Efficient On-Street Parking Sensor Placement

Lukas Rottkamp
LMU Munich
Munich, Germany
lukas.rottkamp@campus.lmu.de

Matthias Schubert
LMU Munich
Munich, Germany
schubert@dbs.ifi.lmu.de

Niklas Strauß
LMU Munich
Munich, Germany
strauss@dbs.ifi.lmu.de

ABSTRACT

When placing sensors in an environment, it may not be possible to directly cover all entities of interest with sensors due to cost or other restraints. This leads to a sensor placement problem in which only a subset of all sensible sensor locations is equipped with sensors. If data concerning the system to be measured is already available or easily procured, sensor locations can be selected in a data-driven approach. Without data, alternative methods have to be applied. In this paper, we present and compare various data-driven and data-agnostic methods for selecting parking sensor locations in a city environment. Experiments using real-world data show that methods only requiring parking bays' locations compare reasonable well to data-driven approaches requiring environment data which may be expensive to acquire.

CCS CONCEPTS

• **Applied computing** → **Transportation**; *Forecasting*; • **Information systems** → **Spatial-temporal systems**; • **Computing methodologies** → *Model development and analysis.*

KEYWORDS

sensor placement, smart city, parking

ACM Reference Format:
Lukas Rottkamp, Matthias Schubert, and Niklas Strauß. 2022. Efficient On-Street Parking Sensor Placement. In *The 15th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS '22), November 1, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3557991.3567796

1 INTRODUCTION

Accurate data is important when making statements regarding the current or future state of a system such as an urban environment. Such data can be automatically recorded by sensors and processed according to the needs of relevant use-cases. For example, the City of Melbourne, Australia, fitted individual parking bays in its Central Business District with in-ground occupancy sensors to obtain real-time parking occupancy information. This naturally comes with costs for planning, installation and maintenance. Secondary factors such as obtaining necessary permits, privacy concerns, property restrictions or similar issues may complicate the installation of a complete sensor network. Consequently, outfitting only a subset of possible sensor locations with sensors may be preferred or even required. This comes with the disadvantage of having blind spots which needs to be addressed in order to still be able to feed suitable data to smart city applications. Although methods for filling gaps exist, they come with an error due to the inherent uncertainty caused by the environment's dynamic behavior, e.g. visitors or commuters parking their cars. In order to enable a good service quality, data providers seek to minimize such errors.

Selecting a subset of possible sensors can be done in a data-driven way by obtaining data from a particular environment and then, using this data to calculate optimal sensor positions. In the parking use case, this could be done by conducting a study in which parking occupancy is recorded by human observers during a short time-frame. As this also comes with drawbacks such as personnel costs and an increased project duration and complexity, strategies not requiring such data may be beneficial if their results are not much worse than data-driven methods. It is important to note that such strategies do not have to work completely without data: Data regarding the environment is usually available, such as a map of the street network including parking bays.

In this paper, we review data-driven and data-agnostic methods relevant for placing sensors in order to interpolate parking occupancy data. We state methods not needing initial occupancy data but only a street network graph including parking bay locations. We then evaluate various methods using real-world parking data and conclude that our method is a viable alternative when obtaining initial occupancy data is not possible or expensive.

2 RELATED WORK

Using information obtained through spatial sensor networks is a common theme in smart city applications. For example, it was shown that information about the current occupancy state of parking bays can be used to guide drivers to a free parking bay faster than approaches without such data [15]. Future parking occupancy can be predicted given data of sensor networks [2]. Methods for monitoring parking availability include in-ground sensor networks [19] and analysis of camera footage [16]. A comprehensive overview of parking bay sensors and use-cases making use of parking data is given in [6]. In case not every parking opportunity is directly covered by sensors, interpolation methods can be used to estimate parking occupancy at unobserved locations [3].

Approaches using stationary sensors described above assume an already specified and fixed sensor network, either by outfitting each parking bay in the target area with a sensor or making use of an existing sensor infrastructure. In contrast, this paper is concerned with selecting a limited number of street segments for sensor placement.

IWCTS '22, November 1, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9539-7/22/11…$15.00
https://doi.org/10.1145/3557991.3567796

Basic approaches for selecting a subset of possible sensor locations for sensor placement in a spatial environment include spreading the sensors randomly or uniformly over the area in question, e.g. by drawing a suitable grid and placing sensors near grid intersections. A more sophisticated method is to solve a coverage problem requiring that all areas or entities of interest are covered by at least one sensor, i.e., are within its detection range. This problem can be formalized as an optimization objective given coverage constraints and efficiently be solved using integer linear programming solvers. For example, [1] use a coverage approach to optimize placement of data relay nodes given a fixed set of parking sensors. Finding an exact solution may be computationally infeasible if the set of possible sensor combinations is large. In this case, heuristics such as simulated annealing or genetic algorithms can be applied [14].

