A-GWR: Fast and Accurate Geospatial Inference via Augmented Geographically Weighted Regression

Mohammad Reza Shahneh
mohammadreza.zareshahneh@email.ucr.edu
University of California, Riverside
Riverside, USA

Samet Oymak*§
oymak@ece.ucr.edu
University of California, Riverside
Riverside, USA

Amr Magdy*§
amr@cs.ucr.edu
University of California, Riverside
Riverside, USA

Abstract

Geographically Weighted Regression (GWR) is a seminal technique with rich applications in geospatial data analysis. However, it has critical drawbacks in the age of big data in terms of expressiveness, i.e., predictive power, and scalability. This work proposes Augmented GWR (A-GWR) that alleviates these drawbacks. A-GWR adapts a novel technique, Stateless-MGWR or S-MGWR, that enriches the predictive power by allowing different training data features to influence at different spatial scales. S-MGWR uses a customized black-box optimization approach for discovering optimal parameters in a fast and efficient way. In addition, A-GWR modularly combines S-MGWR with versatile models such as random forest models. Moreover, A-GWR enables scalability by operating on partitioned data to adapt to tight computational budgets. Our extensive experiments on various real and synthetic datasets demonstrate the scalability and accuracy benefits of the proposed techniques over state-of-the-art competitors.

CCS Concepts

• **Computing methodologies** → *Machine learning algorithms*; • **Information systems** → *Information systems applications.*

Keywords

multiscale Geographically Weighted Regression, MGWR, GWR, Regression, Machine Learning, Black-box Optimization, Ensemble Learning, Modular Models

ACM Reference Format:
Mohammad Reza Shahneh, Samet Oymak, and Amr Magdy. 2021. A-GWR: Fast and Accurate Geospatial Inference via Augmented Geographically Weighted Regression. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21), November 2–5, 2021, Beijing, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3474717.3484260

*Equal advising.
§This work is partially supported by MURI Grant W911NF-21-1-0312 by the Army Research Office, the National Science Foundation, USA, under grants CCF-2046816, IIS-1849971, SES-1831615, and CNS-2031418.

SIGSPATIAL '21, November 2–5, 2021, Beijing, China

ACM ISBN 978-1-4503-8664-7/21/11…$15.00
https://doi.org/10.1145/3474717.3484260

1 Introduction

Regression is one of the most traditional and popular learning models in an excessive number of applications. In spatial applications, regression models of different types are used for various use cases [2, 3, 5, 12, 14, 27, 28, 34, 35, 39]. Geographically Weighted Regression (GWR) [11] is one of the most popular spatially-varying coefficient (SVC) models that are used to capture spatial non-stationarity in spatial data. It has been widely adopted in many areas such as climate science [2], health and health care [28], criminology [39], transportation analysis [3], house price modeling [12], data analysis and visualization [14]. GWR and its variations specifically are popular and have been used recently for COVID-19 pandemic data to analyze the spatio-temporal effects of its driving factors to limit its spread worldwide [27]. SVC models are powerful alternatives to most traditional methods of modeling data when dealing with spatial data. In traditional methods, such as ordinary least squares, the role of location is often neglected or is not properly captured. Despite its efficacy, GWR has two major limitations: expressiveness and runtime scalability. These limitations restrict utilizing GWR for modern applications that involve diverse features and thousands to millions of data points. We outline both limitations as follows:

(1) Expressiveness: GWR uses a single number, known as bandwidth, to control the contribution of neighboring data around each location. This bandwidth plays the role of the spatial influence parameter, so a point in a location L is only affected by other points within L's bandwidth but not beyond. GWR uses one bandwidth for all features, which enforces the same spatial scale for the impact of different features and reduces the expressive power of the model. In reality, different features potentially have effects at different spatial scales. For example, a feature representing *wind speed* might be spatially correlated at a state level, while another feature that expresses *temperature* is spatially correlated at a county level. Multiple works [20, 26, 40] attempted to overcome this issue by providing a vector of bandwidths , assigning a control knob to each of the features. One of the recent notable advancements is Multiscale GWR (MGWR) [13]. MGWR assigns different bandwidths for different features, enabling each parameter surface to operate on a different locality. MGWR uses the backfitting algorithm for calibration involving many uni-variate GWR calibrations in each step which results in considerably higher runtime than regular GWR. In addition, due to the nature of backfitting, it is not possible to produce the same results only using the learned bandwidth; the complete history of bandwidths throughout the process is required.

Besides the limited expressiveness of using a single bandwidth, GWR also ignores nonlinear feature interactions beyond location. To overcome this, it is important to augment GWR with other powerful supervised learning models, such as random forests and

neural networks. These models can effectively capture nonlinear feature interactions and have shown superiority in various modern applications. While there have been attempts to combine GWR with other learning methods [22], previous works are often optimized for a single problem. They do not provide a general method of using other learning models with GWR.

