Article

PC-ILP: A Fast and Intuitive Method to Place Electric Vehicle Charging Stations in Smart Cities

Mehul Bose [1], Bivas Ranjan Dutta [1], Nivedita Shrivastava[2,*] and Smruti R. Sarangi [1]

[1] Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India; mcs212128@cse.iitd.ac.in, mcs212139@cse.iitd.ac.in, srsarangi@cse.iitd.ac.in
[2] Department of Electrical Engineering, Indian Institute of Technology Delhi, India; nivedita.shrivastava@ee.iitd.ac.in
* Correspondence: nivedita.shrivastava@ee.iitd.ac.in

Abstract: The widespread use of electric vehicles necessitates meticulous planning for the placement of charging stations (CS) in already crowded cities, so that they can efficiently meet the charging demand while adhering to various real-world constraints such as the total budget, queuing time, electrical regulations, etc. Many classical and meta-heuristic-based approaches provide good solutions, but they are not intuitive, and they do not scale well for large cities and complex constraints. Many classical solution techniques often require prohibitive amounts of memory and their solutions are not easily explainable. We analyzed the layouts of the 50 most populous cities of the world and observed that any city can be represented as a composition of five basic primitive shapes (stretched to different extents). Based on this insight, we use results from classical topology to design a new charging station placement algorithm. The first step is a topological clustering algorithm to partition a large city into small clusters and then use precomputed solutions for each basic shape to arrive at a solution for each cluster. These cluster-level solutions are very intuitive and explainable. Then next step is to combine the small solutions to arrive at a full solution to the problem. Here, we use a surrogate function and repair-based technique to fix any resultant constraint violations (after all the solutions are combined). The third step is optional where we show that the second step can be extended to incorporate complex constraints and secondary objective functions. Along with creating a full software suite, we perform an extensive evaluation of the top-50 cities and demonstrate that our method is not only 30× faster but its solution quality is also 36.62% better than the gold standard in this area – an integer-linear programming (ILP) approach with a practical timeout limit.

Keywords: Topological data analysis; Persistent homology; Convolutional neural network; Electric vehicle charging station placement

Citation: Title. *Smart Cities* **2023**, *1*, 1–33. https://doi.org/

Received:
Revised:
Accepted:
Published:

1. Introduction

It is widely believed that in the next 10-20 years, the sale of electric vehicles (EVs) will overtake that of petrol and diesel based vehicles. A recent analysis indicates that the number of EVs will increase by a factor of $60 - 70\times$ and will account for 28% of the global fleet by 2040 [1]. As a direct consequence of this, there will be a substantial increase in the need for placing charging points in our already-crowded cities [1].

Efficiently placing charging stations in cities as part of infrastructure planning to make them EV-friendly has been a very active area of research for at least the last 7 years, and as of today, there is a rich body of literature in this area [1–12]. Charging station placement is a specialization of the generic facility location problem that is known to be NP-Hard. Along with bespoke algorithms, there are many specialized approximation algorithms for facility location that are tailored towards charging station placement [13–16]. This area is still far from saturation because existing algorithms are still quite slow and many produce non-intuitive solutions.

Skeptics may argue that town planning and construction are slow activities and take months to years. As a result, charging station placement is not a very time sensitive activity. However, the academic consensus is different because it is often necessary to run these algorithms hundreds of times with different kinds of objective functions, constraints, traffic models and demand maps [17]. With a small change in the city layout, all these simulations need to be run again. Hence, the **need for speed will always be there subject to a minimum bar on the quality of the solution**. Moreover, with the rapid development of fast-charging technologies such as eXtremely Fast Charging (XFC) [4,18–20], this problem has become even more important. The short-term power requirement is very high and can bring down a sub-grid unless the load is appropriately distributed across charging stations (CS) [12,21].

Integer Linear Programming (ILP) [22] is the first approach that one would use to solve such problems given that most variants have linear objective functions and constraints. ILP-based methods are very easy to code, and are often used as the *gold standard* for such problems because they produce optimal solutions. They are, however, very slow and thus seldom considered for practical use[23].

Hence, a rich set of metaheuristics has evolved to solve such problems quickly with an acceptable quality (with respect to ILP). These approaches include methods that use particle swarm optimization [24], genetic algorithms [5], ant colony optimization [15], chicken swarm optimization [14], gray wolf optimization [25] and bee colony optimization [16]. Furthermore, there are approximation algorithms as well as intuitive heuristics that are, in essence, smart greedy algorithms that intelligently combine local and global information [8]. There is an accuracy-performance trade-off, and different algorithms fall at different points. The standard objective for designing new algorithms in this space is to either to add new constraints/objective functions or, given the same quality, minimize the execution time (increase the performance).

Sadly, this vast body of work scores low on intuitiveness. For instance, it is not possible to logically argue with simple arguments as to why a given city layout leads to a better-quality solution with a certain algorithm and another city does not. What features of a city ensure that charging stations can be placed easily and efficiently? If we intend to expand the city, what should the layout of the new satellite township be like? Given that most algorithms are in a certain sense *opaque*, a more intuitive analysis of the solution space is not possible. This sometimes causes some consternation to town planners because they may not want to make all their decisions based on the output of an optimization algorithm, which is a black box. It may lead to extremely unconventional city designs that may inconvenience human beings in many other ways. It is not always possible to capture everything in the objective function and constraints.

Aim of this paper: To summarize, there are three axes that characterize the solution space: quality of the solution, its intuitiveness and the time it takes to compute the solution. We can never compromise on the quality. However, given a quality bar we would like to get a solution that is as intuitive and explainable as possible, and we need to be able to quickly compute it. Additionally, it should be possible to make small iterative changes when the layout of the city slightly changes without recomputing the entire solution. Our approach PC-ILP provides all these features, which we claim is novel (vis-a-vis known related work).

1.1. Salient Features of PC-ILP

We rely on a class of approaches that use a *hierarchical decomposition framework [26–28]*. The idea is to break a large problem into much smaller subproblems, solve them and combine them to form the final solution. Charging station placement problems are natural fits because there is little interaction between regions that are far apart in a city – most of the interaction is local (across adjoining sub-regions). Hence, a lot of constraints can be split (one per sub-region). An astute reader may argue that it may be the case that after combining the smaller solutions, some constraints may get violated when the global solution is created. This is indeed possible, which is why the method of surrogate functions is used [29,30]. Here, we solve a problem that is slightly different from the original, often a simplified version with less constraints. For sub-regions, we solve a surrogate version of the original optimization – this can be done quickly. Then, we combine the solutions. Each surrogate function comes with a repair method, which is a way to degrade the solution such that all the constraints are satisfied. The penalty that is paid is a slightly worsened quality [29].

For, each sub-region, we use a fast ML-based method to compute our solutions. We observe that the layouts of cities across the world are not very different from each other – their basic structures are similar. For example, many American cities are defined by mesh-like arrangements, particularly in the downtown areas of the city [31]. Many European cities are designed like a star – all roads pointing to the center of the city. Many Indian cities are designed as a set of concentric circles. We looked at the top-50 cities by population and found that their sub-regions broadly align with one of a few basic sets of patterns. A city is basically formed using these basic patterns that act as primitives – we shall refer to them as *basic shapes*. For example, New Delhi's main government area looks like a star, the outer region is a set of concentric circles, and the satellite township of Noida looks like a mesh. Our idea is that if we can compute partial solutions for these *basic shapes*, we can combine them to solve the overall problem. This method is intuitive, easy to understand and reason, scales incrementally and is fast.

A brief overview of our approach is as follows. We first divide a large city into regions or a cluster of candidate charging station points (CCSs). The CCSs are uniformly distributed across the city – at the end, the charging stations will be placed in a small subset of these CCSs. Instead of using standard clustering algorithms, we use a topology-based clustering algorithm to divide a big city (or a large number of CCSs) into smaller and well-defined clusters. We found that they provide more meaningful and intuitive results. They perceive the world roughly the same way a human would.

The mapping of these clusters to basic shapes (mesh, rings, star, etc.) is then done using a bespoke *Convolution Neural Network* (CNN) model [32]. To enhance the accuracy of the CNN, we pre-process the CCS data using the *Medial Axis Transform* (MAT) [33] and we use *Persistence-homology Diagrams* (PDs) [34] as features. These clusters act as smaller sub-problems. Identification of these basic shapes is a contribution in itself because it gives us a unique insight into how cities are designed and what are the basic layout elements.

We pre-compute a set of optimal solutions for each basic shape for different runtime conditions. Given that there are a few such basic shapes, a large database can be created depending on our desired accuracy and storage resources. At runtime, the problem reduces to reading the database for each shape and demand distribution and creating a global solution out of optimal local solutions. The end result may require the use of *repair functions* to satisfy all constraints. There is a scope here for making manual adjustments as well. The key point to note is that experimentally we have observed that the quality of our solutions is good, and we are always aware of the way the solution process is proceeding and the expected quality of the solution.

1.2. List of Contributions

Needless to say, it is not possible to compare our work PC-ILP with all the work in this area; we thus compare our solution with some of the latest work in this area that has shown good speedups with respect to prior work. We compare with a fast metaheuristic [5] JAYA and a smart greedy algorithm LGEG [8]. We consider them to be state-of-the-art in this area. Our solution PC-ILP is 3.27× faster than the LGEG approach, and the cost of the solution (defined in Section 3) is 38.87% better. PC-ILP is 200× faster than JAYA while generating a 5.09× better solution.

To summarize, **the main contributions of this paper** are as follows. ❶ A novel clustering technique to divide a city into basic shapes. ❷ A way to classify (identify) the basic shapes by designing a highly accurate deep learning model that takes into account the point cloud diagram and the *persistent homology* diagram. ❸ A novel approach to estimate the similarity between two clusters and then adapt the pre-computed solution of one cluster to the other. ❹ Design and implementation of a novel algorithm to find the optimal placement of CSs for a city and its demand points (DPs) using topological data analysis. A detailed analysis and evaluation of the proposed scheme vis-a-vis state-of-the-art solutions. ❺ A fully-featured tool that is integrated with OpenStreetMaps [35] to interactively conduct all these analyses and run different charging station placement algorithms. ❻ An extensive experimental analysis of the impact of various basic topological shapes on the accuracy and performance of the proposed solution for the top-50 most populated cities.

The rest of the paper is organized as follows. We provide the required background in Section 2. We formulate and describe the problem statement in Section 3. Section 4 presents an analytical and experimental characterization of the parameters and Section 5 describes the proposed scheme. Section 6 presents the results and analysis. Section 7 presents the related work, and we finally conclude in Section 8.

2. Background

In this section, we present the background of some mathematical techniques that we use in the paper and the Simulation of Urban Mobility (SUMO) traffic simulator.

2.1. Mathematical Techniques

2.1.1. Clustering

Clustering is an extremely useful unsupervised machine-learning technique for grouping data based on its characteristics in order to understand its underlying structure. The *K-means* clustering [36] algorithm is one of the most well-known clustering algorithms; it organizes data into κ clusters, where κ is a user-defined variable. It assigns each data point to the closest centroid of a cluster and modifies the centroids based on the mean of the data points.

In *agglomerative* clustering, each point is treated as its own cluster, and clusters are merged iteratively. The number of clusters or a distance threshold can be specified for deciding when the algorithm should terminate. In density-based clustering, data points are clustered according to their spatial density. There are numerous such density-based clustering algorithms such as *DBSCAN (Density-Based Spatial Clustering of Applications with Noise)* [37] and *OPTICS (Ordering Points To Identify Cluster Structure)* [38].

ToMATo (Topological Mode Analysis Tool) [39] is a popular state-of-the-art algorithm that combines the density with persistent homology (which is discussed in more depth in Section 2.1.3). Broadly speaking, the field of topology tries to group all geometrical shapes that have a similar structure, for instance a ring can be stretched and deformed to form a coffee cup – both have a single hole. However, a ring is not the same as a Figure 8, because the latter has two holes.

We present a high-level comparison between different clustering algorithms in Figure 1. Figure 1a shows the data points, while Figures 1b, 1c, and 1d show how the three algorithms perform: *K-means* clustering, *agglomerative* clustering, and *ToMATo*.

2.1.2. *Medial Axis Transform (MAT)*

MAT, often known as skeletonization or topological thinning, is a powerful mathematical technique to extract the central line representation or *skeletal outline* of an object [40] (see Figure 2b). It has numerous applications in object tracking, path planning and image processing such as shape-based matching, shape recognition, feature extraction, and object segmentation. The *medial axis* (skeleton) approximates the object's shape [33] by passing something conceptually similar to a regression line through the points such that we trace out the shape of the object from the points. It is obtained by removing the redundant pixels while preserving the basic connectivity to generate a set of curves and lines outlining the object's shape. In Figure 2b, the circular curve denotes the skeleton of the input image.

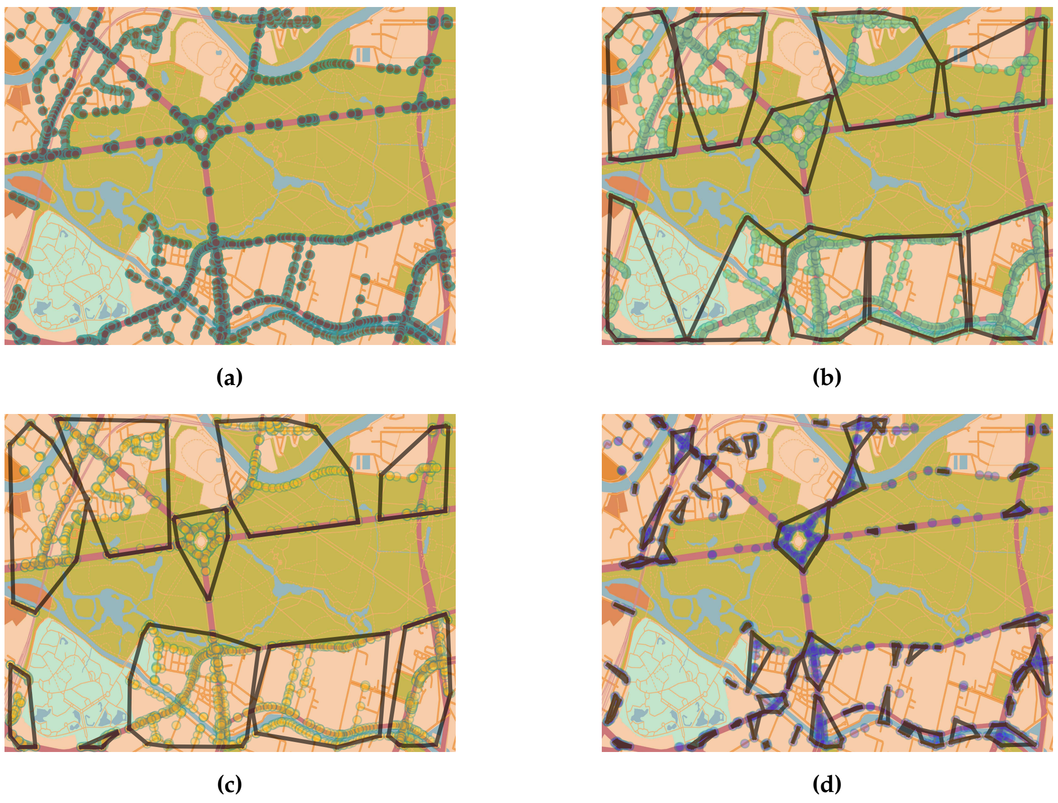


Figure 1. A high-level comparison of different types of clustering algorithms: **(a)** A point cloud representation of the original data; **(b)** K-means clustering with 10 clusters; **(c)** Agglomerative clustering with 10 clusters; **(d)** Topological clustering with a radius of 80 meters [Note: the difference. It captures the key shapes in a more intuitive manner.]. Clusters are formed by combining the density and topological data within the given radius.