Data-driven approaches need data of the type sensors would yield before the actual sensor placement is decided. These data may be collected by temporary sensors or a field study (e.g. personnel placed on streets with notepads). In some cases, it may be inferred from other data, although this adds another layer of uncertainty. Outfitting only a subset of possible sensor locations with sensors is known as the *sparse sensor placement optimization for reconstruction (SSPOR)* problem [11]. While work on SSPOR often uses spatial examples such as temperature interpolation, methods do not explicitly include spatial information. Instead sensors are placed only using sensor data. The PySensors toolkit for sensor placement presented by [4] contains algorithms to solve SSPOR: It takes a set of complete measurements and determines which components should be selected to reconstruct the remaining ones in an optimal way. The same problem is solved by Polire, an open source toolkit for spatial interpolation and sensor placement [9, 13]. It applies a greedy algorithm which selects sensors according to gains in a specified criterion, such as mutual information or entropy. The Chama framework for sensor placement covers many relevant algorithms and strategies for sensor placement but focuses on global event detection such as detecting an earthquake or a pollutant leaking into a system [7]. We on the other hand are not interested in detecting global events but detecting a multitude of individual parking events. [8] determine how a fixed budget may be best spent on individual observations when each observation is connected with a certain cost. They do not determine a fixed subset of sensors but determine which locations should best be queried at which time. In contrast, we are interested in determining a permanent network of stationary sensors.

3 PROBLEM DEFINITION

A spatial sensor selection problem includes entities of which state information should be collected by sensors. Multiple problem definitions are possible depending on the nature of this entity. In our definition, we consider n discrete entities p_i, e.g. each being the set of individual parking bays of a given street segment. This covers the use-case of recording on-street parking availability with in-ground devices as the infrastructure needed for a street segment (such as a relay node processing raw data of individual sensors and transmitting occupancy data to a central database) can be use by multiple sensors connected by wires or near-field communication [1]. In this case, we call the combined set of devices monitoring exactly one street segment a sensor. It also covers the use-case of recording such data by analyzing camera images, as a camera can typically be installed in a way that covers all parking bays of a street segment, but not multiple street segments at once.

Note that in our definition, a street segment never contains an intersection. Further, the link between two intersections is divided into multiple street segments if it would otherwise contain too many parking bays. We call a street segment connected to an entity a *candidate segment* as it is a candidate for sensor placement. Sensors are never placed on street segments that are not candidate segments.

Each entity is connected to a measurable value $v_{i,t}$ ($0 \leq v_{i,t} \leq 1$) for each time t, e.g. the fraction of occupied parking bays at time t. The spatial relation of p_i is given as their location in a graph G with nodes N and edges E. Each p_i is connected to an edge $e_i \in E$, e.g. a street segment. Each node carries location information in form of Cartesian coordinates. Therefore, each entity p_i can be assigned a location l_i that is defined as the midpoint between both nodes of its edge e_i.

As stated above, each entity p_i is monitored by exactly one sensor. Sensor presence is indicated by an indicator variable $s_i \in \{0, 1\}$ which is set to 1 if a sensor is present at p_i, otherwise 0. Each of m sensors placed covers exactly one entity, i.e. $\sum_{1 \leq i \leq n} s_i = m$.

The sensor selection problem now selects the set of entities $\hat{\sigma}$ to be equipped with a sensor that maximizes an objective function λ over all valid sensor subsets $\sigma_k \in \Omega$:

$$\hat{\sigma} = \underset{\sigma_k \in \Omega}{\operatorname{argmax}} \lambda(\sigma_k) \tag{1}$$

with

$$\Omega = \{(s_1, ..., s_n) \forall s_i \in \{0, 1\}\}, \sum_{1 \leq i \leq n} s_i = m \tag{2}$$

Reasonable functions for λ include averaged interpolation or prediction errors when using the subset to reconstruct actual values $v_{i,t}$. Assuming a predictor $\Gamma(\sigma, i)$ taking a subset of sensors to predict the value of the i-th sensor, the mean absolute error (MAE) may be used:

$$\lambda_{\text{MAE}}(\sigma) = \sum_{0 \leq i \leq n, \forall t} |v_{i,t} - \Gamma(\sigma, i)| \tag{3}$$

Note that the best subset depends on both predictor Γ and objective function λ. These have to be chosen according to the use-case.

4 SENSOR PLACEMENT METHODS

Various methods for sensor placement are mentioned in Section 2 above. These include data-driven and data-agnostic approaches. Data-driven approaches use observation data for placement of sensors. In our parking use-case, this is parking occupancy data. Data-agnostic methods don't use such data but may use *metadata* such as locations of candidate segments and the street graph. Some methods are deterministic while others involve random components such as random initialization or random tie-breaks. Each method is given a input parameter m denoting the exact size of the target subset of candidate segments to select for sensor placement, i.e. the number of sensors to place.

We now describe a number of methods which are relevant for our parking use-case and included in our evaluation.

4.1 Data-agnostic placement methods

Simple placement methods *Random* and *Largest* are included as benchmark methods for comparison. We devised methods *Clusters* and *MaxMin* to exploit the observation that pairwise correlation between spatial resources depends on the distance between them: A smaller distance tends to coincide with higher similarity. This may be due to their shared neighborhood with points of interest targeted by drivers. Other reasons may include differing parking rules or peculiarities of the street network such as dead-end streets or especially busy areas. While we are certainly not the first to use the underlying algorithms, we are not aware of other attempts of using them in related problem settings. Method *Coverage* is included for comparison as coverage-based methods are routinely used for sensor placement [1, 18]. Note that we don't use it in the "traditional" way of optimizing *direct* sensor coverage as explained below.

4.1.1 Random. Candidate segments are selected for sensor placement by random draw. Each segment has the same probability of being drawn.