(2) Scalability is arguably a core bottleneck of the GWR, as training a GWR model has a quadratic time complexity in terms of the training data size. Such quadratic growth implies that scalability beyond a few tens of thousands of examples becomes highly challenging. This situation is exacerbated with MGWR-type approaches, which require many iterations of GWR subproblems to optimize bandwidths. For modern datasets, it is critical to develop resource-aware algorithms to adapt to the computational constraints and utilize resources efficiently. Even though there have been some works on reducing the runtime of GWR [24] and MGWR [25], it is still a major runtime bottleneck to handle large datasets.

Contributions: Towards addressing these challenges, this paper proposes Augmented Geographically Weighted Regression (A-GWR), a novel and scalable framework for applying regression models to spatial data. A-GWR provides scalability for large spatial datasets and benefits from the advantages of modern supervised learning techniques to boost both accuracy and runtime efficiency. A-GWR goes three steps beyond the existing GWR-related literature. First, it innovates a stateless version of MGWR [13], called Stateless-MGWR (S-MGWR). S-MGWR eliminates the need for the full history of bandwidth values to optimize the bandwidth of different features. This allows S-MGWR to use various out-of-the-box parameter optimization methods and enables faster, more flexible, and accurate parameter tuning. Second, A-GWR augments S-MGWR with a general-purpose model, e.g., random forest model, to bring the benefits of supervised learning techniques into spatial regression models. This enables spatial regression models to capture non-linear interaction among different features and seamlessly integrates with modern machine learning techniques. Third, A-GWR provides a framework that allows operating on smaller chunks of data to process large datasets with limited computational budgets. Therefore, A-GWR significantly enhances the scalability and expressiveness of GWR-like models. Our novel contributions are summarized as follows:

- We propose S-MGWR, a new adaptation of the MGWR that adjusts each set of bandwidths independently rather than sequentially, making it an ideal model to be compatible with black-box optimization methods.
- We design a customized black-box optimization algorithm to accelerate discovering an optimal set of feature bandwidths for S-MGWR.
- We propose a framework for merging spatial regression algorithms such as GWR and S-MGWR with fast machine learning models via ensemble learning to improve accuracy.
- We propose a divide-and-conquer technique that splits the data and operates on its partitions to adapt for tight computational resources.
- We provide an extensive experimental evaluation that shows the superiority of our techniques in both accuracy and runtime with up to 14.4 times faster runtime.

The remainder of the paper is organized as follows: Section 2 presents related work. Section 3 reviews the basics of the GWR and MGWR and their drawbacks in more detail. Section 4 introduces S-MGWR and Section 5 introduces our ensemble framework with general machine learning models. The experiments and analysis are presented in Section 6. The last section concludes the paper.

2 Related Work

This section outlines our related work that is mainly the literature of geographically weighted regression (GWR) [11] and its variations. In GWR, regression coefficients vary across space to capture the spatial non-stationarity aspect of the data. However, GWR has limitations in both computational scalability (quadratic complexity) and expressiveness (using same spatial scale for all features). To overcome these limitations, several efforts have been made. We classify this literature into three main categories as follows.

Variable spatial scale models. To improve the GWR expressiveness over datasets with varying spatial scales for different features, different models have been proposed in [13, 20, 21, 26, 40]. All of these methods use a vector of bandwidths to express a different spatial scale for each feature instead of a single bandwidth value. To find that vector, they use an iterative method that adjusts the state of coefficients until a convergence.

Non-linear models. Combining GWR weighting schemes with artificial neural networks have been used to capture the non-linear relation between data points [8, 15]. Even though these methods can improve the accuracy, they need a tremendous amount of data to train the neural net which, in many cases, is not possible. Also, these approaches do not address the multiscale bandwidths and assign the same locality to all the features [15]. Finally, besides prediction, the GWR is a popular visualization tool to explore patterns in datasets [42]. Given that interpretability is the Achilles' heel of deep neural networks [41], combining GWR with neural networks may hurt the visualization and interpretability of GWR. Another approach to capture the non-linearity is to use non-linear kernels on data and extract features and then use a GWR on extracted features [22]. A key downside of this approach is that, identifying good kernels is task-specific and the kernel design can be challenging.

Parallel and grid models. Parallelization [24, 25] is one way to improve the runtime of GWR. However, the resulting models are still bounded to comparably small datasets as they do not reduce the time complexity. Another approach is to develop grid infrastructures. In this approach, the data points are divided to different sections and different GWR model is trained over these sections [10, 17].

Distinguished from existing work, A-GWR is the first model that simultaneously enables using variable spatial scales, capturing non-linear feature interactions with customizable general-purpose models, and using both parallelization and grid-based modeling to support large datasets. First, our proposed Stateless-MGWR (S-MGWR) model trains a stateless model to find a vector of bandwidths. This enables better computational scalability for variable spatial scales models, while maintaining the high model expressiveness. Second, our A-GWR multi-stage framework enables combining GWR variants with any general-purpose learning models so it effectively

visualizes the locality and captures the non-linear relations between features. A-GWR adapts to various spatial and general models and it is fully customizable for use with a problem-specific kernel or a completely general method. Third, A-GWR framework enables using both parallelization and grid-based modeling. It uses grids to split large data based on spatial locality. The grid system in A-GWR is flexible to use various split criteria including equal-sized cells, variable cells, and clusters. In addition, the bandwidth search method of S-MGWR, which is the bottleneck, is also parallelizable. Having all these features, A-GWR outperforms all existing models as a generic model that supports various applications with high computational scalability and model expressiveness.