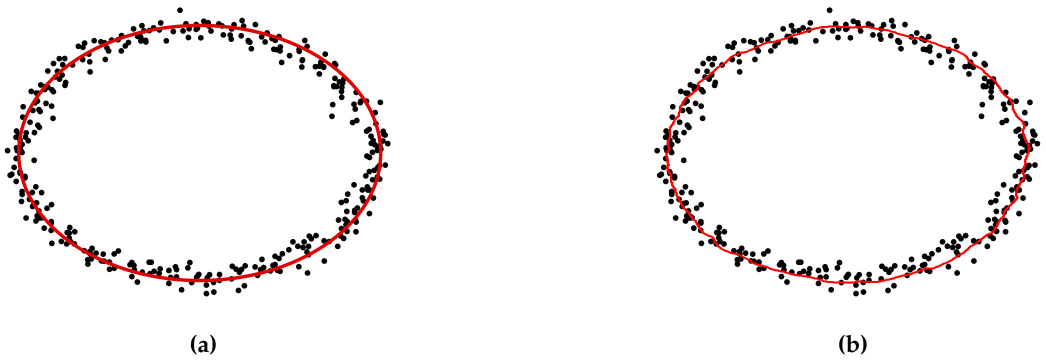
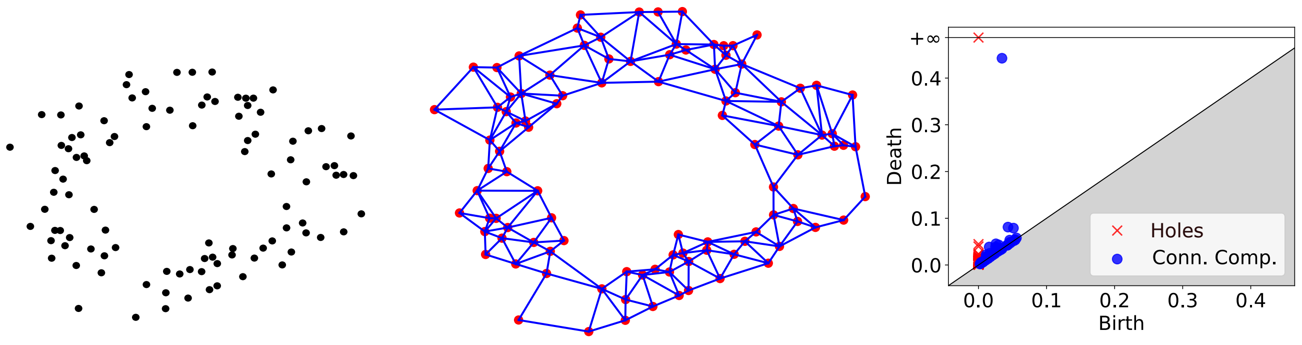


Figure 2. MAT of an image: **(a)** A point cloud representation of points. The elliptical curve is the ideal skeleton that we wish to achieve; **(b)** MAT of the points

2.1.3. *Topological Data Analysis (TDA) and Persistence Homology*

This is a new field with many applications in computational geometry and data analysis. It uses the topology and geometry of the data to obtain information about its structure and perform qualitative and quantitative analyses [41]. TDA captures complex topological structures within data that are represented as a point cloud. A *point cloud* refers to a collection of distinct data points distributed in an n-dimensional space.

The concept of a *Persistent homology* involves measuring topological features at multiple spatial scales. This is achieved by studying the evolution of different topological features such as the number of holes in the object representation (topological space) of the point cloud. The *Rips complex* [42] and *Cech complex* [43] are two such well-known topological



(a) **(b)** **(c)**

Figure 3. Steps to generate the PD diagram of a point cloud: **(a)** A point cloud representation of points; **(b)** Rips complex of the points; **(c)** PD of the points. The red points (crosses) denote the holes, whereas the blue points (filled dots) denote the connected components

space construction methods. Both these approaches consider neighborhoods of a given radius around points and merge all the points within the neighborhood.

The *Persistence Diagram* (PD) provides a way to study a topological space by depicting the birth and death of topological features, such as connected components or holes, as we increase the radius. It provides a nice graphical view of the topological structure of a dataset (represented as an n-dimensional point cloud). It is used for various purposes such as feature selection, pattern recognition, and shape analysis [34]. It maintains a simplex tree, which is a flexible and efficient data structure to represent and store filtered data. Figure 3 demonstrates the creation of a PD for a given cluster using the *Rips complex* as the topological space creation method [44]. Figure 3a is the point cloud representation of the cluster, Figure 3b represents the Rips complex of the points in the cluster, and Figure 3c is the final PD of the cluster. The x and y axes represent time units, where it is assumed that the radius is increased linearly with time. A point at (x_1, y_1) means that a specific feature (hole or connected component) was visible for the first time (born) at time x_1 and stopped being visible at time y_1 (died).

2.2. *Simulation of Urban Mobility (SUMO): Traffic Simulator*

SUMO is an open-source, microscopic traffic simulation software widely used to generate traffic distributions in an urban area [45]. It can simulate various transport-related entities such as public transport, vehicles, and traffic control mechanisms. We can model traffic, build networks, control traffic, conduct environmental analyses, and visualize real-time simulated traffic data. It is widely used in academia and industry. It can easily be integrated with other tools for various applications. Figure 4 represents the traffic simulated on the map of a region in Berlin, Germany. The points shown in this figure depict the road network of the city.

2.3. *JAYA algorithm*

The JAYA algorithm is a recent population-based metaheuristic algorithm proposed by Rao et al. [46]. The algorithm combines the characteristics of evolutionary algorithms and swarm intelligence. It is inspired by the *survival of the fittest* principle of natural selection. The algorithm uses very few hyperparameters. Hence, it does not require extensive tuning.

3. Problem Formulation

We shall describe a generic charging station placement problem [2–5,8] in this section. A few definitions follow. Candidate charging stations (CCSs) are all the potential locations for deploying CSs on a given city map. We define a DP (demand point) as a location where EV charging is in significant demand. Let us define $\mathcal{S}_{DP}$ to be the set of all demand points, $\mathcal{S}_{CS}$ to be the set of all charging stations, and $\mathcal{S}_{CCS}$ to be the set of all candidate charging

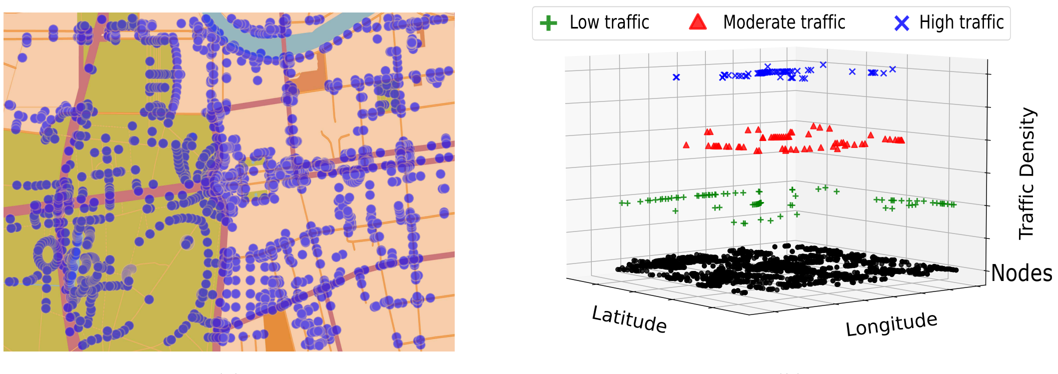


Figure 4. Simulated traffic of a region in Berlin, Germany using SUMO. The black circles indicate the points on the road. **(a)** Points on the road network in 2D; **(b)** Simulated traffic density for the points in 3D

stations. Clearly, $\mathcal{S}_{CS} \subseteq \mathcal{S}_{CCS}$. $N_{dp} = |\mathcal{S}_{DP}|$ is the total number of DPs, $N_{cs} = |\mathcal{S}_{CS}|$ is the total number of charging stations and $N_{ccs} = |\mathcal{S}_{CCS}|$ is the total number of candidate charging stations.

We can now define a shortest Euclidean distance matrix $\mathbf{D}$ of dimension $N_{dp} \times N_{ccs}$, where, $\mathbf{D}[i][j]$ denotes the shortest Euclidean distance between the i^{th} DP and the j^{th} CCS. Let the maximum distance that an EV user can travel to reach a charging station be τ. This is known as the *reachability distance*. As a result, we must ensure that after traveling a distance of τ, the EV user definitely has access to a CS.

The total number of CSs that can be deployed in a city is subject to a budget. Let β represent the maximum number of CSs that can be deployed in a city. Hence, the actual number of deployed CSs (N_{cs}) should always be less than the maximum budget (β) for CSs i.e. $N_{cs} \leq \beta$. Let $\mathbf{S}$ be the supply matrix of dimension $N_{dp} \times N_{ccs}$, which represents the allocation (mapping) of a CS to a DP. $\mathbf{S}[i][j] = 1$ implies that the j^{th} CCS is allocated to the i^{th} DP, otherwise $\mathbf{S}[i][j]$ is 0. Let $\mathcal{O}$ be a binary array with N_{ccs} elements which indicates whether a CCS is finally chosen to be a CS or not. $\mathcal{O}[i] = 1$ implies that the i^{th} CCS is considered as a CS, otherwise it is not considered to be a CS.

The aim is to ensure an optimal placement of the CSs across the city. To solve this problem, we minimize the **cost**, which can be represented as an objective function $\sum_{i=1}^{N_{dp}} \sum_{j=1}^{N_{ccs}} \mathbf{D}[i][j] \times \mathbf{S}[i][j]$. This is the sum of the total distance that all EV users have to travel (assuming there are the same number of users at each demand point). The overall mathematical formulation of the problem is shown below.

$$\textit{Minimize} \sum_{i=1}^{N_{dp}} \sum_{j=1}^{N_{ccs}} \mathbf{D}[i][j] \times \mathbf{S}[i][j] \quad \triangleright \text{Overall distance} \tag{1a}$$

Subject to:

$$\sum_{j=1}^{N_{ccs}} \mathbf{D}[i][j] \times \mathbf{S}[i][j] \leq \tau, \forall i \in \{1 \dots N_{dp}\} \quad \triangleright \text{Reachability} \tag{1b}$$

$$\sum_{j=1}^{N_{ccs}} \mathbf{S}[i][j] = 1, \forall i \in \{1 \dots N_{dp}\} \quad \triangleright \text{One CS connected to one DP} \tag{1c}$$

$$\mathbf{S}[i][j] \leq \mathcal{O}[j], \forall i \in \{1 \dots N_{dp}\}, j \in \{1 \dots N_{ccs}\} \quad \triangleright \text{Associate only if the CCS is a CS} \tag{1d}$$

$$\sum_{i=1}^{N_{ccs}} \mathcal{O}[i] \leq \beta \quad \triangleright \text{Budget condition} \tag{1e}$$

Constraint 1b ensures that all the DPs must be at most τ units of distance away from a placed CS. Constraint 1c states that $\forall DP$, a CS must be allocated to it. This constraint is very helpful in distributing the demand of DPs efficiently. Constraint 1d ensures that the allocated CS must be chosen as a CS (sanity check). Finally, Constraint 1e states that the number of CSs must be less than or equal to the budget β. This model is similar to the one proposed by Kulkarni et al.[47] and is similar to the models in many more references[1,7,9–12].

Our primary constraints helped us to identify the exact location to place the CSs. But, just placing the CSs does not solve our woes. We need to take care of other electrical constraints such as Constraint 2 (described in Section 3.1), and we need to be sure that the areas with high traffic (most visited roads) are handled well. We establish that if a CS is located in an area with high traffic density, it would require additional chargers to serve the customers. Basically, a charger is a power supply device that supplies power for recharging an EV. A CS can have multiple chargers. These chargers reflects the size and capacity of a CS.

In order to simplify the underlying problem and enhance the performance of the method, we consider all these constraints as additional constraints.

It is a standard practice to divide the problem into two parts: a solution that satisfies the primary constraints and then modifications to the solution based on additional constraints: traffic conditions, capacities of charging stations (issue of inductive and capacitive loads), electrical regulations and importance of the area as additional constraints. The additional constraints mostly affect the internal working of the charging station [48,49].

3.1. Additional Constraints

Let us incorporate additional information about the traffic, electrical load, and queuing time in the model. The overall electrical load is dependent upon multiple aspects, including but not restricted to the charging pattern (fast charging), the popularity of the charging stations (public charging stations, workplace charging stations), the power efficiency of the chargers, and the charging capacity of the electric vehicles. EV chargers are high frequency electronics converters that transform the AC supply into a DC supply to charge an EV. These convertors impose a load of nonlinear nature that has an adverse impact on the performance of the power grid as they introduce harmonics into the system [50]. Additionally, the process of charging electric vehicles (EVs) results in a quick and impulsive increase in the load on the charging infrastructure, hence causing voltage instability issues. Thus, balancing the load among the CSs is very crucial.

To achieve this, first, we need to define the number of chargers in a CS. So, let the matrix $\mathbf{K}$ of size $N_{cs} \times N_a$ represent the allocation of a charger to a CS, where N_a is the total number of chargers. $\mathbf{K}[i][j] = 1$ means that the j^{th} charger is allocated to the i^{th} CS, otherwise $\mathbf{K}[i][j]$ is 0. Using $\mathbf{K}$, we define $a_i = \sum_{j=1}^{N_a} \mathbf{K}[i][j]$ as the number of chargers at the i^{th} charging station.

Let us add some *electrical constraints* to the model [12]. In the real world, CSs will present themselves as large electrical loads. By adding these constraints, we can distribute the electrical load more efficiently across phases and across CSs. We define an array L of size N_{cs}, where $L[i]$ denotes the total load placed on the i^{th} CS. It is a deciding factor in estimating the largest amount of load that can be placed on a CS such that it still maintains harmonic in-line currents, phase balance, and voltage deviations (within the limit). Additionally, each CS should contain at least one charger (sanity check). The following equation (Constraint 2) highlights that the chargers in a CS should not be overloaded, and still contain at least one charger.

$$1 \le a_i \le \frac{L[i]}{\rho_{ev}} \quad \triangleright \text{Chargers in a CS must not be overloaded} \tag{2}$$

Here, ρ_{ev} denotes the charging power rating of an EV. Constraint 2 is used to tackle the load imbalance problem across CSs as the maximum load of a CS must be less than the overloading factor L [21]. The chargers are also alloted a budget (β_a). Constraint 3 highlights the fact that the total number of chargers in a region should not exceed the budget for that region (β_a). This will ensure that there are no local hotspots and the power requirement of a CS never exceeds the substation's capacity and power quality is maintained.

$$\sum_{i=1}^{N_{cs}} a_i \leq \beta_a \quad \triangleright \text{ Total budget of chargers} \tag{3}$$

Next, we define the *queuing time* [51,52] as the time spent waiting in a queue to charge the EV. This depends on several factors such as the traffic at a CS, the charging time for each EV and the number of chargers available at a CS [51]. We use a queuing model to model the traffic and the queuing time across the city; this is a popular model that has been used in a lot of prior work. It was proposed by Zhu et al [51].