4.1.2 Largest. Candidate segments are selected only by their respective number of individual parking bays contained. Segments with higher counts are selected first. This method is explicitly included as a "higher-bound" benchmark as we have no reason to believe that it leads to advantageous selections.

4.1.3 Clusters. This method is shown in Algorithm 1: To select n sensors, segments are first clustered into n clusters through K-Means clustering using Lloyd's algorithm [10]. Each cluster's center point is calculated as the mean location of all candidate segments it contains. The nearest not previously selected candidate segment to each center point is determined and selected for sensor placement. Note that the algorithm uses locations given in a local Cartesian coordinate system. Locations denoted in geographic coordinates are projected to a suitable local Cartesian coordinate system first. The method is not deterministic as results of Lloyd's algorithm depend on its random initialization and ties in distance are resolved randomly.

4.1.4 MaxMin. This method selects candidate segments so that the minimum pairwise graph distance over all selected sensors is maximized. This effectively spreads the sensors as widely as possible while preventing sensors to be near to each other. The large number of possible subsets prevents us from obtaining an optimal solution due to the high computational complexity. Instead, we use a greedy heuristic shown in Algorithm 2. We restart this algorithms multiple times, always keeping the best result seen so far, to minimize the risk of ending up in local optimum worse than the global optimum.

4.1.5 Coverage. This method also exploits the spatial relationship of parking segments. Here, a candidate segment is defined to be covered if at least one sensor is present within a certain graph distance d. Note that "coverage" in this sense does not refer to direct coverage through actual observance by the sensor but indirect coverage due to a statistically higher likeness of occupancy because of spatial closeness. The method places m sensors so that d is minimized under the constraint that each candidate segment is covered.

```
Data: P                          ▷ Set of all possible sensor locations
Data: k ≥ 1                      ▷ Amount of sensors to select
clusterCenters ← clusterKMeans(P, k);
bestSubset ← {};
foreach center ∈ clusterCenters do
    minDist ← inf;
    foreach c ∈ P \ bestSubset do
        d ← dist(center, c);
        if d < minDist then
            minDist ← d;
            bestCandidate ← c;
        end
    end
    bestSubset ← bestSubset ∪ {bestCandidate};
end
return bestSubset;
```

Algorithm 1: Part of selection method Clusters.

```
Data: P                          ▷ Set of all possible sensor locations
Data: k ≥ 1                      ▷ Amount of sensors to select
bestSubset ← pickOneRandomly(subsetsOfSize(P, k));
maxMinDist ← minPairwiseDist(bestSubset);
repeat
    improved ← False;
    S ← bestSubset;
    foreach T ∈ subsetsOfSize(S, k − 1) do
        foreach c ∈ P \ S do
            U ← T ∪ {c};
            d ← minPairwiseDist(U);
            if d > maxMinDist then
                maxMinDist ← d;
                bestSubset ← U;
                improved ← True;
            end
        end
    end
until not improved;
return bestSubset;
```

Algorithm 2: Part of selection method MaxMin.

This can efficiently be done by pre-computing solutions to the the set-cover problem [12] for a each distance d in a suitable range. Each solution of the set-cover problem is a minimal set of sensors so that each candidate segment is covered assuming a sensor range of d. An optimal set of m sensors can then be looked up in the pre-computed solution list by selecting the solution with lowest distance under the condition that m sensors are selected.

Minimal set sizes for our evaluation environment Melbourne are shown in Figure 1. For example, the installation of 50 sensors can be done in a way so that no parking segment is more than about 300 meters away from a sensor.

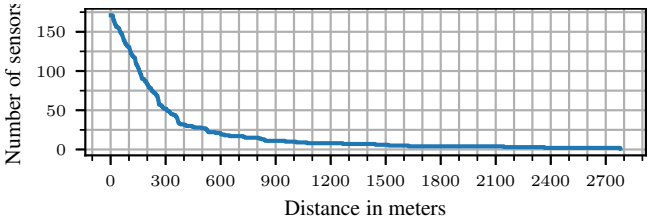


Figure 1: Minimum amount of sensors needed to cover all parking segments for various coverage distances in our evaluation environment.

4.2 Data-driven placement methods

Data-driven approaches are given observation data of all candidate segments. This means they can effectively select the best sensor sets in hindsight. They are connected to higher costs in real-world applications in case such data needs to be obtained first. The following methods are included:

4.2.1 MinError. This method places sensors in a way that minimizes the evaluation metric given occupancy data. Because the computational complexity prevents us from finding the optimal solution for larger experiments such as ours, we implemented a greedy search heuristic: Starting with an empty set, sensors are added one-by-one until the targeted number of sensors is reached. The sensors added in each step is the one which, compared to all other remaining candidates, leads to the smallest total error. While this heuristic only approximates the optimal selection, it produces results very close to the optimal solution in smaller experiments and we are confident it is a good benchmark method.

4.2.2 PySensors. The PySensors [4] framework is included using the original implementation[1]. It includes three choices of basis functions, *Identity*, *SVD* and *Random*, each requiring parameters. For our evaluation, we determined the best basis and parameters experimentally in preliminary experiments. It should be noted that PySensors makes no use of segments' locations nor the street graph but is a pure data-driven selection method working on occupancy data.

4.2.3 Polire. Like *PySensors*, the Polire framework [13] does not consider location information but only the supplied occupancy time series. We include it in our evaluation via the original implementation[2]. Users can choose between stationary or non-stationary models, different kernels and parameters. Those were again determined beforehand in preliminary experiments.