3 Preliminaries

This section introduces the preliminaries of GWR [11] and MGWR [13].

GWR formulation: In GWR [11], the parameters can vary by location. Assume that each observation i is in the location (u_i, v_i). Then the GWR model is

$$y_i = \sum_{j=0}^{k} \beta_j(u_i, v_i) x_{i,j} + \epsilon_i.$$

$\beta_j(u_i, v_i)$ can be computed using the following equation:

$$\hat{\beta}(u_i, v_i) = [X^T W_i X]^{-1} X^T W_i y.$$

W_i in this equation is a diagonal matrix of weights assigned to each data point based on its distance to the i-th observation. The main idea is to assign more weight to data points closer to i-th observation as close data points tend to share more similarities. Matrix W_i is calculated based on a specified kernel function. This kernel function can be adjusted using an input variable. This variable, known as bandwidth, is determined using a number of trials. In each trial, a bandwidth is selected and then the kernel function uses the bandwidth to construct the weights. GWR then uses these weights to solve the equation. The bandwidth that produces the least error is selected as the trained bandwidth value and is used during the prediction. The problem with GWR is that it bounds the locality of all features to a single bandwidth. That is, all the features are assumed to have the same locality which limits the flexibility of GWR in describing real-world datasets.

MGWR formulation: MGWR [13] tries to address the the single locality issue by introducing a bandwidth for each feature. The MGWR is formulated under Generalized Additive Model (GAM):

$$y = \sum_{j=0}^{k} f_j + \epsilon \tag{1}$$

Where f_j is a smooth function estimated with covariate-specific bandwidths and the ϵ is the i.i.d error. MGWR uses the backfitting algorithm to determine the f_j. To compute f_i, the backfitting algorithm interpolates the function in data points' locations. For the initial state, all the coefficients are assigned with an initial value. Then, MGWR at the first iteration, fixes all the functions $f_{j\neq 1}$ and considers them as constants, then, it solves a uni-variate GWR for f_1 and finds the optimum bandwidth for it. Using this optimal bandwidth, the MGWR updates its state by the new coefficients computed for function f_1 interpolation. For the second iteration, all the values of the updated state, except for the interpolated values of the second function f_2, are considered constants. This procedure is then repeated for all the different features. After adjusting all the features, MGWR starts to readjust f_i with the updated state. The process continues until a stopping criteria is reached.

MGWR allows each feature to have its own spatial scale, represented by a separate bandwidth. Due to the flexibility of locality for each feature, MGWR fits the training data well and computes a good estimate over the coefficients of training data which makes it a good tool for data analysis. However, the backfitting algorithm introduces a number of problems. The first problem in MGWR is its running time. It solves a GWR for each feature until convergence. The number of iterations for convergence can be large and therefore the runtime of the program can be very high. After computing a set of bandwidths, MGWR cannot produce the same result using only that set of bandwidths. During the training, each feature is adjusted using the updated state. This means that for creating the same results, a complete history of all the intermediate bandwidths is required. Due to this problem, MGWR is bound to use iterative methods and the bandwidth search method cannot be parallelized.

4 Stateless-MGWR

In this section, we propose Stateless-MGWR (S-MGWR), a stateless variation of MGWR spatial model that assigns different localities to different features. S-MGWR extends MGWR by eliminating the need for a history of bandwidth values. Therefore, S-MGWR is able to fit a set of bandwidths directly, hence, any search algorithm, independent of being parallel or linear, can be used to find the locality of each feature. S-MGWR can fit a model for a given vector of bandwidths, using this ability, a search algorithm takes repeated steps. In each step, a vector of bandwidths is generated and based on the validation error over the trained model, the search algorithm decides its next step. This is in contrast to MGWR where the bandwidth search happens one feature at a time and the trained model depends both on the bandwidth and the previous state of the model.

Algorithm 1 gives the pseudo-code for the training phase of S-MGWR model. S-MGWR initializes the model values with the results of a GWR model fitted to the data. These values are stored in *initials* and are not re-computed again. Afterward, starting from the *initials* values, it proceeds to adjust the model based on the bandwidths. To adjust the model, S-MGWR adjusts features based on the vector of bandwidths one at a time. Starting from the first feature, the rest of the coefficients are fixed as constants and updates the dependent values accordingly. A weighted least squares computes the adjusted coefficients of the feature and the algorithm proceeds to the next feature. For each set of bandwidths generated by the bandwidth search algorithm, using *generate_bandwidth* subroutine, S-MGWR adjusts the model based on the bandwidths and computes a validation error based on the adjusted model. The set of bandwidths with minimum validation error is then selected as the trained bandwidths for the S-MGWR model. This process is repeated until a user-defined number of iterations are performed. In each cycle of error computation, we set the number of iterations to three as this is enough experimentally to adjust coefficients for the best set of bandwidths in this cycle. However, stopping criteria of error computation could also depend on a certain error difference threshold or a certain amount of time.