The queuing time at a CS increases proportionally to an increase in demand and traffic, whereas an increase in the number of chargers at the said station will reduce the queuing time as presented in Equation 4a. This model is similar to the one proposed by Zhu et al [51].

$$Q_i = \frac{a_i(\eta_i a_i)^{a_i+1} P_{0i}}{\lambda_i a_i!(a_i - a_i\eta_i)^2} \quad \triangleright \text{ Queuing time at the } i^{th} \text{ CS} \tag{4a}$$

$$\lambda_i = \frac{\sum_{j=1}^{x} \mathcal{X}_i[j]}{t_c} \quad \triangleright \text{ Traffic flow rate around the } i^{th} \text{ CS} \tag{4b}$$

$$P_{0i} = \left[\left(\sum_{j=0}^{a_i-1} \frac{(\eta_i a_i)^j}{j!} \right) + \frac{(\eta_i a_i)^{a_i}(a_i)}{a_i!(a_i - a_i\eta_i)} \right]^{-1} \quad \triangleright \text{ Probability of finding an idle charger at the } i^{th} \text{ CS} \tag{4c}$$

$$\eta_i = \frac{\lambda_i}{\mu a_i} \quad \triangleright \text{ Chargers' utilization at the } i^{th} \text{ CS} \tag{4d}$$

$$\mu = \frac{1}{t_s} \triangleright \text{ Service time of a charger} \tag{4e}$$

Here, λ_i denotes the EV traffic at the i^{th} CS (Equation 4b) and t_c denotes the time duration during which the traffic was monitored. We estimate the traffic at a CS by simulating the traffic flow using the SUMO traffic simulator. Here, we make the reasonable assumption that the traffic in a CS is dependent on the EV traffic within the reachability distance. λ_i is calculated by adding all the EV traffic of the nodes on the road that are τ-reachable (within τ units of distance) from that CS. Let $\mathcal{X}_i$ be an array of size x, where, x is the number of nodes that are τ-reachable from the i^{th} CS and $\mathcal{X}_i[j]$ denotes the traffic between the i^{th} CS and the j^{th} connected node.

Next, P_{0i} defines the probability of finding an idle (empty) charger upon arrival at the i^{th} CS, and is defined in Equation 4c, where η_i represents the utilization factor of a CS, which measures the degree of utilization of the chargers (Equation 4d). Here, μ represents the service time of a charger, which is inversely propontial to the charging time of a single charger t_s (Equation 4e).

The aim is to reduce the queuing time and minimize the total number of chargers used. Keeping this in mind, additional objective functions can be defined as follows:

$$\begin{aligned} Minimize \sum_{i=1}^{N_{cs}} Q_i \quad & \triangleright \text{ Additional objective function 1} \\ Minimize \sum_{i=1}^{N_{cs}} a_i \quad & \triangleright \text{ Additional objective function 2} \end{aligned} \tag{5}$$

We can have multiple objective functions in our formulation. From a single-objective optimization problem, it will become a multi-objective optimization problem. A Pareto optimal point can be chosen subject to some overall desirability function, as is the standard practice. Now, that the problem has been defined, we need to find a method to effectively solve it. For doing this, we need to understand the features present in modern cities and show that their layouts are quite similar – in a topological sense. This means that they consist of a set of basic primitives, where each primitive can be arbitrarily stretched and deformed (in certain ways). This is the objective of the next section.

4. Characterization

A collection of selected nodes (CCSs) represents a city's road network. In order to identify the topological shapes present in a city, the first aim is to select the best clustering algorithm to identify and isolate constituent topological clusters (of CCSs) from a city's road network.

4.1. Clustering Algorithm - Identification of the Clusters in a City

We perfrom an extensive analysis of several state-of-the-art clustering algorithms namely *K-means*, *Agglomerative*, and *ToMATo* to determine which algorithm can successfully isolate the topological clusters of CCSs present in a city. The details of the clustering algorithms are presented in Section 2.1.1. The analysis is performed on a section of Berlin, Germany using the system configuration mentioned in Table 1.

Hardware Settings	
Chip: Apple M1	CPU cores: 8
GPU: Apple M1 8-core GPU	DRAM: 8GB
Software Settings	
Operating System: MacOS Monterrey 12.6	Python Version: 3.7
TensorFlow Version: 2.11.0	Tkinter Version: 8.6.12
Gudhi Version: 3.8.0	CVXPY Version: 1.3.1

Table 1. System configuration

We improve the computation speed of the clustering algorithms by performing random point reduction [53–55] (random sampling) to reduce the number of points. This is done for all the algorithms. We empirically determined that it is possible to decrease the points by up to 75% without causing much distortion to the topological shapes.

Let us consider a representative example. Figure 1a shows the input points for the clustering algorithm after performing point reduction on a section of the Berlin city. Figure 1b shows the outcome of *K-means clustering* with 10 clusters ($k = 10$). Due to the fact that *K-means* assumes that each cluster has the same size and is susceptible to noise, it is quite incapable of isolating the topological shapes.

Figure 1c shows the outcome of *agglomerative clustering* with 10 clusters. We observe that the clusters that are generated by the algorithm are relatively similar to the ones generated by *K-means clustering*. They also fail to isolate the topological shapes. Figure 1d shows the outcome of *ToMATo* with radius $r = 80$. The algorithm generates clusters by connecting all the nodes within a radius of 80 meters. It can isolate the majority of the topological shapes as it considers both the highly dense and less dense areas. We conclude that **the clusters generated by ToMATo can successfully isolate a majority of the shapes**. In our experiments, this observation is found to hold across all big cities (our dataset). Therefore, this becomes the most desirable clustering algorithm for identifying the constituent topological shapes of a city.

Next, we take a look at the proposed scheme and how it uses the clustering algorithm.

5. Material and Methods

5.1. Overview of the Scheme

Problem: 1 The primary objective of this work is to identify the locations of CSs based on a set of primary constraints (budget, reachability radius, demand points). Similar to prior work [56–58], we address an (**Problem: 2**) additional objective of distributing the chargers among the CSs such that no additional constraints (traffic, electrical regulations, queuing time) are violated.

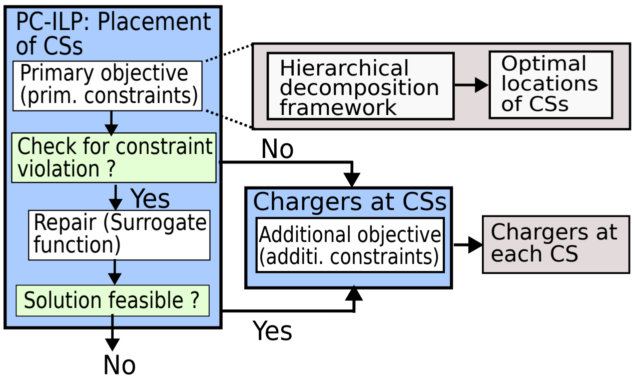

PC-ILP: Placement of CSs
Primary objective (prim. constraints)
Check for constraint violation ?
Yes
Repair (Surrogate function)
Solution feasible ?
No
Hierarchical decomposition framework
Optimal locations of CSs
No
Chargers at CSs
Additional objective (additi. constraints)
Chargers at each CS
Yes

Figure 5. A high-level representation of the proposed scheme.

Problem: 1 Figure 5 shows that the primary objective is acheived using the proposed PC-ILP algorithm by using a hierarchical decomposition method. In this method, the city map (represented as a set of CCSs) is decomposed into clusters of CCSs using a topological clustering algorithm. Next, a CNN is used to identify the geometric shape of each cluster.

We study topological shapes (clusters of CCSs) present in 50 of the largest cities in the world and based on our findings, we argue that a few basic shapes can be used to capture the topological structure of most cities. This conclusion is supported by the fact that a small number of 2-D simplexes (a line, a circle, a mesh, a star and a concentric circle) can be used to define any topological shape of interest in a modern city [59]. We present a pictorial representation of all the 2-D simplexes in Figure 6. Our experiments support this assumption.

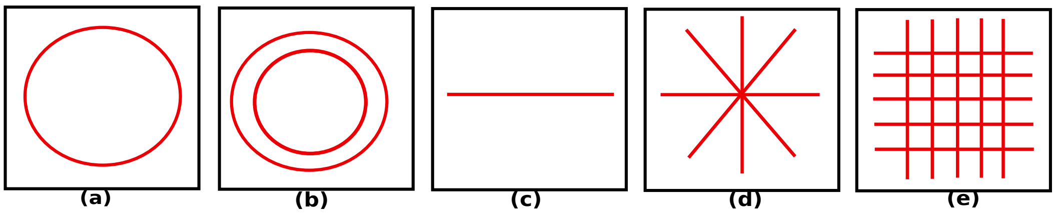


Figure 6. Representation of the five 2-D simplexes. **(a)** Circle; **(b)** Concentric circle; **(c)** Line; **(d)** Star; **(e)** Mesh

These clusters are seen as small sub-problems and we use a database of precomputed solutions to solve these sub-problems. The database comprises optimal solutions for various shapes (clusters of CCSs) over a spectrum of parameters including the number of DPs, budget, and reachability distance as shown in Table 2. During runtime, an user only needs to look at the database to get a solution for each shape. However, the final solution that has been obtained from the database may not be feasible. Such infeasible solutions are fixed using a surrogate function (details to follow).

Parameter	Description
$\mathcal{S}_{CCS}$	The CCSs depicts the potential charging stations across the city.
τ	The reachability distance is the maximum distance an EV user needs to travel from a DP.
β	The budget represents the maximum possible CSs in a city.
$\mathcal{S}_{DP}$	The demand points correspond to the points in the city that demand EV charging.

Table 2. Primary parameters of the problem

Problem: 2 Then, the repaired/feasible solution from PC-ILP is sent to the second stage – a small *ILP* optimization problem. This considers a few of the additional parameters (traffic, queuing time, and electrical regulations). The reason for this is that the traffic patterns within a city can exhibit severe variations. Consequently, attempting to create a database capable of accommodating all of these combinations would not be feasible. The queuing time also depends on the traffic in a CS, so it is also considered as an additional parameter. Additionally, the electrical regulations also directly influence the number of chargers in a given CS. Therefore, all of these factors are categorized as additional parameters, which have a direct impact on the number of chargers in each CS. This ILP stage (which is a much smaller problem) finds the optimal number of chargers that need to be placed at each CS. This stage is needed for flexibility. Modern EV placement problems can have a lot of constraints and objective functions (soft and hard). Hence, a two-stage approach suits us well.

5.2. *Placement of Charging Stations (Primary Objective)*

A high-level diagram of the proposed scheme is shown in Figure 7. The algorithm is divided into six major components: ❶ Precomputation of the ILP solutions for various basic shapes. These solutions are stored in a database. ❷ Identification of the potential charging point locations (CCSs) in a city. ❸ Partitioning of the city map into distinct clusters of CCSs using *ToMATo*. ❹ Basic shape identification of the clusters using a CNN. ❺ Finding the most appropriate match in the database for the identified shape. ❻ Estimation of the final solution based on the matching solution from the database. Algorithm 1 describes our proposed scheme. We also show a glossary of the symbols in Table 3.

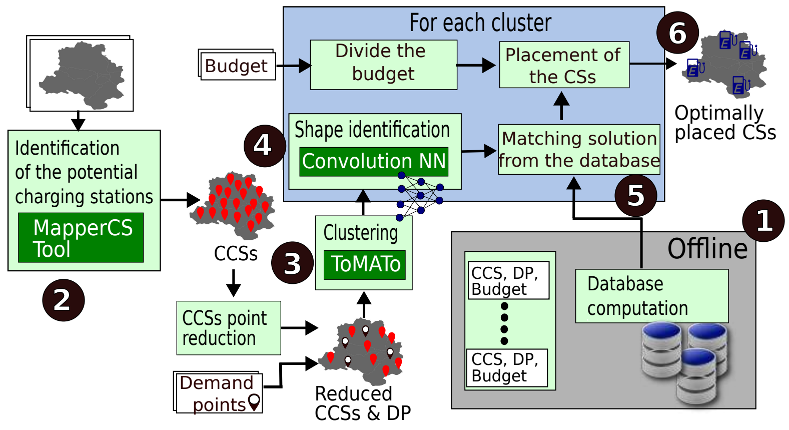


Figure 7. A high-level diagram of the proposed scheme

5.2.1. Create Database of Pre-Computed Solutions (Offline)

The first stage is creating an exhaustive database containing solutions to the CS placement problem for a set of basic shapes. Each basic shape is represented by a set of CCSs.

▶ **Normalization of the Latitude and Longitude** Each of the CCS nodes in a shape is denoted by a pair of latitude and longitude coordinates. Now, across cities, the constituent topological shapes may remain the same, but their sizes may vary significantly. Since it is not possible to store the results for all potential basic shape sizes in a database, we normalize the values of longitudes and the latitudes for different shapes. Normalizing the geographical coordinates in a large urban city to store a scaled version of the geographical information is a well known concept that is used in urban city planning [60–65]. It helps to simplify and generalize the geographic data (nodes), making it more manageable.

To acheive this, we normalize the location of each node, i.e., the latitude and longitude values such that they lie in the range of $[0, 1]$. This is done for a combined set of DPs and CCSs ($\mathcal{S}_{CCS} \cup \mathcal{S}_{DP}$) as normalization of each set individually would distort the locations of the DPs.

We empirically estimated the size of the shapes in 50 of the largest cities in the world and observed that the maximum area of a shape among all the five basic shapes is 8.5 km^2, while the minimum possible area is 1 m^2. Our resolution is quite fine, even though such a granularity is almost never required. Most of the time, we can cover a roughly $3km \times 3km$ section of the city in a single well-defined basic shape.

► **Patterns of DP Distribution in a Shape** DPs denote the points at which a demand for charging exists. We place DPs manually based on the density of CCSs, presence of public amenities like malls and hospitals and downtown areas. To characterize the location of DPs, we subdivide all the normalized shapes into *zones*. If a DP falls in a zone, we assume that it is in the centroid of the zone (refer to Figure 8). It means that there is some feature of interest in the city and all the roads in the vicinity lead to it.

We ensure that our database captures all the possible locations of the DPs in a city. Let us assume that the maximum possible number of DPs associated with a shape is d and the total number of zones in a shape is z, then the total number of possible DP patterns that should be present in the database is $\sum_{i=0}^{d} {}^{z}C_i$. ${}^{z}C_i$ is the number of ways in which we can choose i elements out of z elements (number of combinations).

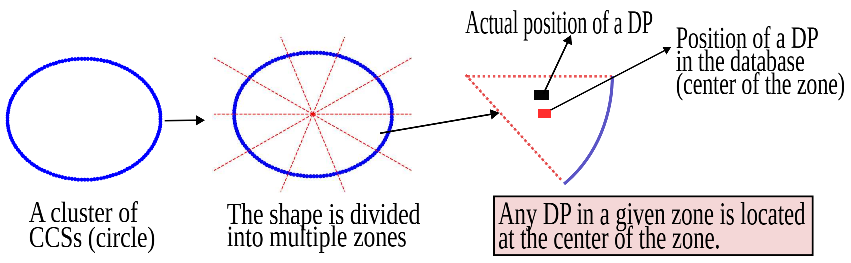

Actual position of a DP
Position of a DP in the database (center of the zone)
A cluster of CCSs (circle)
The shape is divided into multiple zones
Any DP in a given zone is located at the center of the zone.

Figure 8. Creation of zones for a cluster

After successfully estimating all the possible DP distribution patterns for a given shape, we apply ILP to solve the CS placement problem for the shape and the corresponding DP distribution patterns, given a budget and a reachability radius. This generates possible sets of locations for the CSs. This procedure is conducted offline. We combine the locations of the CSs with the shape (clsuter of CCSs) and the parameters (DP distribution pattern, budget, reachability radius) into one single database entry. In addition, we also compute ILP solutions for a given shape and DP distribution pattern for a range of budget and reachability radii (the range is estimated empirically) and add all the computed solutions to the database.

After creating the database, we estimate the solution for a new input map using the precomputed database. The next series of steps are all performed online.

5.2.2. Locating Potential Charging Stations in the Input Map

The first task is to identify the prospective CCSs in the input map. To identify prospective CCSs in the cities, we identify the road network of the city and place CCSs along it. We developed our in-house interactive tool *MapperCS* that enables the filtering of the street data from a specific city map and also identifies the potential CCSs (see Section 6).