5 INTERPOLATION METHODS

Having selected a subset of segments for sensor placement, predictor Γ (see Section 3) infers the states of remaining parking segments from data obtained through those sensors. We use spatial interpolation techniques for this task as parking bays are spatially related. Two interpolation methods are used in our evaluation:

[1]https://github.com/dynamicslab/pysensors
[2]https://github.com/sustainability-lab/polire/blob/SenSys20_Poster/polire/placement/base/base.py

The **KNN** (k nearest neighbors) method as shown in Equation 4 is employed to calculate parking availability A at location x given the set R_k of the k nearest parking bays and their respective availability values A_i measured by sensors. In our evaluation, we set the parameter k to 5 as this value gave lowest interpolation errors in preliminary experiments.

$$A(x) = \frac{\sum_{i \in R_k} A_i}{k} \tag{4}$$

We also include the **IDW** (inverse distance weighting) method [17] shown in Equation 5. Here, availability A at location x is calculated by a weighted average over all available sensor values A_i. The weights $w_i(x)$ depend on the distance between location and sensor raised to the power of α. We set α to 2 as preliminary experiments gave good interpolation performance using this value.

$$A(x) = \frac{\sum_{i \in R} w_i(x) A_i}{\sum_{i \in R} w_i(x)}, w_i(x) = \frac{1}{d(x, x_i)^\alpha} \tag{5}$$

According to [3], IDW is well-suited for estimating parking availability given a limited number of sensors. This method seems especially beneficial as the information given by a sensor in spatial settings like ours is expected to statistically decrease with distance.

6 EVALUATION

Evaluation follows the two-step process described in Section 3. First, a subset of parking segments is selected according to the respective sensor placement method. Occupancy values are then "virtually" measured by those sensors according to ground truth data. These values are then used to obtain all remaining parking segments' occupancy values using an interpolation method. Finally, the interpolation error is calculated using ground truth data. Non-deterministic methods were evaluated multiple (7) times to reduce influence of random effects.

Two experiments have been conducted to gain insights into the performance of methods: The first scenario assumes complete availability of observation data. This enables us to evaluate models' best-case performance and gain insights in the general task of interpolating parking occupancy. A low error in this scenario is however not alone representative of a method's real-world capability as there is no need to place sensors if all data is be available anyway. The interpolation error on previously unseen data is also a crucial metric. Therefore, we include a second scenario which only presents a fraction of data to selection algorithms. In practice, this data may be acquired through a field study. Interpolation errors are then calculated over a test set consisting of a time span not included in the initial training data.

6.1 Evaluation dataset

Real-world parking occupancy ground truth data for our evaluation is taken from the City of Melbourne, Australia, open data platform[3]. This platform provides the dataset *On-street Parking Bays*[4] containing locations of on-street parking bays in the Central Business

[3]City of Melbourne open data platform: https://data.melbourne.vic.gov.au; Data licensed under *Creative Commons Attribution 3.0 Australia*: https://creativecommons.org/licenses/by/3.0/au/
[4]https://data.melbourne.vic.gov.au/Transport-Movement/On-street-Parking-Bays/crvt-b4kt

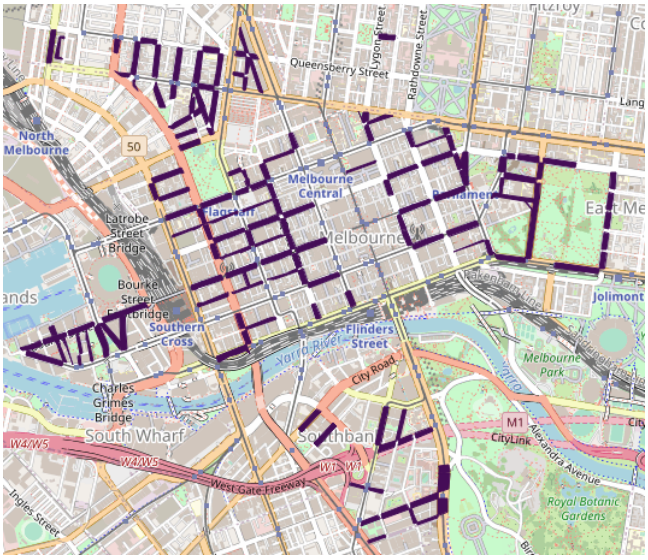


Figure 2: On-Street parking segments used for evaluation.

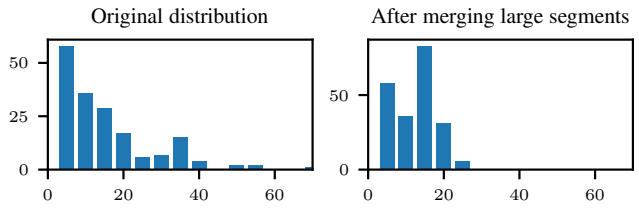


Figure 3: Histogram of parking segment sizes before and after processing.

District of Melbourne. We connected these parking bays to a street graph obtained through OpenStreetMap[5]. Parking bays for loading only and bays requiring special permissions were removed.