Algorithm 2 details the *error* subroutine. The subroutine takes two datasets of points, training and validation data, each point

Algorithm 1: S-MGWR Training Phase

```
1  Input: L, training data D_t, validation data D_v
2  Output: A vector of k bandwidths
3  initials = GWR(D_t ∪ D_v)
4  bw = initials.bandwidth
5  // compute_error is detailed in Algorithm 2
6  error = compute_error(3, bw, D_t, D_v)
7  while number of iterations < L do
8      next_bw = generate_bandwidth(bw, error)
9      next_error = compute_error(3, next_bw, D_t, D_v)
10     if next_error < error then
11         error = next_error
12         bw = next_bw
13     end
14 end
15 return bw
```

Algorithm 2: compute_error

```
1  Input: T_c, bandwidth vector bw, training data D_t, validation
       data D_v
2  // each d ∈ D_t or D_v consists of (X, y, coordinates)
3  Output: validation error
4  // initials computed in Algorithm 1
5  coefficients = initials.coefficients
6  while number of iterations < T_c do
7      foreach feature j do
8          foreach observation i in D_t do
9              w_i = get_weights(coordinates_i, bw_j)
10             x' = X_{:,j}
11             y' = y − (X_{j̄} × coefficients_j^T)
12             coefficients_{i,j} = WLS(x', y', w_i)
13         end
14     end
15 end
16 ŷ = predict(bw, coefficients, D_v)        // see Algorithm 3
17 return residual error of ŷ
```

consists of a triple $(X, y, coordinates)$, where X is the features, y is the dependent variable, and *coordinates* are the location coordinates. Line 5 initializes the coefficients to the initial coefficients values of GWR as computed in Algorithm 1. Then, for a user-defined number of iterations, it adjusts coefficients of all features one by one using each observation in the input training data. In particular, for each regression point i, *get_weights* subroutine computes the weights of other data point based on its distance to point i. In the next lines, $X_{:,j}$ is the j-th column of X, $X_{:,\bar{j}}$(coefficients$_{:,\bar{j}}$) refers to all the values in X (coefficients) except for the values in column j, and the $*$ operator is the sum of the rows of the Hadamard product:

$$c = A * B \Rightarrow c_i = \sum_j A_{i,j} B_{i,j}. \quad (2)$$

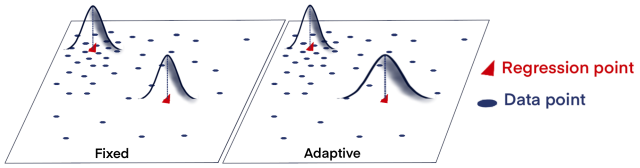


Figure 1: Adaptive bandwidth illustration. Points are weighted based on a their distance from the regression point by a fixed kernel in the left plane and by an adaptive kernel in the right plane.

The subroutine *WLS* returns the value of weighted least squares [7] for the features vector and the dependent vector using the weights w_i. The predict subroutine, that is called in Algorithm 2 line 16, is the same subroutine that is used to predict new observation. We detail the prediction procedure in Section 4.3.

S-MGWR and MGWR share the core idea of using distinct bandwidths for different features to enable multiple spatial scales. Also, both techniques adjust the model one feature at a time using a variation of weighted least squares. However, there are significant differences between the two models. MGWR uses back-fitting to adjust the coefficients which means for every iteration and every feature, MGWR searches for the best bandwidth for that specific feature and then updates its state based on the trained coefficients calculated from the bandwidth. For example, based on the initial values, MGWR finds b_1 as the best bandwidth describing the first feature. Then, based on this it will update all the other features. When MGWR tries to readjust the first feature, it will use the updated coefficients which were computed based on b_1. Choosing a different bandwidth in first iteration can greatly impact the second iteration and the final result. This means, to produce the same result as training MGWR needs the complete history of the bandwidths to adjust the model. Generating the complete history where the length of the sequence is not known extends the search space by orders of magnitude making it very difficult to use a black-box optimization algorithm. This problem does not exist in S-MGWR where the bandwidths adjust the model independently from the values of past bandwidths. In addition to the flexibility to use various optimization algorithms, it also enables S-MGWR to search multiple bandwidths simultaneously in parallel. Such parallelization facilitates running more iterations to find the set of bandwidth that best describes the data. So, it boosts both runtime scalability and accuracy of the model without compromising the model expressiveness.

The remainder of this section detail different parts of the S-MGWR technique. Section 4.1 explains the concepts of the weight kernel used to compute the weights of data points in Algorithm 2. Section 4.2 discusses different strategies to explore the bandwidth space to find the optimal set of bandwidths. Section 4.3 explains the prediction procedure for the validation as well as the test data. To further improve the runtime performance of S-MGWR, we use a Least Recently Used (LRU) cache for the computed weights that is described in Section 4.4.