Symbol	Definition	Symbol	Definition
N_{ccs}	Number of CCSs	$\mathbf{D}$	The distance matrix between DPs and CCSs
N_{dp}	Number of DPs	$\mathcal{M}$	Mapping between cluster and DPs
τ	Reachability distance	β	Budget (max allowed CSs)
N_{cs}	Total number of CSs	$\mathbf{S}$	The supply matrix (DP-CS allocation)
$\mathcal{O}[N_{ccs}]$	Boolean array that indicates whether a CCS is a CS	$\mathcal{P}$	A set of basic shapes and DP distribution
$\mathcal{C}$	A set of clusters	$\mathcal{DB}$	Pre-computed database
S	The shape of a cluster	$SimS$	A similar shape retrieved from the database
$\lvert X\rvert$	Size of the set X	$\{x, y\}$	Concatenate x and y

Table 3. A glossary of the symbols

Algorithm 1: PC-ILP (Online stage)

```
1  Input: Candidate charging stations $CCS = (CCS_1, CCS_2...CCS_{N_{ccs}})$ ; Demand
    points $DP = (DP_1, DP_2...DP_{N_{dp}})$; Budget $\beta$; Reachability distance $\tau$;
    Precomputed Database $\mathcal{DB}$; Threshold $T$
2  Output: Charging stations $CS = (CS_1, CS_2..CS_{N_{CS}})$
3  \* Reduce the size of each $CCS$ by a factor of x *\
4  $CCS_{new} = \textbf{ReducePoints}(CCS, N_{ccs}/x)$
5  \* Identify the clusters in CCSs *\
6  $\mathcal{C} = ToMATo(CCS_{new})$
7  \* Assign each DP to its nearest cluster *\
8  $\mathcal{M} = \textbf{Assign}(\mathcal{C}, DP)$
9  Sort $\mathcal{C}$ in decreasing order of $|\mathcal{C}|$
10 $i \leftarrow 1$
11 while $i \leq |\mathcal{C}|$ do
12     \* Divide the budget among the clusters *\
13     $b = \left\lfloor \frac{\beta}{|\mathcal{C}|} \right\rfloor$
14     if $i \leq (\beta \bmod |\mathcal{C}|)$ then
15         $b \leftarrow b + 1$
16     end
17     \* Identify the shape of the cluster $\mathcal{C}_i$*\
18     $S = \textbf{IDShape}(\mathcal{C}_i)$
19     \* Find the most similar shape in the database for the cluster $\mathcal{C}_i$*\
20     $SimS = \textbf{TraverseDB}(S, \mathcal{C}_i, \mathcal{M}, \mathcal{DB}, b)$
21     \* Apply the solution of $SimS$ to cluster $\mathcal{C}_i$*\
22     $CS_{\mathcal{C}_i} = \textbf{ApplySolution}(SimS, \mathcal{C}_i, \mathcal{M}, T)$
23     $CS.append(CS_{\mathcal{C}_i})$
24     $i \leftarrow i + 1$
25 end
26 return $CS$
```

5.2.3. Clustering Algorithm

Next, we perform clustering on the CCS set to isolate the basic clusters using a topological clustering algorithm, *ToMATo*. It captures all the topological shapes present in a city. The computation speed of the clustering algorithm is improved by randomly reducing the number of points [53–55] in the CCSs using the **ReducePoints** function.

► **Mapping DPs to clusters -** In the next step, we map each DP (represented as DP_i) to its nearest cluster $\mathcal{C}_i \in \mathcal{C}$. A single cluster can be mapped to many DPs, creating a many-to-one mapping. To map DPs to a cluster, we first compute the convex hull for each cluster using the *QuickHull* algorithm [66]. We categorize the DPs into two categories based on their position relative to the convex hull ① The DPs that are contained within the convex hull of a cluster $\mathcal{C}_i$ are automatically mapped to $\mathcal{C}_i$. ② The DPs that fall outside the convex hull of all clusters are mapped to the closest cluster $\mathcal{C}_i$ using the **Assign** function as described in Algorithm 2. Basically, the clusters are characterized by their centroids. We find the cluster closest to a DP by estimating the distance between the centroid of the cluster and the DP. We map the DP to its nearest cluster based on the minimum distance to each centroid [67].

Algorithm 2: Function $Assign(\mathcal{C}, DP)$

```
1  Input: Demand points $DP = (DP_1, DP_2...DP_{N_{dp}})$, Clusters $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2...\mathcal{C}_t)$
2  Output: Mapping $\mathcal{M}$
3  for $i = 0$ to $N_{dp}$ do
4      \* Identify the closest cluster from the DP ($DP_i$).*\
5      $old = \infty$
6      $clust = \mathcal{C}_1$
7      for $j = 1$ to $t$ do
8          $new = dist(DP_i, centroid[\mathcal{C}_j])$
9          if $new < old$ then
10             $clust = \mathcal{C}_j$
11         end
12     end
13     $\mathcal{M}[clust]$.append($DP_i$)
14 end
15 return $\mathcal{M}$
```

Once we find the closest cluster ($\mathcal{C}_j$) for a DP (DP_i), we add the DP (DP_i) to $\mathcal{M}[\mathcal{C}_j]$ where $\mathcal{M}$ is the *Demand point-Cluster Mapping (DCM)*, which stores the details of the mapping between DPs and clusters. This process is repeated until we find a mapping for every DP. After the mapping procedure is complete, we divide the total budget in proportion to the number of CCSs covered by each cluster. To acheive this, clusters ($\mathcal{C}$) are sorted in decreasing order of the cluster size. The clusters with more CCSs are allotted a greater portion of the budget (β) as described. Subsequently, we iterate over the sorted clusters $\mathcal{C}$ and process each cluster $\mathcal{C}_i$ individually. The next task is to identify the shape of the cluster $\mathcal{C}_i$.

5.2.4. Shape Identification using a Convolution Neural Network (CNN)

To determine the shape of a given cluster, we use a CNN model. We studied the constituent shapes of the major urban agglomerations of the world to create a realistic dataset for training and testing the CNN model. After extensive experimentation, we designed a novel CNN architecture capable of identifying shapes. The architecture is presented in Figure 9.

The input of the model is the point cloud and the PD of its MAT. The PD of the MAT of the point cloud is generated by removing holes and connected components, which are less persistent by introducing a threshold as illustrated in Figure 10. Figure 10a is a typical PD and Figure 10b is the PD of the MAT (modified PD) after thresholding. Now, let us take a deeper look at the architecture of the model.

► **Model Architecture** The proposed model architecture comprises two parallel CNN layers as depicted in Figure 9. Both layers consist of an identical three-layer CNN architecture. In the three sequential convolution layers, the number of filters increases in the sequence of (32, 64, 128). Each convolution layer is followed by a *LeakyReLu* activation function and a *MaxPooling* layer with a pool factor of 2×2. Finally, we employ a regularization technique

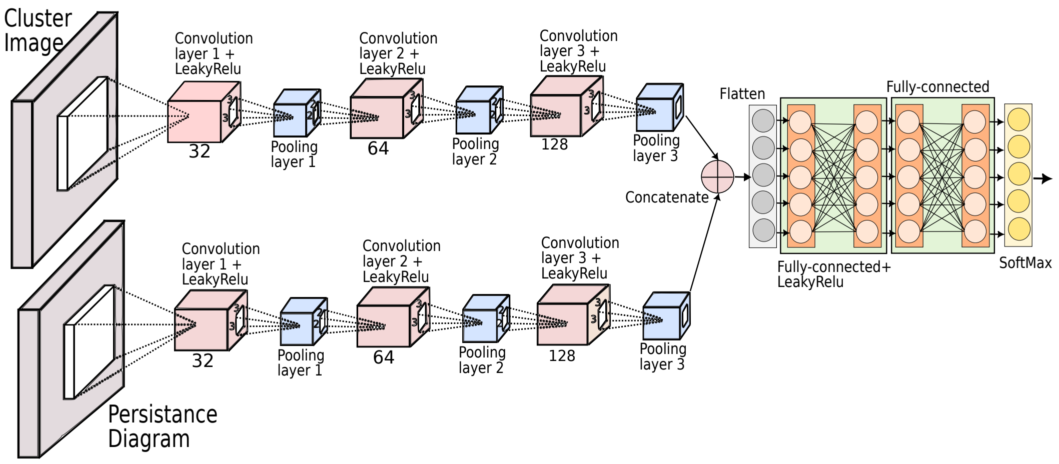
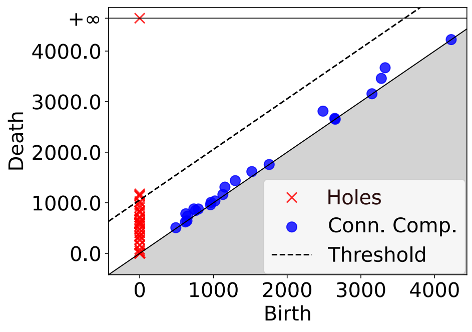
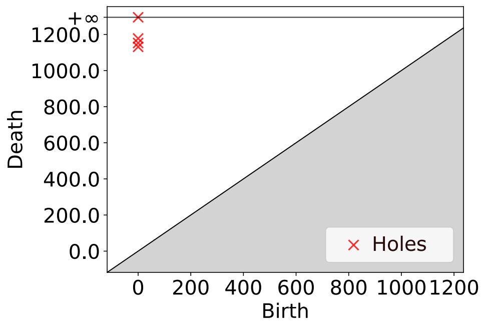


Figure 9. Proposed CNN architecture



(a)



(b)

Figure 10. Model Input - A PD diagram with threshold and point reduction. The red crosses denote the holes present in the point cloud, and the blue circles denote the connected components present in the point cloud: **(a)** A typical PD diagram ; **(b)** The PD of the MAT. All points below the threshold are removed

called *Dropout*, which involves randomly deactivating the neurons within a layer. The respective dropout percentages for the three convolution layers are 25%, 25% and 40%.

The outputs of the two parallel convolution structures are combined using a *Concatenate* layer. Thereafter, the result of this layer is changed from a 2D matrix to a 1D vector using a *Flatten* layer. This conversion enables the output to be processed by the fully connected layers (*Dense* layer).

The *Dense* layer consists of 512 neurons and a *LeakyReLU* activation function, which is followed by another *Dropout* layer that is configured to exclude 50% of the neurons. The subsequent fully connected layer has five neurons for the five basic shapes. In the end, we use the *Softmax* activation function to predict the class of the cluster. The above-described model can identify an input shape quite accurately.

Now that we have successfully identified the shape of a cluster (**IDShape** function), we must query our precomputed database to discover the closest matching solution for a new input cluster.

5.2.5. Retreival of the Pre-computed Solution from the Database

The next step of PC-ILP is to identify the closest solution for a new cluster from a database of precomputed solutions using the **TraverseDB** function. We need to first normalize the input cluster to set it in the same scale as the clusters in the database (refer to Section 5.2.1). For the solutions of any two clusters to be broadly similar, they must be ① of the same shape and ② have similarly placed DPs with respect to the shape of the CCSs.

The similarity in the DPs can be measured by finding out the zone of each demand point with respect to the shape. If the DPs in both the clusters fall in the same zones, then we can argue that the the DP distribution is similar.

Next, we determine whether the solutions presented in the closest matching shape(s) are within the input budget. We choose the solution with the lowest budget (represented as $SimS$). This enables us to maintain a contingency budget that can be utilized in the event of constraint violations. A closest match will be always found in the database as we know that the database is exhaustive (as described in Section 5.2.1). Algorithm 3 describes this process in further detail.

Algorithm 3: Function $TraverseDB(S, \mathcal{C}_i, \mathcal{M}, \mathcal{DB}, b)$

```
Input: Input cluster $\mathcal{C}_i$, DCM $\mathcal{M}$, Shape $S$ of cluster $\mathcal{C}_i$, Database $\mathcal{DB}$, Budget $\beta$
Output: $SimS$, the closest match of $\mathcal{C}_i$ present in $\mathcal{DB}$
\* Normalize the input cluster $\mathcal{C}_i$ *\
$CCS'_{\mathcal{C}_i}, DP'_{\mathcal{C}_i} = normalize(\{CCS_{\mathcal{C}_i}, \mathcal{M}(\mathcal{C}_i)\})$
$old \leftarrow \infty$
for $(\mathcal{P}_j, shape_j) \in \mathcal{DB}$ do
    \* Match the shapes and the DPs. $\mathcal{P}_j = \{CCS_{\mathcal{P}_j}, DP_{\mathcal{P}_j}, CS_{\mathcal{P}_j}\}$ and $shape_j$ is the
     shape of the cluster in the database. *\
    if $S == shape_j$ and $|\mathcal{M}[\mathcal{C}_i]| == |DP_{\mathcal{P}_j}|$ then
        \* Find the zones of $DP'_{\mathcal{C}_i}$ and compare them with the zones of $DP_{\mathcal{P}_j}$. *\
        if $zones(DP'_{\mathcal{C}_i}) == zones(DP_{\mathcal{P}_j})$ then
            \* Find the matching solution with least budget utilization. *\
            if $|CS_{\mathcal{P}_j}| < \beta$ and $|CS_{\mathcal{P}_j}| < old$ then
                $SimS \leftarrow \mathcal{P}_j$
                $old \leftarrow |CS_{\mathcal{P}_j}|$
            end
        end
    end
end
return SimS
```

5.2.6. Mapping the Precomputed Solution

The database provided the positions for the optimal locations of the CSs for the closest matching cluster ($SimS$) with the least possible budget. To apply the solution of $SimS$ to the input cluster $\mathcal{C}_i$, we need to relocate the CS in $SimS$ with respect to the CCSs in the input cluster. To achieve this, we find a CCS in our input cluster for each CS in $SimS$, which is within the distance threshold of T from the said CS. In the event that no such CCS satisfies this constraint, we select the one which is closest to the CS. We have empirically estimated that we can set T to be 0.1 units (where the points are normalized to the [0-1] range). Once this is complete, we denormalize the CSs to get the final solution. Algorithm 4 describes this process in detail.

Figure 11 shows a graphical illustration of the **ApplySolution** function. Figure 11a represents the input cluster $\mathcal{C}_i$, and its closest matching solution from our database $SimS$ is shown in Figure 11b. Next, we try to find the closest CCS ($CCS_{\mathcal{C}_i}$) in the input cluster for each charging station (CS_S) in $SimS$ – this is shown in Figure 11c. The final solution is presented in Figure 11d.

5.3. Repairing an Infeasible Solution

As with any other hierarchical solution, we must check that the assembled solution satisfies all the constraints of the original problem [28]. We observe that three of our four

Algorithm 4: Function $ApplySolution(SimS, \mathcal{C}_i, \mathcal{M}, T)$

```
1  Input: Input cluster $\mathcal{C}_i$, with $CCS_{\mathcal{C}} = CCS_{\mathcal{C}_1}, CCS_{\mathcal{C}_2}...CCS_{\mathcal{C}_n}$; Closest match
   $SimS = (CCS_S, DP_S, CS_S)$ ; DP mapping $\mathcal{M}$; Threshold $T$
2  Output: Final locations of charging stations $CS_{opt} = CS_1, CS_2...CS_m$
3  \* Normalizing the input cluster. {X, Y} refers to the concatenation of the sets X
   and Y *\
4  $CCS'_{\mathcal{C}}, DP'_{\mathcal{C}} = normalize(\{CCS_{\mathcal{C}}, \mathcal{M}(\mathcal{C}_i)\})$
5  \* Find the closest CCS in the input cluster for each CS $(CS_S)$*\
6  for $CS_j \in CS_S$ do
7      $old = \infty$
8      $c = 1$
9      for $k = 1$ to $n$ do
10         $new = distance(CS_j, CCS'_{\mathcal{C}_k})$
11         if $new < T$ then
12             break
13         end
14         else if $new < old$ then
15             $c = k$
16         end
17     end
18     $CS'_{\mathcal{C}}.append(CCS'_{\mathcal{C}_c})$
19 end
20 \* Denormalize $CS^n_{\mathcal{C}_i}$ to get original solution.*\
21 $CS_{opt} = denormalize(CS'_{\mathcal{C}}, \{CCS_{\mathcal{C}}, \mathcal{M}(\mathcal{C}_i)\})$
22 return $\mathbf{CS_{opt}}$
```

constraints cannot be violated due to the design of PC-ILP – each DP is associated with the CS placed closest to it. So Constraint 1c is never violated. To ensure that Constraint 1d is not violated, we make sure that when applying the solution of $SimS$ to $\mathcal{C}_i$, the CSs placed for $\mathcal{C}_i$ are CCSs, as illustrated in Algorithm 4. We also divide the provided budget across the various shapes in order to ensure that Constraint 1e is not violated. Subsequently, a match for each cluster for a given budget is located in our database.