Individual on-street parking occupancy data of the same parking bays is available in dataset *On-street Car Parking Sensor Data - 2017*[6]. In this dataset, arrivals and departures of vehicles are recorded to the second. We resampled the occupancy into 5-minute time slots, each of which represents a state of the overall parking situation. Individual parking bays were then aggregated according to their street segments, a street segment being defined a segment of a street between two intersections. Segments with more than 30 parking bays were split into multiple smaller ones then amounting to sizes of between 15 and 20 bays. This was done as exceptionally large segments (usually caused by a street with few intersections) may not be covered by a single sensor such as a camera sensor. The resulting parking segment size histogram is presented in Figure 3. After processing, we obtain 189 parking street segments.

The following datasets were included for evaluation:

[5]https://www.openstreetmap.org
[6]https://data.melbourne.vic.gov.au/Transport/On-street-Car-Parking-Sensor-Data-2017/u9sa-j86i

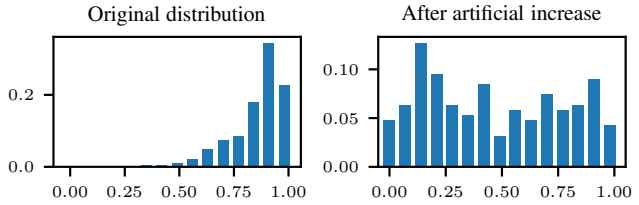


Figure 4: Distribution of parking bays having at least one free bay, before and after artificial increase.

6.1.1 Regression. This dataset contains each parking street segment's occupancy for each 5-minute time slot. These values are calculated as average occupancy values over all individual parking bays of respective segments, giving a numeric target variable between 0 and 1.

6.1.2 ThreeBins. This dataset represents a classification problem as numeric values of dataset *Regression* are sorted into the three bins *low occupancy*, *medium occupancy* and *high occupancy*. It is motivated by existing parking occupancy tools indicating estimated availability using traffic light colors. This is easier to communicate to end users than numeric values. Further, numeric values may suggest a sense of accuracy that may be difficult to achieve when predicting parking accuracy. On the other hand, a more coarse classification such as this one is satisfactory for a user looking for parking opportunities.

6.1.3 OneFreeHigh. In this dataset, the target variable is binary and states if at least one individual parking bay of the parking street segment in question is free at a given point in time. This is motivated by the fact that in practice, drivers searching for a parking opportunity in their immediate surroundings are primarily interested in streets containing at least one free parking bay. It should be noted that according to the Melbourne in-ground sensor data, most parking segments contain at least one free parking bay at a given time. Naturally, a parking information system is most appreciated when finding a free parking opportunity presents a challenge to motorists, i.e. when parking opportunities are rare. To evaluate the various methods in such a setting, we artificially increased the parking occupancy of all parking bays so that only about every second parking segment contains at least one free parking bay. This means an increase of parking demand by 38%. A comparison of the resulting increased occupancy distribution versus original occupancy distribution is shown in Figure 4.

6.2 Experiment 1: Training on complete data

Experiment 1 covers six months, from June to November 2017. Data-driven methods will receive the complete ground-truth occupancy data. As described above, this is unrealistic in practice but yields insights about the best-case performance of placement methods. No-data methods make no use of occupancy data.

Each selection method is executed multiple times: The respective number of sensors to select increases from only one sensor to 171 sensors (out of 189 possible sensors). Interpolation errors for each method and parameter are then calculated as described above.

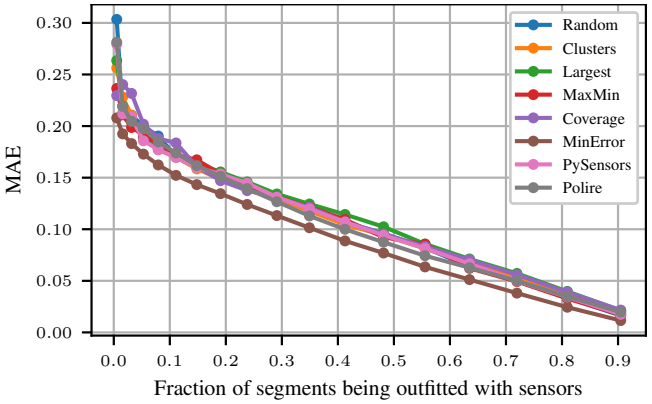
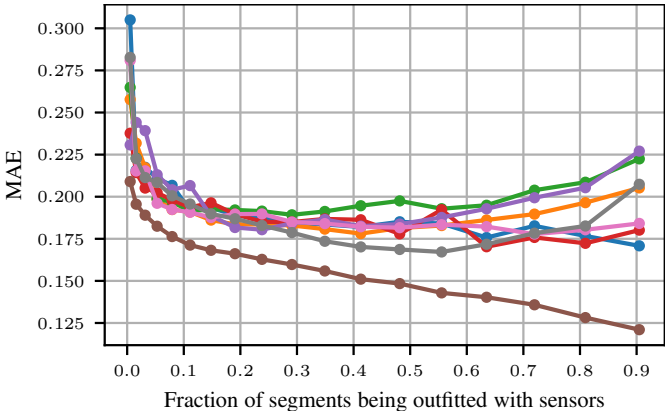


Figure 5: Mean absolute interpolation errors (over all parking segments) by size of sensor subset on *Regression* dataset. Each line represents a selection method.