4.1 Spatial Weighting Scheme

The weight function can be any function that assigns weights to the data points based on their distance to the regression point. The

following equation is an example of computing weight of point k based on regression point i and bandwidth of feature j using a Gaussian weight function.

$$w_i[k] = exp[-\frac{1}{2}(\frac{d_{i,k}}{b_j})^2]. \quad (3)$$

In Equation 3, $d_{i,k}$ is the spatial distance between data point i and regression point k. b_j is computed using the bandwidth parameter related to the feature j. The bandwidth value for each feature helps indicate how local or global the impact of that feature is. We use adaptive bandwidths in S-MGWR. For each bandwidth b for feature j and each regression point i, the value b_j (Equation 3) is computed based on the distance of b nearest neighbors of data point i. This allows the weighting scheme to adjust the weight kernel based on how dense or sparse the neighborhood of the regression point is. Figure 1 is an illustration of fixed and adaptive kernels. In Figure 1, the dark blue points represent the data points and the red points represent the regression points. The left plane shows a fixed kernel where for each regression point the same kernel is applied. Either the locality is sparse or dense with data points, the same influence range is used. The plane on the right represents the adaptive kernel where different kernel functions are applied for points in dense or sparse areas. Areas that are sparse and have fewer data points use larger influence ranges to capture enough nearby points. On the contrary, areas that are dense with data points use smaller influence ranges as they will be enough to capture nearby points. In both cases, the weight for distant points would be very small. Therefore, in order to improve the speed, we introduced a cutoff distance to our algorithm. For each b_j and cutoff number c, any point with a distance more than $c \times b_j$ from the observation point j is not considered and has a weight of zero. The example kernel of Equation 3 will then change to:

$$w_i[k] = \begin{cases} exp[-\frac{1}{2}(\frac{d_{i,k}}{b_j})^2] & d_{i,k} \leq b_j \times c \\ 0 & otherwise \end{cases} \quad (4)$$

Cutoff threshold reduces the search space by allowing the algorithm to omit the data points that are far enough from the regression point.

4.2 Bandwidth Search Strategies

The bandwidth should reflect the rate that regression weights decay around a certain location. Using a sensible weighting function, if the bandwidth is small, weights decrease quickly as the distance between a data point and the regression point increases. On the other hand, using big bandwidths should create a smoother surface and the weights would decline slowly as the distance increases. S-MGWR uses adaptive bandwidths. That means S-MGWR uses the number of neighbors as bandwidths and computes the locality based on the distance of its neighbors. Adaptive bandwidth gives S-MGWR the ability to adjust the locality based on the sparsity of the neighborhood around an observation.

The bandwidth search space is extremely large. If n is the number of observations and m is the number of features, then the bandwidth space size is n^m. Thus, a smart algorithm is necessary to explore the huge search space effectively. We have implemented Hyperband [23], Simulated Annealing, Bayesian Optimization[36], Simultaneous Perturbation Stochastic Approximation (SPSA) [37], and Hill Climbing to search for the best set of bandwidths. We also created a combination of these algorithms to benefit from the strength of each algorithm. Our approach uses a combination of SPSA, Bayesian Optimization, and Hill Climbing methods. This combination is founded on studying the effect of each method on error improvement over time and its speed of convergence to the best error, as outlined in Appendix A and Figure 6. SPSA method is used to explore the space as broadly as possible in a short time frame. After that, the Bayesian Optimization is performed to explore the space around the SPSA's result. Finally, to find a local optima, our algorithm performs the Hill Climbing search. The details of implementation for these methods are discussed in Appendix A.

4.3 Prediction Procedure

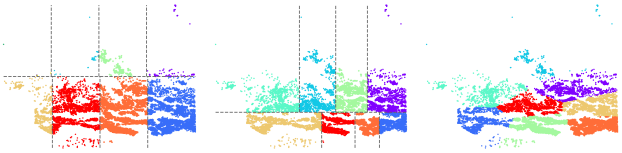
To predict the dependent value for a new sample, similar to the train process, the algorithm computes the coefficients of this sample. Algorithm 3 shows the prediction procedure. For each feature of each new sample, the weight matrix is generated based on the location of the new sample and the training data. After computing the weight matrix, a weighted least squares will compute the coefficient which is used to compute the predicted dependent value of the sample.

Algorithm 3: predict (S-MGWR Prediction Phase)

Input: bandwidth vector bw, trained_coefficients, dataset D
Output: prediction values
m = |D|
k = number of features of $d \in D$
X = $m \times k$ matrix of feature values of D
coords = $m \times 2$ matrix of location coordinates of D
coefficients = $[0]_{m \times k}$
foreach *observation i in D* **do**
  foreach *feature j* **do**
    w_i = **get_weights**(coords$_i$, bw_j)
    $x' = X_{:,j}$
    $y' = y - (X_{\bar{j}} \times \text{trained\_coefficients}^T_{\bar{j}})$
    coefficients = WLS(x', y', w_i)
  end
end
predictions = coefficients $\times X$
return predictions