This leaves the reachability constraint, which may be violated, resulting in an infeasible solution. The solution to the problem is to maintain a contingency budget for use in the event of a constraint violation. This enables us to *repair* the infeasible solution by adding additional CSs for the demand points where a constraint violation is detected, without affecting the pre-existing locations of the CSs. This simplifies the problem, allowing for a fast estimation of the feasible solution.

The initial phase is the verification step in which we determine whether or not a constraint has been violated in the original solution. To determine this, we map all DPs within a reachability distance from a CS. This is performed for all CSs. Next, we determine if any DPs are left unmapped, which indicates that the reachability constraint has been violated (refer to Algorithm 5).

To address this issue, we solve the original optimization problem with a 15% budget reduction, allowing us to use the remaining budget to fix the possibly infeasible solution. This is the *surrogate optimization* problem where we use the PC-ILP algorithm albeit with a reduced budget. First, we query the database with the unmapped DPs and a portion of the remaining budget (using the **TraverseDB** function). The function returns the closest matching shape for the given set of unmapped DPs and CCSs (details in Section 5.2.5). Then, we apply the solution to the matching shape using the **ApplySolution** function, which returns the most optimal placement of the CSs for the given unmapped DPs. We recheck whether or not all the DPs are mapped using the obtained solution within the given

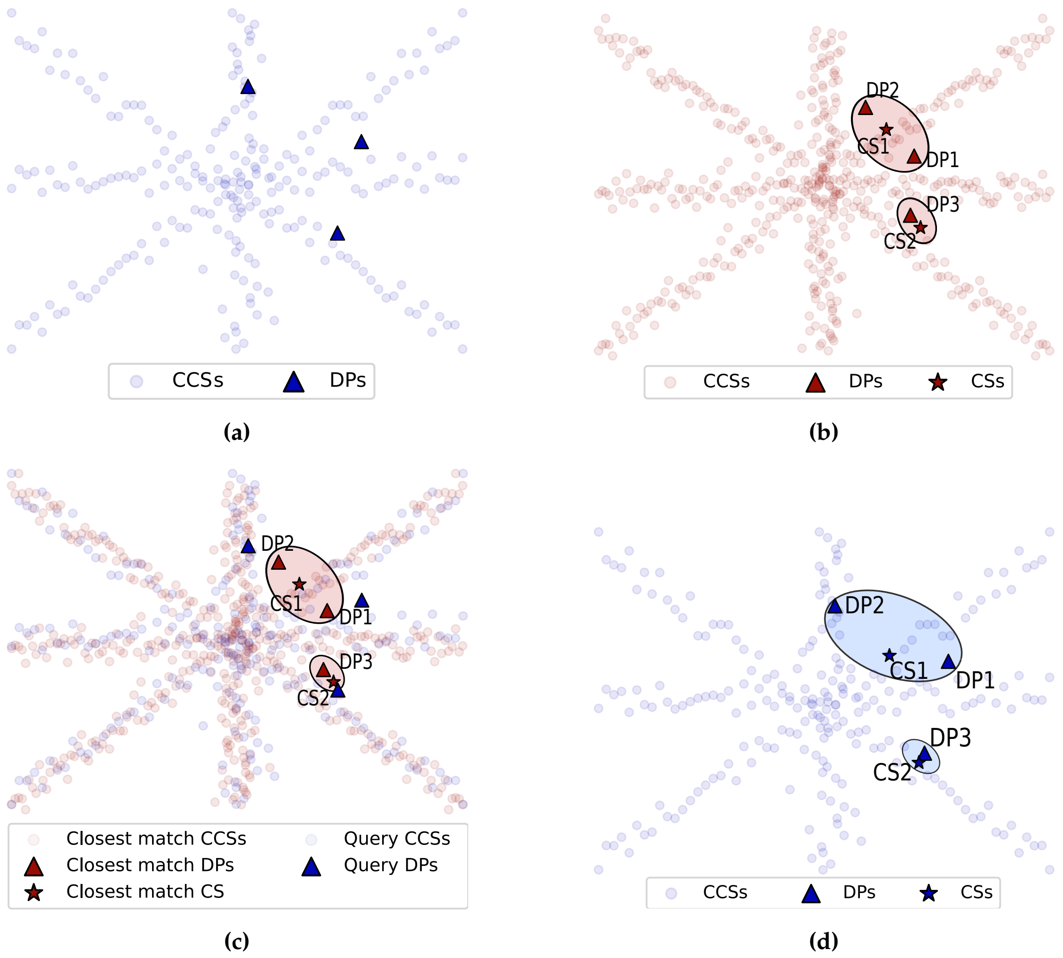


Figure 11. A pictorial representation of the *ApplySolution* function: **(a)** The input cluster $\mathcal{C}_i$ (query); **(b)** Closest matching solution (*SimS*) from the database. CS1 is mapped to DP1 and DP2, CS2 is mapped to DP3; **(c)** Finding the closest $CCS_{\mathcal{C}_i}$ in the input cluster for each *CS* in *SimS*; **(d)** Applying the closet matching solution (CSs) on the input cluster (query)

budget. If they are not mapped, the entire procedure is repeated with a slightly increased budget.

If a feasible solution is found, to arrive at the final repaired solution, we combine both the initial infeasible solution and the surrogate solution for the DPs with constraint violations. The infeasible solution consists of the locations of CSs (*old CSs*) that served a subset of DPs. Sadly, some DPs were left unmapped, and we used the surrogate function to add *new CSs* to serve those DPs using the extra budget. The final solution is essentially a union of the locations of both the sets of the charging stations (*new CSs* and *old CSs*). In cases where the entire budget has been spent and no solution has been found, a solution to the problem is deemed to be impossible.

5.4. Chargers at Each Charging Station (Additional Objective)

Once a solution for the primary objective is obtained from our database, we proceed to solve an ILP problem in order to integrate the additional constraints into the estimated solution. We have referred to this as *Problem 2*. Our experimental observations indicate that the proposed approach has the capability to include a wide variety of additional constraints and objective functions also. Nevertheless, the present version of the work presents a proof-of-concept by considering only the additional electrical constraints, queuing time, and traffic information. Equation 2 (see Section 3.1) introduces the concept of adding multiple chargers at each charging station, which can easily be found out by solving an ILP problem. Note that the size of this problem is much smaller than our original problem,

Algorithm 5: Function $Repair(\mathcal{CS}, \mathcal{DP}, \beta_0, \mathcal{M}, \mathcal{C}_i, \mathcal{S}, b)$

```
Input: CS locations $\mathcal{CS}$, Demand points $\mathcal{DP}$, Database $\mathcal{DB}$, Remaining budget $\beta_0$,
  Shape $\mathcal{S}$ of the given cluster $\mathcal{C}_i$, Budget $\Delta$
Output: New optimally placed charging stations $\mathcal{CS}$
\* Check the constraint violation in the given solution*\
for $\mathcal{CS}_o \in \mathcal{CS}$ do
    $\mathcal{DP}_M \leftarrow$ Find DPs within the $\tau$ distance from $\mathcal{CS}_o$
end
\* Handle the constraint violation *\
if $\mathcal{DP}_M \neq \mathcal{DP}$ then
    for $(DP_i \notin \mathcal{DP}_M)\&(DP_i \in \mathcal{DP})$ do
        $\mathcal{D}.append(DP_i)$
    end
    \* Repeat the process until a feasible solution is found or the budget is
      exhausted *\
    while $b \leq \beta_0$ do
        \* Search the database for the solution *\
        $SimS$ = TraverseDB$(\mathcal{S}, \mathcal{D}, \mathcal{DB})$
        $\mathcal{CS}_r$ = ApplySolution$(SimS, \mathcal{C}_i, \mathcal{D}, b, \mathcal{DB})$
        \* Check the constraint violation in the new solution *\
        for $\mathcal{CS}_o \in \mathcal{CS}_r$ do
            $\mathcal{DP}_M \leftarrow$ Find DPs within the $\tau$ distance from $\mathcal{CS}_o$
        end
        if $\mathcal{DP}_M == \mathcal{D}$ then
            \* No violation - Get the repaired solution *\
            return $\mathcal{CS} \bigcup \mathcal{CS}_r$
        end
        else
            \* Violation - Increase the budget *\
            $b \leftarrow (b + \Delta)$
        end
    end
    return No feasible solution
end
else
    \* No violation *\
    return $\mathcal{CS}$
end
```

where our solution space was much larger. Here, we just have to determine the capacity (in terms of the number of chargers) of each CCS.

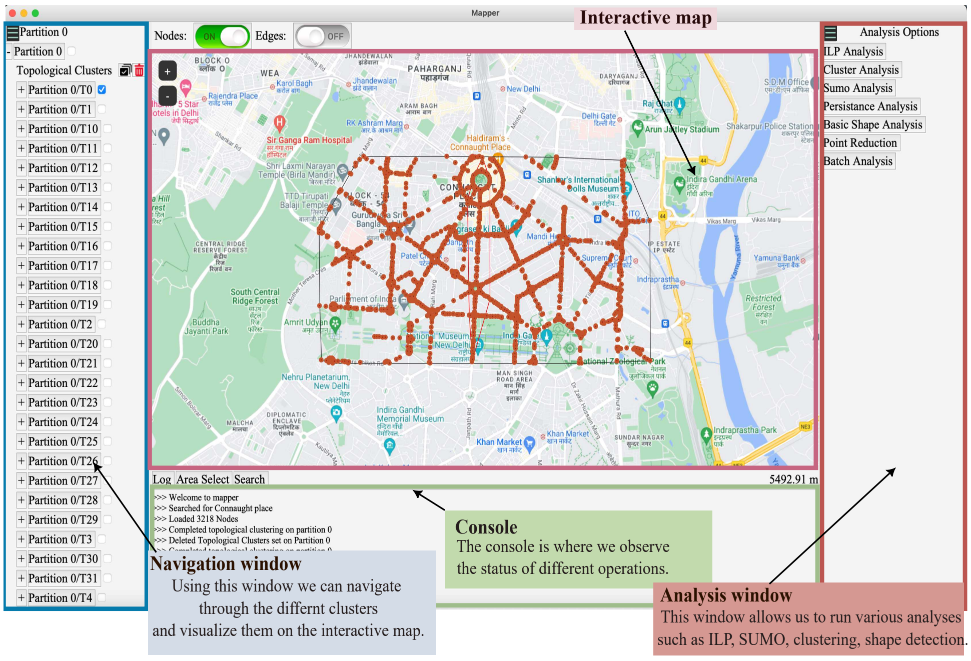
For our experiments, we formulate an ILP, which is in line with the equations shown in Section 3.1. The ILP problem returns the optimal number of chargers at each location (CS). We observe that the execution time for these ILPs is very small as compared to the ILP for Problem 1, due to the fact that its search space is much lower.

6. Results and Discussion

In this section, we shall discuss the evaluation methodologies that are used to compare PC-ILP to other state-of-the-art algorithms. In addition, we characterize the effect of different shapes on the performance of the solution obtained by PC-ILP for various cities.

6.1. Setup

The details of the system setup along with all the software and hardware configurations are shown in Table 1. We have developed an in-house tool named *MapperCS* (MAP-based tool using PERsistence homology for Charging Station placement), which is runs on MacOS, Windows and Ubuntu. The tool has an interactive map in the center of the screen with two side panes. *MapperCS* takes map data from *OpenStreetMaps* [35] using the *overpass* [68] API, filters out many details and retains the street data, which are used to create the CCSs via a manual annotation process. Figure 12 shows a screenshot of the *MapperCS* tool. This highly interactive tool is designed to perform many operations on the city map using two interactive side panes. The users can not only annotate the DPs and potential CSs, but they can also perform more complex operations such as clustering the CCSs and run an ILP for a given set of constraints.



```
>>> Welcome to mapper
>>> Searched for Connaught place
>>> Loaded 3218 Nodes
>>> Completed topological clustering on partition 0
>>> Deleted Topological Clusters set on Partition 0
>>> Completed topological clustering on partition 0
```

Using this window we can navigate through the differnt clusters and visualize them on the interactive map.

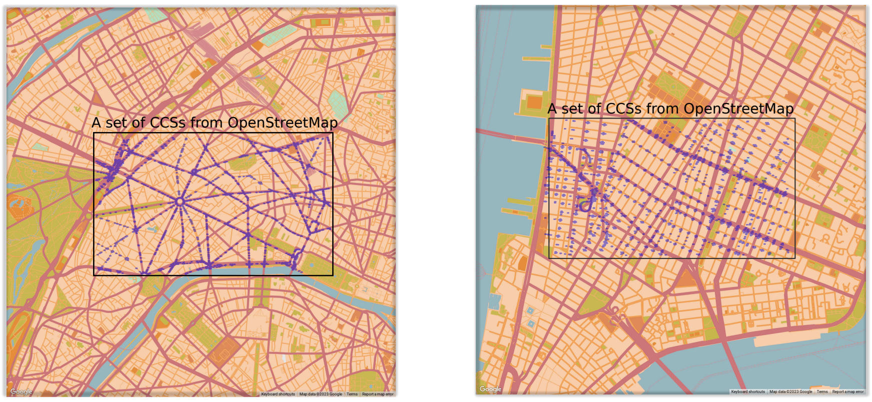
The console is where we observe the status of different operations.

This window allows us to run various analyses such as ILP, SUMO, clustering, shape detection.

Figure 12. A screenshot of the *MapperCS* tool

► **Benchmarks** We consider the top 50 cities by population (source:[69]). We focus on the areas that have a high population density in the cities (35-387 km^2). The details are shown in Table 4. We can make some broad observations based on the maps of the cities (also visualized using our MapperCS tool). We observe that American cities have historically been laid out as grids (meshes), whereas European towns have predominantly adopted a radial organization (all the arterial roads are oriented towards the center of the city). We show two examples in Figure 13 for a section of downtown Paris and New York.