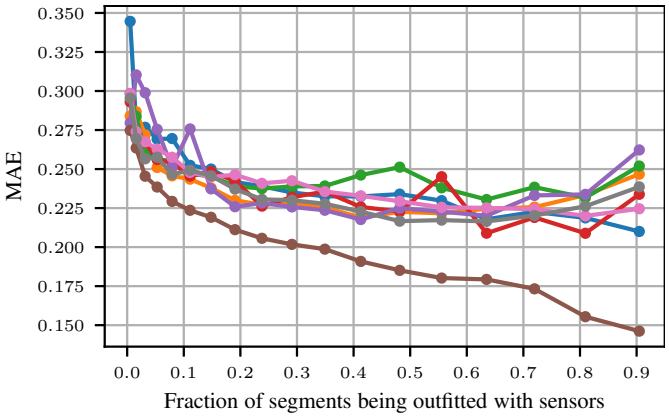
6.2.1 Comparing results for different methods. The mean absolute IDW interpolation errors of evaluated methods given varying numbers of segments to select are shown in Figure 5. For non-deterministic methods, multiple repetitions were run whose errors are averaged. Not surprisingly, errors decrease with growing amount of sensors. Our "lower-bound" benchmark method *MinError* clearly produces the smallest error as expected given its direct optimization of the IDW error. Other methods are not easily distinguished in this overview.

For a better evaluation, it is appropriate to exclude candidate segments fitted with sensors from evaluation, as these naturally show no error. We will focus on these error values in the remainder. The are shown in Figure 6a for the *Regression* dataset. Our earlier observations appear in more detail. Method *Polire* shows second-best performance while *Random* appears to be average. Errors for datasets *ThreeBins* and *OneFreeHigh* are shown in Figure 6a and Figure 6c, respectively. The ranking of methods is similar to the one observed for the *Regression* dataset. A notable exception is the error curve of method *Clusters* when evaluated using dataset *OneFreeHigh*: It is now intersecting with *Polire* multiple times. This is especially remarkable given their different nature, as *Clusters* does not need occupancy data but only parking bays' locations.

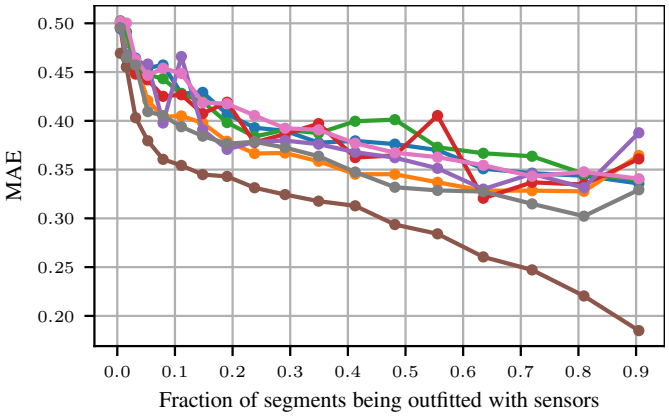
A condensed comparison of errors is shown in Table 1. Relative errors compared to *MinError*, the best method for each dataset, are included. *MinError* being the overall best method is not surprising as it by design directly optimizes the evaluation metric. Second-best is *Polire*, also requiring complete data. Method *Clusters* reaches third place even though it doesn't use occupancy data. Sensors selected by method *PySensors* produces slightly higher errors than some data-agnostic methods. It can be seen that methods' ranks are often equal for both interpolation methods. Selection method *Random* ranks in the bottom half of results. However, error differences are relatively small in some cases. We attribute this to the fact that random placement tends to cover the whole area which is beneficial for interpolation as spatial resources such as parking bays are typically locally correlated. Further, if random placement considers not just an area but the set of possible sensor locations



(a) Error using *Regression* dataset.



(b) Error using *ThreeBins* dataset.



(c) Error using *OneFreeHigh* dataset.

Figure 6: Mean absolute interpolation errors (only unselected parking segments) by size of sensor subset. Each line represents a selection method (color definition in Figure 5).

as in the setting of this paper, it implicitly draws from a density distribution of entities to be measured, potentially reducing the interpolation error even more.

Method	Regression	ThreeBins	OneFreeHigh	Mean
MinError	.161 (+0%)	.207 (+0%)	.327 (+0%)	+0.0%
Polire	.193 (+19%)	.239 (+15%)	.377 (+15%)	+16.9%
Clusters*	.196 (+21%)	.240 (+16%)	.384 (+17%)	+18.4%
MaxMin*	.192 (+18%)	.241 (+16%)	.398 (+21%)	+19.0%
PySensors	.194 (+20%)	.244 (+18%)	.407 (+24%)	+21.1%
Random*	.197 (+21%)	.247 (+19%)	.404 (+23%)	+21.8%
Coverage*	.202 (+25%)	.247 (+19%)	.396 (+20%)	+21.9%
Largest*	.203 (+25%)	.249 (+20%)	.408 (+24%)	+23.7%

Table 1: Mean IDW interpolation MAE values of selection methods over all subset sizes. Relative difference to best in column is shown in brackets, with mean in right column. Methods not using parking data are marked with asterisks.