4.4 Accelerating Weight Computation

For each bandwidth value, a weight vector with size n is computed for each observation point i where n is the number of training data points. Thus, for each bandwidth, we can store all the weights of all observations on a $n \times n$ matrix. Since S-MGWR uses adaptive bandwidth, the bandwidth can only be a positive integer number smaller than n. Therefore, all the possible weights would require $O(n^3)$ space. Since most generated bandwidths have at least one value in common with previous bandwidths, storing the values of previously computed weights can be helpful to reduce the runtime of S-MGWR. In our implementation, we use a cache with *least recently used* (LRU) replacement policy. If the cache stores c (a constant number) bandwidths at a time, then the memory usage will



(a) Equal Height Grid (b) Equal Count Grid (c) k-means Splitting

Figure 2: Splitting a sample data into eight sections using grid and k-means methods.

be $O(c \times n^2) \rightarrow O(n^2)$. The impact of cache is noticeable, especially when using the Hill Climbing algorithm for the bandwidth search. In Hill Climbing, only one of the bandwidths changes each time. Hence, the rest of the weights can be read from the cache. We evaluate the caching impact in our experimental evaluation.

5 Augmented GWR (A-GWR)

S-MGWR provides a multi-bandwidth spatial regression model that does not need a full history of bandwidth values for computations. Hence, it has eliminated the dependency problem among the consecutive iterations of adjusting the model parameters. To make use of such significant advancement in improving the accuracy and reducing the runtime, we propose augmented GWR (A-GWR) model. A-GWR divides data into different sections. Then, a combination of S-MGWR and a general-purpose supervised learning model is used for each section. S-MGWR, as a spatial model, captures the spatial relationships of the data and the other model captures nonlinear interactions. To combine the models, we use two different methods, namely, *pipeline* and *ensemble*, that apply models in a different order. A great advantage of A-GWR is that it is generalizable to other spatial regression models as well. This gives flexibility for spatial applications to choose the best models according to their needs and data characteristics. This is also applicable for the general-purpose supervised model, which can be a simple least-squares algorithm or a complex neural network. However, for specificity, and without loss of generality, we discuss A-GWR in the rest of this section using **S-MGWR** as a spatial model and **Random Forest** as a general-purpose learning model. The rest of the section discusses data splits, model integration, and computational complexity.

5.1 Data Splitting

In order to divide the data, we use two methods that can be applied to different applications.

• *Grid Splitting:* Dividing data into grid cells is a popular method which is also used in [10, 17]. In the grid splitting, the data is divided into grid cells where the number of rows and columns are defined by the user. The grid either divides the space into equal-height cells, by dividing the space to equal height rows and then dividing the rows to equal width sections, as in Figure 2a. Another grid option is to divide the space into varying size cells where each of them contains the same number of training data points, as in Figure 2b. For a new point, its corresponding grid cell is located. Then, the result of the model associated with that grid cell is reported.

• *Cluster Splitting:* Using randomly generated k-means centers, we perform k-means clustering for training data points. Each cluster

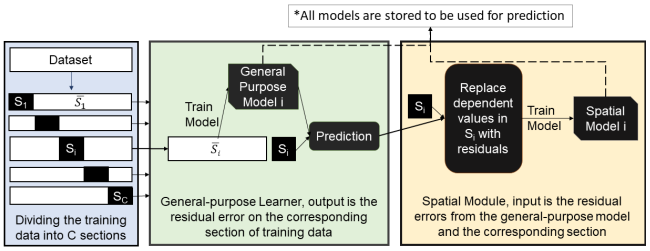


Figure 3: Ensemble model integration order (general-purpose model first).

represents a section of data containing all the training data points in that cluster. For a new point, after finding its designated cluster, the result of the model associated with that cluster is reported. An example of this method is shown in Figure 2c.

Using k-means and grid splitting methods focus more on preserving the spatial neighborhood, which is more in accordance with Tobler's first law of geography, i.e., "everything is related to everything else, but near things are more related than distant things" [38]. Our empirical evaluation has confirmed the advantage of these splitting methods.

5.2 Model Integration

This section discusses integrating the two models employed by A-GWR: S-MGWR and Random Forest. We apply these to different data sections using two different orders. The first order, *ensemble*, employs the Random Forest model first, while the second order, *pipeline*, employs S-MGWR first. We outline each of them below.

• ***Ensemble (general-purpose model first):*** In this method, for each section i, first, we train a Random Forest (RF) model on all the training data except for the data in section i. Then, using the trained model, we predict dependent values for the data in section i. Since this data was not in the training process, the model does not overfit the data in section i. Using the true dependent values and the predicted ones, we compute the residual errors. In the last step, an S-MGWR model is fitted to the data and the residuals. This model captures the spatial relation between data that the RF model fails to capture. In addition, due to running the spatial model on a fraction of data, the runtime is improved.