► **Dataset for the CNN-based Algorithm** The shapes of the clusters of CCSs are identified using a CNN model (see Section 5.2.4). The training dataset contains clusters that can be classified into five basic shapes namely *circle, mesh, star, line and concentric circle*. Each cluster is defined by its point cloud representation and the PD of its MAT. We trained our model using synthetic data because in this case we can generate as much as synthetic data as we want (we are not limited by the training set size or real-world constraints regarding the availability of data). For instance, if we want to generate synthetic data for a star, then we laid a random number of points out as a star, and then perturb them randomly. In this way, we can generate a lot of training examples for a given topology. The same approach can be repeated for other topologies and we can continue training our model. Note that there is no need for manual annotation here because we already know which basic shape a


A set of CCSs from OpenStreetMap
A set of CCSs from OpenStreetMap

(a) **(b)**

Figure 13. A section of cities studied using *MapperCS* **(a)** Radial (Paris, France) **(b)** Mesh based (New York City, USA)

City, Country	Area (km^2)	City, Country	Area (km^2)	City, Country	Area (km^2)
New York , USA	386.79	Lima , Peru	241.31	Lahore , Pakistan	160.17
Paris , France	102.34	Xian , China	220.28	Mumbai , India	291.42
Karachi , Pakistan	153.01	Beijing , China	207.69	Moscow , Russia	128.10
Rio de Janeiro , Brazil	171.39	Shanghai , China	157.75	Bangalore , India	125.20
Lagos , Nigeria	165.09	Seoul , South Korea	166.19	Ahmedabad , India	147.39
Hyderabad , India	276.80	Manila , Philippines	151.15	Chicago , USA	139.47
Bogota , Colombia	205.81	Chennai , India	127.07	Delhi , India	257.32
Tokyo , Japan	169.75	Sao Paulo , Brazil	240.79	Hangzhou , China	162.14
Tianjin , China	114.95	Istanbul , Turkey	270.62	Nanjing , China	300.42
Ho Chi Minh , Vietnam	179.39	Kinshasa , Congo	153.25	Cairo , Egypt	170.04
Madrid , Spain	173.26	Chongqing , China	352.70	Osaka , Japan	111.54
Jakarta , Indonesia	183.40	Kolkata , India	150.39	Chengdu , China	170.70
Buenos Aires , Argentina	158.54	Los Angeles , USA	177.53	Dhaka , Bangladesh	185.52
Luanda , Angola	216.53	Kuala Lumpur , Malaysia	287.39	Tehran , Iran	128.08
London , UK	106.80	Nagoya , Japan	103.43	Hong Kong , China	316.96
Shenzhen , China	140.79	Guangzhou , China	132.14	Mexico city , Mexico	192.33
Wuhan , China	198.08	Bangkok , Thailand	183.51	Berlin , Germany	35.80

Table 4. The list of cities considered in our work

given point cloud corresponds to. We used the point clouds (CCS locations) in the 50 cities as test cases. Table 5 shows the number of shapes found across our dataset of 50 cities.

Shapes	Data points
Circle	3600
Line	2889
Star	2189
Concentric circle	1248
Mesh	203

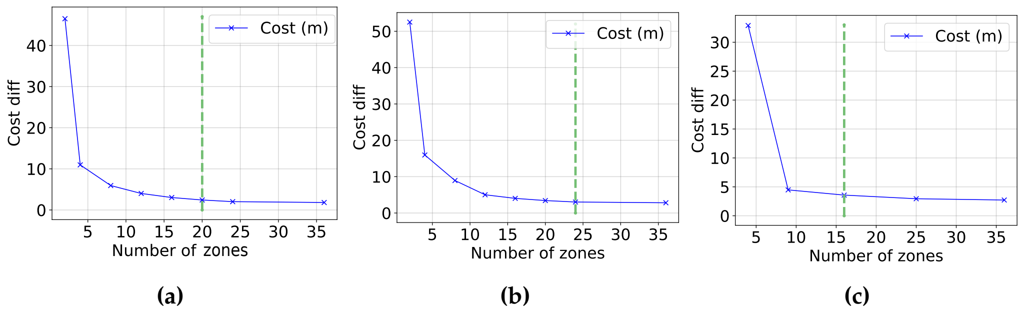
Table 5. Shapes found in our dataset comprising 50 cities

6.2. Parameters for the Creation of the Precomputed Database

6.2.1. Number of Zones in each Shape

The number of zones within a shape affects the total number of DP distribution patterns, which in turn influences the size of the database. We perform extensive experiments to find the total number of *zones* in each basic shape. We estimate the cost (see Equation 1a) using ILP for two scenarios: (a) when the DP is located in the centroid of the zone; and (b) when the DP is located anywhere else in the zone.

We considered 250 randomly generated patterns for DPs (for a given number of CCSs, budget and reachability radius) and computed the difference in the costs for the two scenarios for a given number of *zones*. This experiment is repeated by increasing the number of *zones* in a shape. The results for the average cost difference for different shapes are shown in Figure 14. We observe that after a certain limit, the difference in the cost reduces and reaches a saturation point; the knee point is used to set the number of *zones* in a shape. We conclude that we should divide the circles into 20 *zones*, whereas stars, meshes, and lines should be divided into 16 *zones*, and finally concentric circles should be divided into 24 *zones*. In our experiments, these were found to be the optimal values.



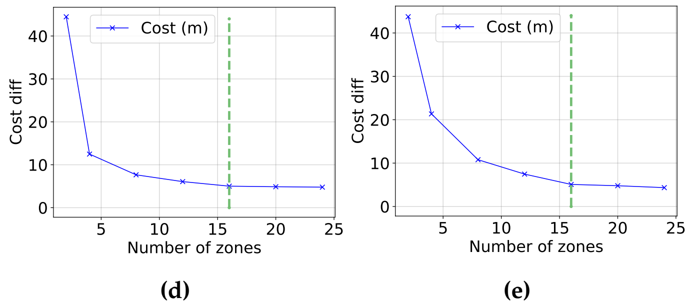


Figure 14. Number of *zones* versus the average difference in the cost for different shapes (arithmetic mean): **(a)** Circle; **(b)** Concentric circle; **(c)** Mesh; **(d)** Star **(e)** Line

6.2.2. Reachability Distance (τ)

The reachability distance (τ) denotes the maximum distance that an EV user needs to travel from its DP to the nearest CS to charge the EV. We performed exhaustive experimentation to analyze the effect of the reachability distance on the cost of the solution. For each shape, we first computed the ILP solution with $\tau = \tau_{input}$, where τ_{input} is the input reachability distance. After this, we compute the ILP with $\tau \in [0.2\tau_{input}, 2\tau_{input}]$. Figure 15 shows the variation in the cost function with the reachability distance for a circle with a constant budget of 4 CSs, 117 CCSs and 3 DPs. We observe that after a certain value of the reachability distance, the cost of the solution remains nearly constant. A similar observation is highlighted by Gopalakrishnan et al. [70]. We ensure that the database has solutions for all potential values of the reachability distance prior to the saturation point for different combinations of the DP distribution and the budget.

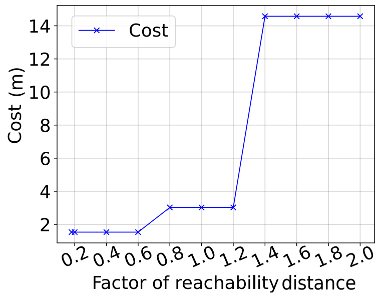


Figure 15. Variation of the cost with respect to the reachability distance

6.2.3. Budget (β)

We allocate a budget to a city that represents the total number of CSs that can be placed in the city. However, we know that placing a CS without demand wastes resources. Therefore, the budget and the demand points are directly related. In the worst-case scenario, each DP will have its own CS. Therefore, the total number of CSs (or budget) can never exceed the number of DPs. This establishes an upper limit on the budget.

Additionally, we observe that our clusters are created in such a way that a cluster can have a maximum of three demand points. This is because big cities may consist of many small sized clusters as the *ToMATo* clustering generates many small clusters as shown in Section 5.2.3, and demand points will be divided across these small clusters. We experimentally validated that a cluster with more than three DPs is quite unlikely. We ensure that our database contains all possible solutions within the specified budget range for each cluster.

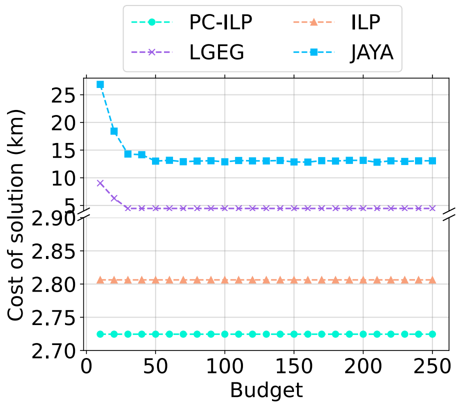
6.3. Performance Analysis

In this section, we evaluate our schemes for a set of micro and macro benchmarks. We consider two metrics for evaluation; ① the execution time of each algorithm and ② the cost of the solution provided by the algorithm (see Equation 1a).

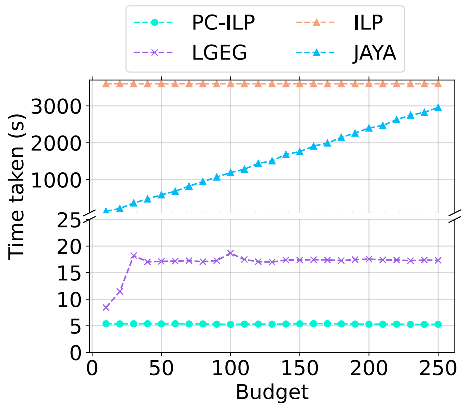
6.3.1. Microbenchmarks

To evaluate the efficiency of PC-ILP, we compare it against the standard ILP approach[47] (timeout set to 1 hour), state-of-the-art *Lazy Greedy with Efficient Gain* (LGEG) algorithm [8] and a fast meta-heuristic *JAYA* algorithm [5], which uses a genetic algorithm. As the ILP algorithm is time and memory intensive, we compare the algorithms on a microbenchmark, which is a section of Berlin, Germany. We select an area of 32.50 km^2 which contains approximately 4100 CCSs. We reduce the number of CCSs in this area to 2500 due to the memory constraints of the ILP. We chose 30 DPs and a budget of 35 CS. The DPs are generated uniformly at random, similar to Lam et al. [71]. Additionally, in order to run the ILP with the given setup, we set an upper bound on the execution time to avoid memory overflow. After exhaustive experimentation, we select an upper bound of 3600 seconds.

► **Cost vs. Budget -** We evaluate the performance of the algorithms by gradually increasing the allocated budget while keeping the CCSs and DPs fixed.



(a)



(b)

Figure 16. A comparison between PC-ILP, state-of-the-art LGEG, standard ILP, and JAYA against the budget: **(a)** Cost; **(b)** Performance

Figure 16 compares the solutions provided by the various algorithms against the allocated budget. Figure 16a shows that the cost of the solution provided by JAYA and LGEG is worse than both ILP and PC-ILP. Specifically, PC-ILP outperforms LGEG by **38.87%** and JAYA by **5.09**× with respect to the solution cost, which is expected as LGEG is not operating exhaustively on the CCSs and DPs and genetic algorithms are known to scale poorly against complexity due to the exponential increase in the size of the search space.

Upon closer inspection, we see that the solutions provided by PC-ILP are marginally better than the ILP solutions. Figure 16b shows that PC-ILP and LGEG are faster than standard ILP. This is expected the ILP is more time and memory intensive than both LGEG and PC-ILP. We also observe that PC-ILP provides an average performance improvement of **3.27**× over LGEG and **289.37**× speedup vis-a-vis JAYA.

► **Cost vs. DPs -** Next, we study the impact of gradually increasing the number of DPs while keeping the CCSs and the budget constant. We evaluate the algorithms on the same metrics as before. Figure 17 compares the three algorithms plotted against increasing DPs. Figure 17a shows that PC-ILP performs better than the standard ILP, LGEG, and JAYA over the entire range of DPs with respect to cost. We also observe that all costs rise with an increase in the number of DPs. This is due to the budget being constant.

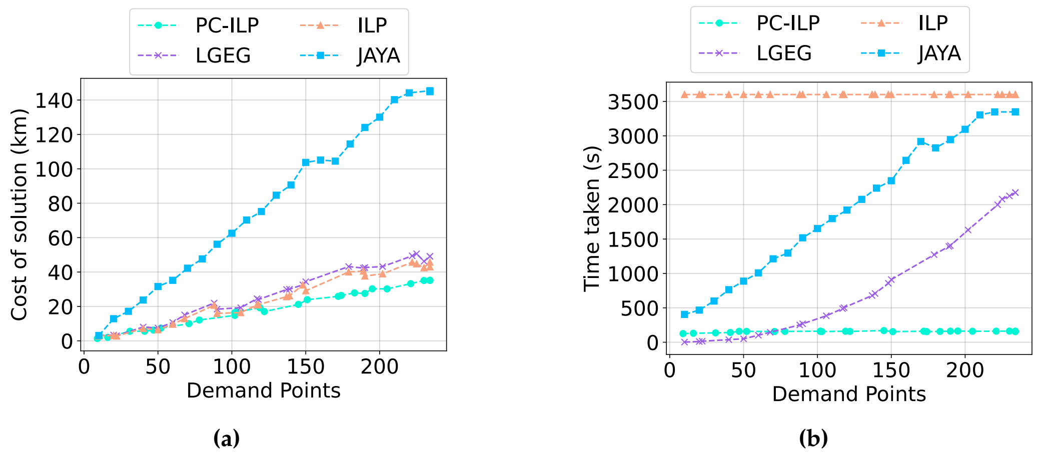


Figure 17. A comparison between PC-ILP, state-of-the-art LGEG, standard ILP, and JAYA against the number of DPs: **(a)** Cost; **(b)** Performance.

Figure 17b shows that the performance of PC-ILP is unaffected by the number of DPs, as the execution time of PC-ILP is directly proportional to the number of clusters and the time taken to process the DPs is minimal. On the other hand, the performance of LGEG and JAYA deteriorate with a gradual increase in DPs since the execution time of LGEG depends on the number of DPs and CCSs, while the execution time of JAYA depends on the budget, CCSs, and DPs.

Summary: We have thus established that PC-ILP performs far better than LGEG, JAYA, and standard ILP while providing marginally better solutions at the same time. Subsequently, we focus on estimating the performance of all the algorithms on a set of macrobenchmarks.

6.3.2. Macrobenchmarks: 50 Cities

Next, we evaluate the performance of PC-ILP against ILP, LGEG and JAYA on a set of macrobenchmarks with the same hardware configurations as mentioned in Table 1. We run this code on our full set of 50 cities. Because of a lack of space, we shall only present the results for 14 cities (representative ones). Figure 18 shows the costs of solutions provided by PC-ILP, ILP, LGEG and JAYA along with their average performance.

We observe that, on average, PC-ILP provides a solution that is **36.62%** better than ILP under the same system configurations, while providing **21.09%** better solutions than LGEG and **30.34%** better solutions than JAYA. Figure 19 shows the time taken to compute the solution using PC-ILP and the ILP for some cities, along with the average execution time for across all cities. We observe that PC-ILP is nearly **1.5**× faster than LGEG, **1.78**× faster than JAYA and **30**× faster than ILP.

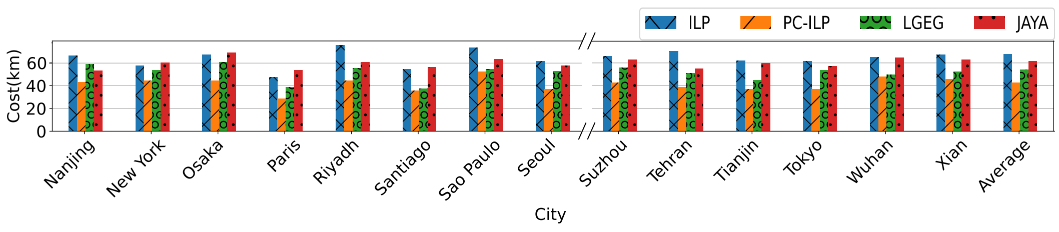


Figure 18. Comparison of the cost of the solution for different cities

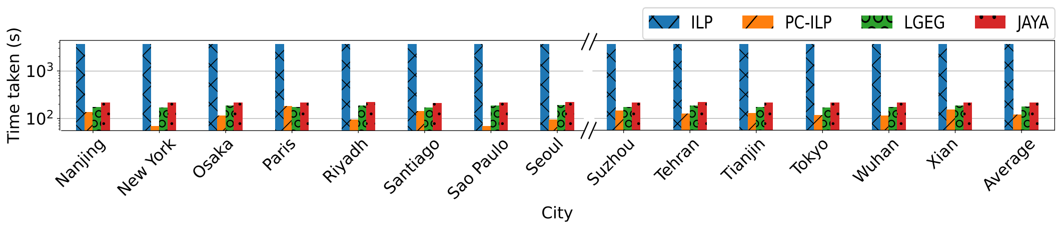


Figure 19. Comparison of the performance for different cities

6.4. Scalability Analysis

In this section, we evaluate the scalability of the proposed scheme with an increasing number of CCSs. We compute the execution time while increasing the total number of CCSs linearly as shown in Figure 20. *We observe that PC-ILP scales linearly with an increase in CCSs and clusters, whereas the standard ILP is known to be an NP-hard problem and will not scale in the same way*. This means that the proposed scheme performs **30**× better than the standard ILP method for a large-area city maps under the given system configuration and assumptions. The cost of the solutions in both the methods remain nearly the same.