Method	Regression	ThreeBins	OneFreeHigh	Mean
MinError	.174 (+0%)	.226 (+0%)	.384 (+0%)	+0.0%
Polire	.198 (+13%)	.246 (+8%)	.404 (+5%)	+9.3%
Clusters*	.194 (+11%)	.246 (+8%)	.414 (+7%)	+9.3%
Coverage*	.194 (+11%)	.244 (+7%)	.421 (+9%)	+9.6%
MaxMin*	.194 (+11%)	.247 (+9%)	.424 (+10%)	+10.2%
PySensors	.197 (+13%)	.251 (+10%)	.429 (+11%)	+11.8%
Random*	.199 (+14%)	.252 (+11%)	.428 (+11%)	+12.3%
Largest*	.205 (+17%)	.254 (+12%)	.429 (+11%)	+13.8%

Table 2: Mean KNN interpolation MAE values of selection methods over all subset sizes. Relative difference to best in column is shown in brackets, with mean in right column. Methods not using parking data are marked with asterisks.

Method	Regression	ThreeBins	OneFreeHigh
IDW	.196	.244	.398
KNN	.197	.249	.423

Table 3: Mean interpolation errors of interpolation methods IDW and KNN for evaluated datasets.

6.2.2 Comparison of interpolation methods. Table 2 shows results for KNN interpolation errors in contrast to the IDW interpolation errors discussed above. Method *MinError*'s error is much higher than before, lowering the relative difference to other methods. This is expected as *MinError* directly minimizes the IDW error but now interpolation uses the KNN method. The ranking of methods is almost the same as before, suggesting that IDW and KNN produce similar estimates.

Aggregated values for interpolation methods are shown in Table 3. Note that these exclude method *MinError* due to its minimization of IDW error as this would skew results towards IDW. Still it can be seen that IDW interpolation yields slightly better overall results. This is not surprising as IDW weights nearer sensor values higher than those farther away which exploits the environment's spatial correlation motivated above. Still differences are very small.

Method	Regression	ThreeBins	OneFreeHigh
Random	.0093	.0102	.0101
Clusters	.0015	.0021	.0041
PySensors	.0041	.0047	.0101

Table 4: Mean standard deviation of error for evaluated methods and datasets. For each evaluated subset size, the standard deviation was calculated over all repetitions. Averages over all subset sizes are shown here.

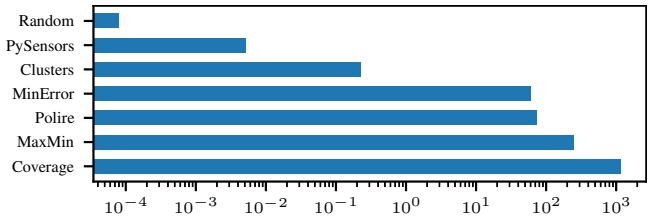


Figure 7: Mean CPU time (in seconds) per selection run during Experiment 1.

6.2.3 Variance analysis. Methods *Random*, *Clusters* and *PySensors* are non-deterministic as they contain random choices at some point. To average out random effects, evaluation results discussed above have been averaged over a number of runs. It is important to also analyze the errors' variance as a high variance method may lead to disappointing results when applied in a real-world setting without ground-truth data. Consistently high variance would also question the reliability of the evaluation procedure itself. The mean standard deviation of individual methods is shown in Table 4: Results are low compared to the absolute errors in Table 1. Values for method *Clusters* are significantly smaller than those of method *Random*, indicating a stable clustering. The error variability of method *PySensors* is caused by its use of randomized SVD solver [5]. Its magnitude is again small compared to absolute error values. Overall, the low variance increases confidence in the applicability of evaluated methods.

6.2.4 Comparison of selection time. Figure 7 shows the CPU time needed per selection run during evaluation using an Intel® Core™ i7-10750H CPU. This comparison should be taken with a grain of salt as implementations of methods are not primarily optimized for best performance. Still the range over multiple orders of magnitude is something to consider when applying methods in larger environments than ours. Generally, the runtime of selection methods is of lesser importance as they are only run during the planning phase before sensors are being installed. For interpolation between measurements of installed sensors, fast implementations of IDW and KNN methods exist.

6.3 Experiment 2: Limited training set

This experiment covers the scenario of conducting a field study to obtain data for data-driven placement methods. As Melbourne ground truth data is readily available in the datasets described above, we conduct a "virtual" field study by extracting a time span

Method	Regression	ThreeBins	OneFreeHigh	Mean
MinError	.165 (+0%)	.213 (+0%)	.333 (+0%)	+0.0%
Polire	.191 (+15%)	.234 (+9%)	.379 (+13%)	+13.1%
Clusters*	.195 (+18%)	.239 (+12%)	.385 (+15%)	+15.4%
MaxMin*	.190 (+15%)	.240 (+12%)	.399 (+19%)	+16.1%
PySensors	.194 (+17%)	.242 (+13%)	.407 (+22%)	+17.9%
Random*	.195 (+18%)	.246 (+15%)	.405 (+21%)	+18.4%
Coverage*	.201 (+21%)	.246 (+15%)	.397 (+19%)	+18.8%
Largest*	.201 (+22%)	.249 (+16%)	.409 (+22%)	+20.6%

Table 5: Experiment 2: Mean IDW interpolation MAE values of selection methods over all subset sizes. Relative difference to best in column is shown in brackets, with mean in right column. Methods not using parking data are marked with asterisks.