Figure 3 demonstrates the *ensemble* process on C sections of data. For Section i, first, a random forest model is trained over all the data in the dataset except for data in section i to create model i. Then, the data of Section i is passed to Model i to compute the residual errors. These residuals are then passed to the spatial S-MGWR model to capture the spatial relativity of the data in each section. For predicting a new sample, if the data splitting method is k-means or grid data, then the new sample is homed into a corresponding data section. Afterward, the data is passed to the corresponding S-MGWR and RF models of that section. The result is the sum of the two outputs. Another option is to run through all the models and then combine the result based on the distance of each section to the point. This can be measured by the distance of the grid center or cluster center of that section to the point.

• ***Pipeline (spatial model first):*** In this method, we feed the training data to the spatial S-MGWR model first. Then, the residual data

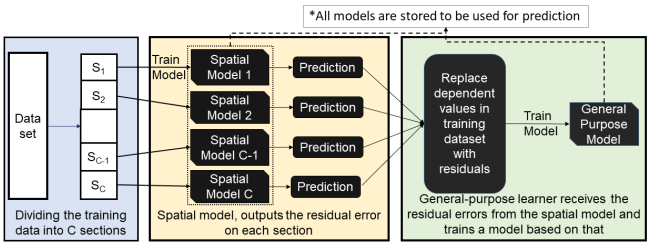


Figure 4: Pipeline model integration order (spatial model first).

computed from the spatial model is passed to the RF model. The prediction is the same as the ensemble method. After determining the section, the sum of results from both spatial and fast models is reported as the result. Figure 4 shows the pipeline method on C data sections. For each section, a spatial model is trained and then the residual errors are passed to the RF model. The spatial model might overfit the data preventing the RF model to completely capture the global relations between data. However, when using models such as S-MGWR where the base is using less complex methods such as least squares, this is not the issue and the pipeline method provides great improvements on accuracy.

5.3 Computational Complexity

Assuming a training dataset of n observations and k features, in Algorithm 1, we perform L iterations, each iteration performs bandwidth for the k features (in $O(k)$) and residual error computation, which is more expensive than the bandwidth search as detailed in Algorithm 2. So, the overall complexity is L multiplied by the cost of error computation. The error computation involves T iterations, each goes over each feature and each training observation to get weights and compute coefficients. According to Algorithm 2, this performs a total of $O(Tkn)$ iterations, each computes weights in $O(n)$ (Line 9), which gives an error computation complexity of $O(Tkn^2)$. So, the overall complexity of training is $O(LTkn^2)$.

A-GWR breaks data into different sections, each section contains a constant number of observations C, so it operates on $\frac{n}{C}$ sections. At each section, the training cost is $O(LTkC^2)$. For all sections, the overall complexity is $O(LTkC^2) \times \frac{n}{C} = O(LTkCn)$. Having L, T, and C as constants, then the overall complexity is $O(kn)$.

This complexity analysis does not consider parallel processing. However, our techniques are parallelizable as different sections are independent and can be trained in parallel using J threads, which reduces the overall time by a factor of J, where each thread complexity is $O(LTkC^2)$. Thus, with constant L, T, C, the overall time would scale as $O(kn/J)$. Also, the bandwidth search process is parallelizable, thanks to the stateless nature of S-MGWR, so it is possible to independently validate each set of bandwidths.

6 Experimental Evaluation

We have implemented A-GWR using Python programming language and we use PySAL library [33] modules to run MGWR and GWR techniques with Gaussian kernel and CV criterion for the search method. For Random Forest (RF) learning model, we use scikit-learn Python library with 60 as the number of estimators and default values for the rest of the model parameters. We report results on the pipeline model integration. For the multiprocessing, we used the *multiprocessing* Python library. We set the number of learners to the minimum value of the number of sections and the number of CPU threads. The number of learners indicates the number of pools to create. Each pool is responsible for a data section. For the spatial data, features are separated from location attributes, and these two properties are passed to the spatial model. For the random forest, the features and locations are concatenated and used as the training data.

All techniques run on an Intel(R) Xenon Silver 4214 machine with CPU @ 2.20GHz and 32 GB of RAM. We evaluate the performance using a total of eight different datasets, six real datasets, and two synthetic datasets. Unless mentioned otherwise, by default each data section contains one thousand data points. Our performance measures include training runtime and prediction accuracy. The scripts to A-GWR and S-MGWR are available at Github [1]. Prediction accuracy are measured on the test data using $[1 - (R^2)]$ error measure:

$$1 - R^2 = \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Evaluation datasets. We use eight datasets, six of which is real and two is synthetic. Four out of the six real datasets are obtained from PySAL library [33], and these datasets are the standard ones that are used to evaluate various variations of GWR and MGWR techniques [18, 30, 31] . Below is the description of different datasets.

• ***PySAL datasets:*** We use four of the datasets provided in the PySAL library, originally hosted by the Spatial Analysis Research Center of the Arizona State University that provided GWR and MGWR software. The datasets represent the socio-economic variables for counties of Georgia, Tokyo Mortality data, the clearwater dataset, and Prenzlauer Berg neighborhood AirBnB data [9]. The dataset sizes are 159, 262, 239, and 2203 respectively. Due to the small size of these standard datasets, we solicited the other two large datasets that are described below.