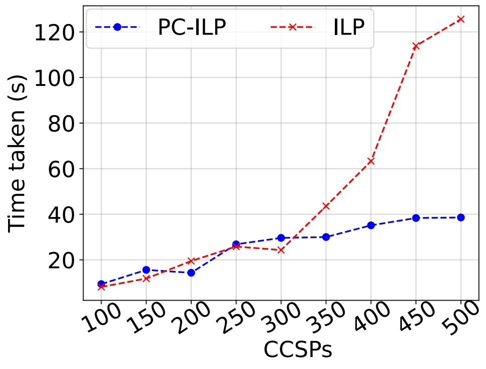


Figure 20. Performance comparison of PC-ILP and standard ILP for different number of CCSs

6.5. Overheads of Fixing Violated Constraints

We note that when we repair the solution upon discovering the violation of any constraint, an additional cost is borne. Constraints are violated in only 6 of the 50 actual real-world cities (only **12%** of cases). After repairing the solution and fixing the constraint violation, we were able to get a solution for all the real-world use cases (real cities). For reference, an average-sized cluster is approximately $250,000\ m^2$. Without corrections in these cases, the average cost (i.e., cost for an average-sized cluster) is 44.94 m (average distance from a DP to its mapped CS). Post corrections, an additional cost of **4.87 m** is borne, bringing the total cost to 49.81 m and only an additional 4% of the total execution time is spend for repairing the solution.

6.6. Overheads of Adding Additional Constraints

We evaluated the proposed scheme with a set of additional constraints (detailed in Section 3.1). For this evaluation, we consider the same section of Berlin, Germany, with 500

CCSs, 10 DPs, a CSs' budget (β) of 3, a chargers' budget (β_a) of 30, traffic monitoring time (t_c) of 24 hours and EV charging time (t_s) as 30 minutes (same as [72]). We incorporate the traffic information using the SUMO traffic simulator as shown in Figure 21. The simulator provides us with the traffic estimate at each CS. Figure 21 depicts the total number of chargers placed at each CS using the proposed algorithm.

We observe that the cost of the PC-ILP and ILP solutions are nearly the same as both algorithms place their CSs in nearly the same locations. The time taken to generate a solution using a combination of PC-ILP along with ILP (for additional constraints) is 34.6 seconds, while the time taken by standard ILP is 112.7 seconds, with nearly the same cost. So, we conclude that the proposed technique can accommodate any extra constraints with minimal performance loss.

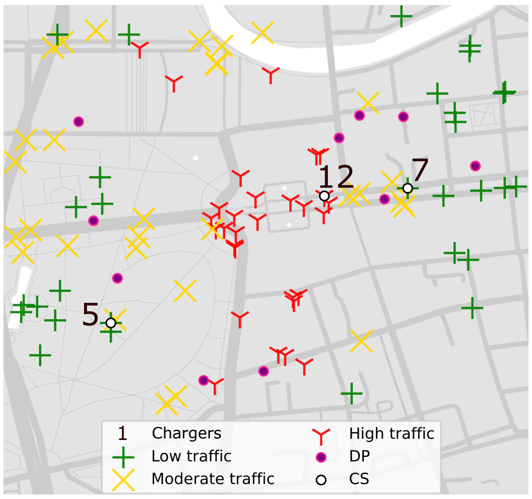


Figure 21. Experiment: Charging station placement with additional constraints. A section of Berlin, Germany. The numeric digits represent the number of chargers placed in a charging station.

7. Related Work

We extensively analyzed prior work and classified the proposed solutions for solving the EV charging placement problem into three main families of approaches ❶ mathematical programming based approaches [73,74]; ❷ heuristic or meta-heuristic based approaches [75–77]; and ❸ hybrid approaches [13,78–81].

7.1. Mathematical Programming based Approaches

The mathematical programming based approaches commonly define the task of determining the optimal placement of charging stations as an optimization problem, wherein the objective functions are created based on different criteria such as the maximum distance between a DP and CS or the number of CSs. This family of approaches includes several classical approaches such as ILP, quadratic programming, mixed integer nonlinear programming (MINLP), etc.

Brandstatter et al. [73] propose utilizing ILP optimization models in order to find the optimal location and sizes (number of chargers) of CSs by considering the expected number of trips made by a shared EV based on the user demand and battery state. They use the IBM ILOG CPLEX solver as a means of solving this problem and show that the problem exhibits NP-hardness when either the budget or the battery capacity of EVs is constrained. However, if we assume that both the budget and the battery capacity are not constrained, the problem may be solved efficiently in polynomial time. Sadly, it is not a realistic assumption. Similarly, Ullah et al. [2] employed multiple sets of constraints with a classical ILP approach to ensure a good coverage by each CS, but this approach is extremely time-consuming and memory-intensive. Furthermore, the authors fail to take into account the queuing time and traffic density in the given region, making the implementation of the proposed strategy unfeasible for real-world settings.

Kockelman et al. [74] used mixed integer programming techniques to optimize the problem of CS placement, taking into account the parking demand and the costs paid by users in accessing the CS. The estimation of parking demand was conducted by considering the factors such as site accessibility, local employment opportunities, and population densities. The approach only determines the ideal neighbourhoods for the placement of CSs, it does not specify the exact location and size of the CSs. Additionally, the majority of mathematical programming based solutions have the drawback of requiring substantial computational and storage resources, which serves as the greatest barrier to their deployment in real-world scenarios.

7.2. Heuristic based Approaches

Heuristic based approaches have been shown to effectively solve big and complex optimization models in an efficient manner as compared to mathematical programming based methods.

Lam et al. [76] propose and analyze multiple approaches to solve the CS placment problem; they highlight the fact that a greedy approach is most helpful and proves to be the most scalable if we want to solve the problem in polynomial time . However, the authors only considered the reachability distance and the coverage by the CSs; they did not account for the queuing time and traffic conditions, limiting the solution's applicability in real-world scenarios. Zhang et al. [8] present two heuristic-based approaches Lazy Greedy with Direct Gain (LGDG) and Lazy Greedy with Effective Gain (LGEG) utilizing a greedy algorithm to determine the ideal location of charging stations. They take into account multiple constraints like the charging demand, budget and cruising range. They utilized real-world GPS trajectory data spanning a duration of 87 days, obtained from taxi cabs in Beijing, China to get an estimate of the high-demand trajectories. However, the authors only considered parking lots to be potential CSs, which limits the applicability of their scheme. Also, the application of such approaches in large-scale road networks presents obstacles due to the computationally intensive nature of their algorithm.

Shaoyun et al. [75] proposed a method of finding the optimal locations and sizes of the charging stations by dividing a given area into small identical partitions. The number of initial partitions is estimated based on the charging demand of EVs as well as on the maximum and minimum capacity of a charging station. The loss is estimated as the sum of weighted distances from the CS to the demand points. A genetic algorithm is used to provide the final number of charging stations by minimizing the loss and adjusting the number and coverage of the partitions accordingly. Sadly, the queuing time and traffic density of the roads are not considered during the formation of the initial partition, which implies that the solution is scenario-specific and is not universal. Mohanty et al. [77] employ another population-based meta-heuristic algorithm known as JAYA to determine the ideal sites for charging stations (CSs) with an aim to reduce both the installation and operation costs associated with the CSs. Our analyses show that PC-ILP is not only faster than the JAYA algorithm but can generate 30% better solutions.

7.3. Hybrid Approaches

Hybrid approaches are techniques where a combination of multiple schemes (greedy algorithm, genetic algorithm, ILP, MILNP, etc.) are used to solve the CS placement problem. Awasthi et al. [78] employ a hybrid approach that combines a genetic algorithm with an enhanced version of standard particle swarm optimization in order to determine the ideal placement and size of the CSs. The particle swarm optimization method is capable of optimizing the sub-optimal solution estimated using the genetic algorithm, resulting in improved algorithm functionality and enhanced solution quality.

Kavianipour et al.[13] also estimate the locations for charging stations and the number of chargers at each location by employing a decomposition-based strategy (similar to our hierarchical strategy) that minimizes the total system cost, which includes charging station and charger installation costs. The initial subproblem uses commercially available solvers and a meta-heuristic algorithm to determine the precise location of the charging stations. This subproblem also generates the information about the traffic flow and energy demand at each CS. This information is sent to the subsequent subproblem, which is a non-linear mixed-integer mathematical model that determines the optimal number of chargers to be installed at each of these stations in order to minimize both the charger installation cost and the waiting time.

PC-ILP also employs a hybrid strategy in which we first estimate the location of charging stations using a hierarchical decomposition framework that uses a topological clustering algorithm to divide a large city area into clusters. Then, we estimate the location of CSs in each cluster using a pre-computed database. In the second phase, we estimate the optimal number of chargers in each location using ILP based on the traffic and electrical constraints. The strategy enabled us to break down the large complex CS placement and sizing problem into simpler components, thus enhancing the computational speed and memory requirement while maintaining the quality of the solution.

8. Conclusion

The problem of EV charging station placement is a classical problem, which is both old and new. There are no two opinions about the fact that it is an established problem and there are numerous meta-heuristic algorithms and pseudo-polynomial time algorithms that provide good solutions. However, the reason that this problem still represents an active area of research is that humans are not in the loop and there is a lot of scope for further improvement. We need to understand that any solution for a smart city will have to take the unique design of the city, its history and the will of the residents into account. Hence, there needs to be a way to have humans in the loop whenever there is some kind of automated city planning. Often, it is hard to represent such desires mathematically and any attempt to do so using standard metrics such as reducing traffic congestion or the distance between DPs and its mapped CSs leads to either very complex solutions or very time-consuming ones that are not intuitive. Hence, we opted for a very different line of thinking in this paper. We relied on the fact that regardless of the city, it is always composed of 5 basic primitives (with some variation).

Using these primitives as the basic design elements, we can create intuitive methods to analyze the map of a city, partition it, propose bespoke solutions for each separate cluster and pre-compute solutions for each cluster (based on the basic primitives). Pre-computation leads to a large speedup and also leads to solutions that are explainable and carry a degree of intuition.

The other major design decision that we used in this paper was the use of surrogate functions and a subsequent repair-based methodology. This allowed us to hierarchically decompose both the map of the city as well as the set of constraints. We were able to compute small solutions and combine them. Whenever a constraint was violated, we were able to quickly fix the violation using our repair function. Along with speed, the advantage of this approach lies in the fact that we can accommodation an arbitrary number of complex constraints and secondary objective functions that are based on variables like the queueing time, traffic density, nature and amount of the electrical loading (discussed in Section 3.1), etc. We expect that this paper will illuminate the path of intuitive topology-driven approaches for future work in this area.

Author Contributions: Conceptualization, Mehul Bose, Bivas Ranjan Dutta, Nivedita Shrivastava and Smruti R. Sarangi; Methodology, Mehul Bose, Bivas Ranjan Dutta, Nivedita Shrivastava and Smruti R. Sarangi; Software, Mehul Bose and Bivas Ranjan Dutta; Validation, Nivedita Shrivastava and Smruti R. Sarangi; Formal analysis, Nivedita Shrivastava and Smruti R. Sarangi; Writing—Original draft preparation, Mehul Bose and Bivas Ranjan Dutta; Writing—review and editing, Nivedita Shrivastava and Smruti R. Sarangi; Supervision, Smruti R. Sarangi; Project administration, Smruti R. Sarangi; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolution Neural Network
EV	Electric vehicle
CCS	Candidate charging stations
CS	Charging station
DP	Demand point
ILP	Integer Linear Programming
PC-ILP	Persistence-based Clustering assisted Integer Linear Programming
MAT	Medial Axis Transform
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
SUMO	Simulation of Urban Mobility
PD	Persistence Diagram
ToMATo	Topological Mode Analysis Tool
LGEG	Lazy Greedy with Effective Gain
LGDG	Lazy Greedy with Direct Gain
MINLP	Mixed integer nonlinear programming

References

1. Kapustin, N.O.; Grushevenko, D.A. Long-term electric vehicles outlook and their potential impact on electric grid. *Energy Policy* **2020**, *137*, 111103. https://doi.org/https://doi.org/10.1016/j.enpol.2019.111103.
2. Ullah, I.; Liu, K.; Layeb, S.B.; Severino, A.; Jamal, A. Optimal Deployment of Electric Vehicles' Fast-Charging Stations. *Journal of Advanced Transportation* **2023**, *2023*, 6103796. TY - JOUR, https://doi.org/10.1155/2023/6103796.
3. Zhong, P.; Xu, A.; Kang, Y.; Zhang, S.; Zhang, Y. An optimal deployment scheme for extremely fast charging stations. *Peer-to-Peer Networking and Applications* **2022**, *15*, 1486–1504. TY - JOUR, https://doi.org/10.1007/s12083-022-01306-7.
4. Shafiei, M.; Ghasemi-Marzbali, A. Fast-charging station for electric vehicles, challenges and issues: A comprehensive review. *Journal of Energy Storage* **2022**, *49*, 104136. https://doi.org/https://doi.org/10.1016/j.est.2022.104136.
5. Mohanty, A.K.; Babu, P.S. Optimal Placement of Electric Vehicle Charging Stations Using JAYA Algorithm. In Proceedings of the Recent Advances in Power Systems; Gupta, O.H.; Sood, V.K., Eds.; Springer Singapore: Singapore, 2021; pp. 259–266.
6. Zafar, U.; Bayram, I.S.; Bayhan, S. A GIS-based Optimal Facility Location Framework for Fast Electric Vehicle Charging Stations. In Proceedings of the 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE), 2021, pp. 1–5. https://doi.org/10.1109/ISIE45552.2021.9576448.
7. Wu, X.; Feng, Q.; Bai, C.; Lai, C.S.; Jia, Y.; Lai, L.L. A novel fast-charging stations locational planning model for electric bus transit system. *Energy* **2021**, *224*, 120106. https://doi.org/https://doi.org/10.1016/j.energy.2021.120106.
8. Zhang, Y.; Wang, Y.; Li, F.; Wu, B.; Chiang, Y.Y.; Zhang, X. Efficient Deployment of Electric Vehicle Charging Infrastructure: Simultaneous Optimization of Charging Station Placement and Charging Pile Assignment. *IEEE Transactions on Intelligent Transportation Systems* **2021**, *22*, 6654–6659. https://doi.org/10.1109/TITS.2020.2990694.
9. Fang, C.; Lu, H.; Hong, Y.; Liu, S.; Chang, J. Dynamic Pricing for Electric Vehicle Extreme Fast Charging. *Trans. Intell. Transport. Syst.* **2020**, *22*, 531–541. https://doi.org/10.1109/TITS.2020.2983385.
10. Bilal, M.; Rizwan, M.M. Electric vehicles in a smart grid: a comprehensive survey on optimal location of charging station. *IET Smart Grid* **2020**.
11. Vazifeh, M.M.; Zhang, H.; Santi, P.; Ratti, C. Optimizing the deployment of electric vehicle charging stations using pervasive mobility data. *Transportation Research Part A: Policy and Practice* **2019**, *121*, 75–91. https://doi.org/https://doi.org/10.1016/j.tra.2019.01.002.
12. Khan, W.; Ahmad, F.; Alam, M.S. Fast EV charging station integration with grid ensuring optimal and quality power exchange. *Engineering Science and Technology, an International Journal* **2019**, *22*, 143–152. https://doi.org/https://doi.org/10.1016/j.jestch.2018.08.005.
13. Kavianipour, M.; Fakhrmoosavi, F.; Singh, H.; Ghamami, M.; Zockaie, A.; Ouyang, Y.; Jackson, R. Electric vehicle fast charging infrastructure planning in urban networks considering daily travel and charging behavior. *Transportation Research Part D: Transport and Environment* **2021**, *93*, 102769.
14. Sachan, S.; Deb, S.; Singh, S.N.; Singh, P.P.; Sharma, D.D. Planning and operation of EV charging stations by chicken swarm optimization driven heuristics. *Energy Conversion and Economics* **2021**, *2*, 91–99.