Method	Regression	ThreeBins	OneFreeHigh
MinError	3.240%	7.092%	4.731%
Polire	0.623%	0.996%	0.034%
PySensors	0.097%	1.210%	0.886%

Table 6: Mean IDW-interpolation MAE increases from training to testing error of selection methods over all subset sizes.

of training data from these datasets. We chose a span of four consecutive weeks for training as shorter spans resulted in large training set error variability due to the limited number of samples. A second extract of the following four months is then used as test data on which sensor selections are evaluated.

6.3.1 Comparing results for different methods. A comparison of test errors for evaluated placement methods can be seen in Table 5. They closely resemble the results of Experiment 1 shown in Table 1. This is no surprise for data-agnostic methods as their selections do not depend on the training data and all test data of Experiment 2 is also included in Experiment 1. Data-driven methods on the other hand are now working on substantially reduced training data (four weeks instead of six month during Experiment 1) which however did not effect their performance. This indicates that four weeks of training data are sufficient for data-driven selection methods.

6.3.2 Comparing training and testing errors. Table 6 shows relative increases of errors when comparing test dataset errors with training errors. Method *MinMax* shows largest increases, probably due to overfitting as it directly optimizes the error metric during training. Methods *Polire* and *PySensors* show smaller deviations. This may indicate that they internally created more robust models which generalize better than the aggressive method of *MinMax*. Generally, the moderate increase indicates that good selections during training are still good during later time spans. This is an important insight as it confirms our initial proposition that placing sensors at only a subset of parking street segments is a viable strategy.

7 CONCLUSION

In this paper, we described and compared various sensor placement methods. Our evaluation using real-world parking data shows that data-driven placement methods lead to slightly lower interpolation errors than data-agnostic methods not receiving such data but only metadata such as locations of on-street parking bays. Data-driven methods however require data typically obtained through preliminary surveys or installation of temporary sensors which may be expensive and time-consuming.

We conclude that data-agnostic methods are a reasonable alternative if suitable data is not readily available. Especially our proposed cluster-based method appears to be a good choice in such cases.

REFERENCES

[1] Antoine Bagula, Lorenzo Castelli, and Marco Zennaro. 2015. On the design of smart parking networks in the smart cities: An optimal sensor placement model. *Sensors* 15, 7 (2015), 15443–15467.

[2] Fabian Bock, Sergio Di Martino, and Antonio Origlia. 2017. A 2-step approach to improve data-driven parking availability predictions. In *Proceedings of the 10th ACM SIGSPATIAL workshop on computational transportation science*. 13–18.

[3] Fabian Bock and Monika Sester. 2016. Improving parking availability maps using information from nearby roads. *Transportation Research Procedia* 19 (2016), 207–214.

[4] Brian M de Silva, Krithika Manohar, Emily Clark, Bingni W Brunton, Steven L Brunton, and J Nathan Kutz. 2021. PySensors: A Python package for sparse sensor placement. *arXiv preprint arXiv:2102.13476* (2021).

[5] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2009. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. (2009).

[6] MY Idna Idris, YY Leng, EM Tamil, NM Noor, Z Razak, et al. 2009. Car park system: A review of smart parking system and its technology. *Information Technology Journal* 8, 2 (2009), 101–113.

[7] Katherine A Klise, Bethany L Nicholson, and Carl Damon Laird. 2017. *Sensor placement optimization using Chama.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[8] Andreas Krause, Eric Horvitz, Aman Kansal, and Feng Zhao. 2008. Toward community sensing. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 481–492.

[9] Andreas Krause, Ajit Singh, and Carlos Guestrin. 2008. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research* 9, 2 (2008).

[10] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[11] Krithika Manohar, Bingni W Brunton, J Nathan Kutz, and Steven L Brunton. 2018. Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems Magazine* 38, 3 (2018), 63–86.

[12] Seapahn Meguerdichian and Miodrag Potkonjak. 2003. *Low power 0/1 coverage and scheduling techniques in sensor networks.* Technical Report. Citeseer.

[13] S Deepak Narayanan, Zeel B Patel, Apoorv Agnihotri, and Nipun Batra. 2020. A toolkit for spatial interpolation and sensor placement. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 653–654.

[14] Sharon L Padula and Rex K Kincaid. 1999. *Optimization strategies for sensor and actuator placement.* Technical Report.

[15] Lukas Rottkamp and Matthias Schubert. 2020. Quantifying the potential of data-driven mobility support systems. In *Proceedings of the 13th ACM SIGSPATIAL International Workshop on Computational Transportation Science*. 1–10.

[16] Xavier Sevillano, Elena Màrmol, and Virginia Fernandez-Arguedas. 2014. Towards smart traffic management systems: Vacant on-street parking spot detection based on video analytics. In *17th International Conference on Information Fusion (FUSION)*. IEEE, 1–8.

[17] Donald Shepard. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*. 517–524.

[18] Chenxi Sun, Victor OK Li, Jacqueline CK Lam, and Ian Leslie. 2019. Optimal citizen-centric sensor placement for air quality monitoring: a case study of city of Cambridge, the United Kingdom. *IEEE Access* 7 (2019), 47390–47400.

[19] Carol Zimmerman, Rachel Klein, Jeremy Schroeder, Katie Turnbull, Kevin Balke, Mark Burris, Emily Saunoi-Sandgren, Elliot Martin, Susan Shaheen, Caroline Rodier, et al. 2014. *San Francisco urban partnership agreement: national evaluation report.* Technical Report. United States. Department of Transportation. Intelligent Transportation Systems Joint Program Office.