Figs. D.11 and D.12.

Data availability

Data will be made available on request.

References

Aardal, K., van den Berg, P.L., Gijswijt, D., Li, S., 2015. Approximation algorithms for hard capacitated k-facility location problems. European J. Oper. Res. 242 (2), 358–368.

Akinc, U., Khumawala, B.M., 1977. An efficient branch and bound algorithm for the capacitated warehouse location problem. Manag. Sci. 23 (6), 585–594.

An, H.-C., Singh, M., Svensson, O., 2017. LP-based algorithms for capacitated facility location. SIAM J. Comput. 46 (1), 272–306.

Andersson, G., Francis, R.L., Normark, T., Rayco, M., 1998. Aggregation method experimentation for large-scale network location problems. Locat. Sci. 6 (1–4), 25–39.

Arthur, D., Vassilvitskii, S., et al., 2007. k-means++: The advantages of careful seeding. In: Soda. vol. 7, pp. 1027–1035.

Avella, P., Boccia, M., Salerno, S., Vasilyev, I., 2012. An aggregation heuristic for large scale p-median problem. Comput. Oper. Res. 39 (7), 1625–1632.

Avella, P., Boccia, M., Sforza, A., Vasil'ev, I., 2009. An effective heuristic for large-scale capacitated facility location problems. J. Heuristics 15, 597–615.

Bach, L., 1981. The problem of aggregation and distance for analyses of accessibility and access opportunity in location-allocation models. Environ. Plan. A.

Basu, S., Sharma, M., Ghosh, P.S., 2014. Metaheuristic applications on discrete facility location problems: a survey. OPSEARCH 52 (3), 530–561.

Beasley, J., 1988. An algorithm for solving large capacitated warehouse location problems. European J. Oper. Res. 33 (3), 314–325.

Çalık, H., Labbé, M., Yaman, H., 2019. P-center problems. In: Laporte, G., Nickel, S., da Gama, F.S. (Eds.), Location Science. Springer International Publishing, pp. 51–65.

Cebecauer, M., Buzna, Ł., 2017. A versatile adaptive aggregation framework for spatially large discrete location-allocation problems. Comput. Ind. Eng. 111, 364–380.

Celik Turkoglu, D., Erol Genevois, M., 2020. A comparative survey of service facility location problems. Ann. Oper. Res. 292, 399–468.

Cornuejols, G., Nemhauser, G., Wolsey, L., 1990. The uncapacitated facility location problem. In: Mirchandani, P.B., Francis, R.L. (Eds.), Discrete Location Theory. Wiley, New York, pp. 119–171.

Current, J.R., Schilling, D.A., 1987. Elimination of source A and B errors in p-median location problems. Geogr. Anal. 19 (2), 95–110.

Erkut, E., Bozkaya, B., 1999. Analysis of aggregation errors for the p-median problem. Comput. Oper. Res. 26 (10–11), 1075–1096.

Francis, R.L., Lowe, T.J., 1992. On worst-case aggregation analysis for network location problems. Ann. Oper. Res. 40 (1), 229–246.

Francis, R.L., Lowe, T.J., Rayco, M.B., 1996. Row-column aggregation for rectilinear distance p-median problems. Transp. Sci. 30 (2), 160–174.

Francis, R.L., Lowe, T.J., Rayco, M.B., Tamir, A., 2009. Aggregation error for location models: survey and analysis. Ann. Oper. Res. 167, 171–208.

Francis, R.L., Lowe, T.J., Tamir, A., 2000. Aggregation error bounds for a class of location models. Oper. Res. 48 (2), 294–307.

da Gama, F.S., 2022. Facility location in logistics and transportation: An enduring relationship. Transp. Res. Part E: Logist. Transp. Rev. 166, 102903.

Gnägi, M., Baumann, P., 2021. A matheuristic for large-scale capacitated clustering. Comput. Oper. Res. 132, 105304.

Goodchild, M.F., 1979. The aggregation problem in location-allocation. Geogr. Anal. 11 (3), 240–255.

Görtz, S., Klose, A., 2012. A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. INFORMS J. Comput. 24 (4), 597–610.

Guastaroba, G., Speranza, M.G., 2012. Kernel search for the capacitated facility location problem. J. Heuristics 18 (6), 877–917.

Hillsman, E.L., Rhoda, R., 1978. Errors in measuring distances from populations to service centers. Ann. Reg. Sci. 12 (3), 74–88.

Hodgson, M.J., Hewko, J., 2003. Aggregation and surrogation error in the p-median model. Ann. Oper. Res. 123 (1/4), 53–66.

Hodgson, M.J., Neuman, S., 1993. A GIS approach to eliminating source C aggregation error in P-median models. Locat. Sci. 1 (2), 155–170.

Irawan, C.A., Salhi, S., 2013. Solving large p -median problems by a multistage hybrid approach using demand points aggregation and variable neighbourhood search. J. Global Optim. 63 (3), 537–554.

Irawan, C., Salhi, S., 2015. Aggregation and non aggregation techniques for large facility location problems: A survey. Yugosl. J. Oper. Res. 25 (3), 313–341.

Jain, K., Vazirani, V.V., 2001. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. J. ACM 48 (2), 274–296.

Klose, A., Görtz, S., 2007. A branch-and-price algorithm for the capacitated facility location problem. European J. Oper. Res. 179 (3), 1109–1125.

Lee, J.-B., 2022. Review on environmentally-friendly vehicle charging station location and size problems. KSCE J. Civ. Eng. 26 (9), 3741–3751.

Liao, K., Guo, D., 2008. A clustering-based approach to the capacitated facility location Problem1. Trans. GIS 12 (3), 323–339.

Lorena, L.A.N., Senne, E.L.F., 2003. Local search heuristics for capacitated p-median problems. Netw. Spat. Econ. 3, 407–419.

Marín, A., Pelegrín, B., 1999. Applying Lagrangian relaxation to the resolution of two-stage location problems. Ann. Oper. Res. 86, 179–198.

Marín, A., Pelegrín, M., 2019. P-median problems. In: Laporte, G., Nickel, S., da Gama, F.S. (Eds.), Location Science. Springer International Publishing, pp. 25–50.

Nauss, R.M., 1978. An improved algorithm for the capacitated facility location problem. J. Oper. Res. Soc. 29 (12), 1195–1201.

Papadimitriou, C.H., 1981. Worst-case and probabilistic analysis of a geometric location problem. SIAM J. Comput. 10 (3), 542–557.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Qi, L., Shen, Z.-J.M., 2010. Worst-case analysis of demand point aggregation for the Euclidean p-median problem. European J. Oper. Res. 202 (2), 434–443.

Salhi, S., Irawan, C.A., 2015. A quadtree-based allocation method for a class of large discrete Euclidean location problems. Comput. Oper. Res. 55, 23–35.

Sankaran, J.K., 2007. On solving large instances of the capacitated facility location problem. European J. Oper. Res. 178 (3), 663–676.

Stefanello, F., de Araújo, O.C.B., Müller, F.M., 2014b. Matheuristics for the capacitated p-median problem. Int. Trans. Oper. Res. 22 (1), 149–167.