15. Wang, J. Optimization of Ev Charging Pile Layout on Account of Ant Colony Algorithm. In Proceedings of the Cyber Security Intelligence and Analytics; Xu, Z.; Alrabaee, S.; Loyola-González, O.; Cahyani, N.D.W.; Ab Rahman, N.H., Eds.; Springer Nature Switzerland: Cham, 2023; pp. 450–458.
16. Garcia Alvarez, J.; González, M.Á.; Rodriguez Vela, C.; Varela, R. Electric vehicle charging scheduling by an enhanced artificial bee colony algorithm. *Energies* **2018**, *11*, 2752.
17. Yang, H.; Gao, Y.; Farley, K.B.; Jerue, M.; Perry, J.; Tse, Z. EV usage and city planning of charging station installations. In Proceedings of the 2015 IEEE Wireless Power Transfer Conference (WPTC), 2015, pp. 1–4. https://doi.org/10.1109/WPT.2015.7139139.
18. Mahadeva Iyer, V.; Gulur, S.; Gohil, G.; Bhattacharya, S. Extreme fast charging station architecture for electric vehicles with partial power processing. 2018, pp. 659–665. https://doi.org/10.1109/APEC.2018.8341082.
19. Liu, Y.; Zhu, Y.; Cui, Y. Challenges and opportunities towards fast-charging battery materials. *Nature Energy* **2019**, *4*, 540–550.
20. Chen, X.; Li, Z.; Dong, H.; Hu, Z.; Mi, C. Enabling Extreme Fast Charging Technology for Electric Vehicles. *IEEE Transactions on Intelligent Transportation Systems* **2021**, *22*, 466–470. https://doi.org/10.1109/TITS.2020.3045241.
21. Ge, S.; Feng, L.; Liu, H. The planning of electric vehicle charging station based on Grid partition method. In Proceedings of the 2011 International Conference on Electrical and Control Engineering, 2011, pp. 2726–2730. https://doi.org/10.1109/ICECENG.2011.6057636.
22. Du, B.; Tong, Y.; Zhou, Z.; Tao, Q.; Zhou, W. Demand-aware charger planning for electric vehicle sharing. *In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* **2018**, pp. 1330–1338. https://doi.org/https://doi.org/10.1145/3219819.3220032.
23. Emelichev, V.; Girlich, E.; Nikulin, Y.; Podkopaev, D. Stability and Regularization of Vector Problems of Integer Linear Programming. *Optimization* **2002**, *51*, 645–676. https://doi.org/10.1080/0233193021000030760.
24. Sriabisha, R.; Yuvaraj, T. Optimum placement of Electric Vehicle Charging Station using Particle Swarm Optimization Algorithm. In Proceedings of the 2023 9th International Conference on Electrical Energy Systems (ICEES). IEEE, 2023, pp. 283–288.
25. Zhang, L.; Gao, T.; Cai, G.; Hai, K.L. Research on electric vehicle charging safety warning model based on back propagation neural network optimized by improved gray wolf algorithm. *Journal of Energy Storage* **2022**, *49*, 104092.
26. Jordanov, I.; Jain, R. *Knowledge-based and intelligent information and engineering systems*; Springer, 2010.
27. Fredriksson, H.; Dahl, M.; Holmgren, J. Optimal placement of charging stations for electric vehicles in large-scale transportation networks. *Procedia computer science* **2019**, *160*, 77–84.
28. Chaieb, M.; Jemai, J.; Mellouli, K. A hierarchical decomposition framework for modeling combinatorial optimization problems. *Procedia Computer Science* **2015**, *60*, 478–487.
29. Koch, P.; Bagheri, S.; Konen, W.; Foussette, C.; Krause, P.; Bäck, T. A new repair method for constrained optimization. In Proceedings of the Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 273–280.
30. Bagheri, S.; Konen, W.; Emmerich, M.; Bäck, T. Self-adjusting parameter control for surrogate-assisted constrained optimization under limited budgets. *Applied Soft Computing* **2017**, *61*, 377–393.
31. Batty, M.; Longley, P.A. *Fractal cities: a geometry of form and function*; Academic press, 1994.
32. Kattenborn, T.; Leitloff, J.; Schiefer, F.; Hinz, S. Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing* **2021**, *173*, 24–49. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2020.12.010.
33. Amenta, N.; Choi, S.; Kolluri, R.K. The power crust, unions of balls, and the medial axis transform. *Computational Geometry* **2001**, *19*, 127–153. Combinatorial Curves and Surfaces, https://doi.org/https://doi.org/10.1016/S0925-7721(01)00017-7.
34. Patel, A. Generalized persistence diagrams. *Journal of Applied and Computational Topology* **2018**, *1*, 397–419. https://doi.org/10.1007/s41468-018-0012-6.
35. OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.
36. MacQueen, J.; et al. Some methods for classification and analysis of multivariate observations. In Proceedings of the Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Oakland, CA, USA, 1967, number 14 in 1, pp. 281–297.
37. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. AAAI Press, 1996, KDD'96, p. 226–231.
38. Ankerst, M.; Breunig, M.M.; Kriegel, H.P.; Sander, J. OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Rec.* **1999**, *28*, 49–60. https://doi.org/10.1145/304181.304187.
39. Chazal, F.; Guibas, L.J.; Oudot, S.Y.; Skraba, P. Persistence-Based Clustering in Riemannian Manifolds. *J. ACM* **2013**, *60*. https://doi.org/10.1145/2535927.
40. Lee, D.T. Medial Axis Transformation of a Planar Shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1982**, *PAMI-4*, 363–369. https://doi.org/10.1109/TPAMI.1982.4767267.
41. Chazal, F.; Michel, B. An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists. *Frontiers in Artificial Intelligence* **2021**, *4*. https://doi.org/10.3389/frai.2021.667963.
42. Maria, C.; Dlotko, P.; Rouvreau, V.; Glisse, M. Rips complex. In *GUDHI User and Reference Manual*, 3.8.0 ed.; GUDHI Editorial Board, 2023.
43. Rouvreau, V.; Montassif, H. Čech complex. In *GUDHI User and Reference Manual*, 3.8.0 ed.; GUDHI Editorial Board, 2023.

44. Goričan, P.; Žiga Virk. Critical edges in Rips complexes and persistence, 2023, [arXiv:math.AT/2304.05185].
45. Krajzewicz, D., Traffic Simulation with SUMO – Simulation of Urban Mobility. In *Fundamentals of Traffic Simulation*; Springer New York: New York, NY, 2010; pp. 269–293. https://doi.org/10.1007/978-1-4419-6142-6_7.
46. Rao, R. Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems. *International Journal of Industrial Engineering Computations* **2016**, *7*, 19–34.
47. Dilip, K.N. Optimal Placement of Electric Vehicle Charging Stations with Electricity Theft Control. *Indian Institute of Technology Delhi* **2020**.
48. Chen, H.; Hu, Z.; Luo, H.; Qin, J.; Rajagopal, R.; Zhang, H. Design and Planning of a Multiple-Charger Multiple-Port Charging System for PEV Charging Station. *IEEE Transactions on Smart Grid* **2019**, *10*, 173–183. https://doi.org/10.1109/TSG.2017.2735636.
49. Zhang, H.; Hu, Z.; Xu, Z.; Song, Y. Optimal Planning of PEV Charging Station With Single Output Multiple Cables Charging Spots. *IEEE Transactions on Smart Grid* **2017**, *8*, 2119–2128. https://doi.org/10.1109/TSG.2016.2517026.
50. Garwa, N.; Niazi, K.R. Impact of EV on integration with grid system–a review. In Proceedings of the 2019 8th international conference on power systems (ICPS). IEEE, 2019, pp. 1–6.
51. Zhu, J.; Li, Y.; Yang, J.; Li, X.; Zeng, S.; Chen, Y. Planning of electric vehicle charging station based on queuing theory. *The Journal of Engineering* **2017**, *2017*, 1867–1871.
52. Zhao, Z.; Li, X.; Zhou, X. Distribution route optimization for electric vehicles in urban cold chain logistics for fresh products under time-varying traffic conditions. *Mathematical Problems in Engineering* **2020**, *2020*, 1–17.
53. Sester, M. Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science* **2005**, *19*, 871–897, [https://doi.org/10.1080/13658810500161179]. https://doi.org/10.1080/13658810500161179.
54. Sakai, T.; Imiya, A. Fast spectral clustering with random projection and sampling. In Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition. Springer, 2009, pp. 372–384.
55. Yuan, F.; Sawaya, K.E.; Loeffelholz, B.C.; Bauer, M.E. Land cover classification and change analysis of the Twin Cities (Minnesota) Metropolitan Area by multitemporal Landsat remote sensing. *Remote sensing of Environment* **2005**, *98*, 317–328.
56. Zhang, H.; Hu, Z.; Xu, Z.; Song, Y. Optimal planning of PEV charging station with single output multiple cables charging spots. *IEEE Transactions on Smart Grid* **2016**, *8*, 2119–2128.
57. Chen, H.; Hu, Z.; Luo, H.; Qin, J.; Rajagopal, R.; Zhang, H. Design and planning of a multiple-charger multiple-port charging system for PEV charging station. *IEEE Transactions on Smart Grid* **2017**, *10*, 173–183.
58. Yang, Q.; Sun, S.; Deng, S.; Zhao, Q.; Zhou, M. Optimal sizing of PEV fast charging stations with Markovian demand characterization. *IEEE Transactions on Smart Grid* **2018**, *10*, 4457–4466.
59. Sklansky, J.; Gonzalez, V. Fast polygonal approximation of digitized curves. *Pattern Recognition* **1980**, *12*, 327–331. https://doi.org/https://doi.org/10.1016/0031-3203(80)90031-X.
60. Ma, Q.; Wu, J.; He, C.; Hu, G. Spatial scaling of urban impervious surfaces across evolving landscapes: From cities to urban regions. *Landscape and Urban Planning* **2018**, *175*, 50–61.
61. Sözen, A.; Arcaklıoğlu, E.; Özalp, M.; Çağlar, N. Forecasting based on neural network approach of solar potential in Turkey. *Renewable Energy* **2005**, *30*, 1075–1090.
62. Jiang, B.; Liu, X. Scaling of geographic space from the perspective of city and field blocks and using volunteered geographic information. *International Journal of Geographical Information Science* **2012**, *26*, 215–229.
63. Panakkat, A.; Adeli, H. Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators. *Computer-Aided Civil and Infrastructure Engineering* **2009**, *24*, 280–292.
64. Panakkat, A.; Adeli, H. Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators. *Computer-Aided Civil and Infrastructure Engineering* **2009**, *24*, 280–292.
65. Ouammi, A.; Zejli, D.; Dagdougui, H.; Benchrifa, R. Artificial neural network analysis of Moroccan solar potential. *Renewable and Sustainable Energy Reviews* **2012**, *16*, 4876–4889.
66. Barber, C.B.; Dobkin, D.P.; Huhdanpaa, H. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.* **1996**, *22*, 469–483. https://doi.org/10.1145/235815.235821.
67. Rezaei, M. Improving a Centroid-Based Clustering by Using Suitable Centroids from Another Clustering. *Journal of Classification* **2020**. https://doi.org/10.1007/s00357-018-9296-4.
68. OpenStreetMap Wiki contributors. Overpass API/Overpass QL. OpenStreetMap Wiki, 2023. Page name: Overpass API/Overpass QL, Date retrieved: 4 July 2023 12:04 UTC, Page Version ID: 2550700.
69. UN. World Urbanization Prospects, 2018. Accessed on October, 2023.
70. Gopalakrishnan, R.; Biswas, A.; Lightwala, A.; Vasudevan, S.; Dutta, P.; Tripathi, A. Demand prediction and placement optimization for electric vehicle charging stations. *arXiv preprint arXiv:1604.05472* **2016**.
71. Lam, A.Y.; Leung, Y.W.; Chu, X. Electric vehicle charging station placement. In Proceedings of the 2013 IEEE International Conference on Smart Grid Communications (SmartGridComm), 2013, pp. 510–515. https://doi.org/10.1109/SmartGridComm.2013.6688009.
72. Veneri, O.; Ferraro, L.; Capasso, C.; Iannuzzi, D. Charging infrastructures for EV: Overview of technologies and issues. In Proceedings of the 2012 Electrical Systems for Aircraft, Railway and Ship Propulsion, 2012, pp. 1–6. https://doi.org/10.1109/ESARS.2012.6387434.

73. Brandstätter, G.; Leitner, M.; Ljubić, I. Location of charging stations in electric car sharing systems. *Transportation Science* **2020**, *54*, 1408–1438.
74. Chen, T.D.; Kockelman, K.M.; Khan, M.; et al. The electric vehicle charging station location problem: a parking-based assignment method for Seattle. In Proceedings of the Transportation Research Board 92nd Annual Meeting, 2013, Vol. 340, pp. 13–1254.
75. Ge, S.; Feng, L.; Liu, H. The planning of electric vehicle charging station based on grid partition method. In Proceedings of the 2011 International Conference on Electrical and Control Engineering. IEEE, 2011, pp. 2726–2730.
76. Lam, A.Y.S.; Leung, Y.W.; Chu, X. Electric Vehicle Charging Station Placement: Formulation, Complexity, and Solutions. *IEEE Transactions on Smart Grid* **2014**, *5*, 2846–2856. https://doi.org/10.1109/TSG.2014.2344684.
77. Mohanty, A.K.; Babu, P.S. Optimal placement of electric vehicle charging stations using JAYA algorithm. In Proceedings of the Recent Advances in Power Systems: Select Proceedings of EPREC 2020. Springer, 2021, pp. 259–266.
78. Awasthi, A.; Venkitusamy, K.; Padmanaban, S.; Selvamuthukumaran, R.; Blaabjerg, F.; Singh, A.K. Optimal planning of electric vehicle charging station at the distribution system using hybrid optimization algorithm. *Energy* **2017**, *133*, 70–78.
79. Battapothula, G.; Yammani, C.; Maheswarapu, S. Multi-objective simultaneous optimal planning of electrical vehicle fast charging stations and DGs in distribution system. *Journal of Modern Power Systems and Clean Energy* **2019**, *7*, 923–934.
80. Sadeghi-Barzani, P.; Rajabi-Ghahnavieh, A.; Kazemi-Karegar, H. Optimal fast charging station placing and sizing. *Applied Energy* **2014**, *125*, 289–299. https://doi.org/https://doi.org/10.1016/j.apenergy.2014.03.077.
81. Rajabi-Ghahnavieh, A.; Sadeghi-Barzani, P. Optimal Zonal Fast-Charging Station Placement Considering Urban Traffic Circulation. *IEEE Transactions on Vehicular Technology* **2017**, *66*, 45–56. https://doi.org/10.1109/TVT.2016.2555083.