• ***New York City Airbnb Open Data:*** This dataset, denoted as *NYCAirBnb*, is Airbnb data for the New York city [6]. After removing rows with missing data, we use the room type, minimum nights that a guest can stay, number of reviews, reviews per month, amount of listing per host, and the availability of the unit along with the location to predict the listing price. The price in this dataset is skewed and therefore we use the logarithm of the price. The dataset has 38782 listings after clean up.

• ***House Sales in King County, USA:*** This dataset, denoted as *kingHousePrices*, has the house sale prices for King County [16]. After removing instances with missing values, we select the number of bedrooms, number of bathrooms, square footage of the apartment's interior living space, number of floors, house condition, and grade, the year built, and the location as our features to predict the price of the house. In total, the dataset has 18708 house sales.

• ***Synthetic Dataset 1:*** We create a synthetic dataset, denoted as *synthData1*, of 1296 points that is very close to what is described in MGWR [13]. We define five different surfaces for covariates and randomly generate the values for features and compute the dependent values based on them. The synthetic dataset is defined by the pair (l, b) where l is the size of the grid and $1 \leq b \leq 5$ is

Method	1 - R^2 ($\times 10^{-3}$)
Hyperband	3.475
SPSA	2.570
Simulated Annealing	2.535
Bayesian Optimization	2.347
Hill Climbing	2.023
Fibonacci Hill Climbing	2.022
Combined Search	**1.955**

Table 1: Error value of different bandwidth search methods

the number of surface functions to use. We define the dependent values using the following equation:

$$y_i = \epsilon_i + \sum_{j=0}^{b-1} \beta_j(u_i, v_i) x_{i,j} \tag{5}$$

Where u_i and v_i are the horizontal and vertical location of each point in the grid, $0 \leq u, v < l$. The definition of The covariate surface functions (β_0 to β_4) are explained in more detail in the Appendix B.4. Figure 7 is an illustration of the surface functions. For all the points in an $l \times l$ grid, we generate x_1 to x_b randomly from a normal distribution $\mathcal{N}(0, 1)$ and combine it with the error term ϵ_i that is randomly generated from $\mathcal{N}(0, 0.5)$.

• ***Synthetic Dataset 2:*** We created a second synthetic dataset with (l, b)=(40, 5), denoted as *synthData2*, of 1600 points that is similar in structure to the first synthetic dataset. However, one feature is developed in a way that cannot be computed with a simple regression over neighboring data points. The rationale of this dataset is to test different techniques for capturing nonlinear interactions among features (as we will evaluate in Table 3 later in the section). The dependent value for this dataset is computed as:

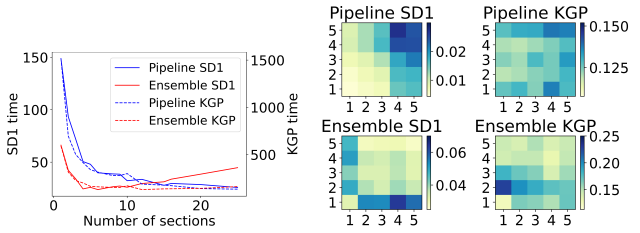
$$y_i = \epsilon_i + \sum_{j=0}^{3} \beta_j(u_i, v_i) x_{i,j} + \left(\frac{1}{4} x_{i,j} x_{i,j-1}\right)^2 \beta_4(u_i, v_i).$$

6.1 Model Tuning

We performed a series of experiments to compare different bandwidth search methods and tune the data section size and show their impact on accuracy and runtime.

Bandwidth search method. To choose the best strategy to search the bandwidth space, we applied each method over the *synthData1* dataset for the same amount of time and compared the results. Due to the random nature of the algorithms, we performed each experiment five times and reported the average error value in Table 1. The Fibonacci version is similar to the ordinary versions but the bandwidths are forced to be Fibonacci values. Using Fibonacci numbers limits the searching space and increases the cache hits. The combined search strategy uses SPSA, Bayesian Optimization, and Hill Climbing enabling it to search globally and locally for the optimum set of bandwidths. As shown in Table 1, the combined search strategy gives the lowest error compared to other strategies. This shows the superiority of our proposed combined search strategy. Therefore, this strategy is used as the default strategy for bandwidth search in other experiments.

Data section size. In A-GWR, one of the most important factors that affect accuracy and runtime is how many data sections the training data is divided to, and hence the size of each data section.



(a) Runtime based on different number of sections **(b) R^2 error for different section sizes**

Figure 5: Accuracy and runtime over synthData1 and kingHousePrices datasets

There is a trade-off between runtime and accuracy. If there are too many sections, each section contains a few data points and cannot necessarily capture the relations among the data points which reduces the accuracy. If each section has too many points, the runtime will be slow. Figure 5 shows the result of different division sizes over two different datasets, *synthData1* (SD1) and *kingHousePrices* (KGP). We use GWR as the spatial module and Random Forest (RF) as the general learning module. Figure 5b represents the error values for different section sizes. Cell (a, b) in Figure 5b is the error value if we divide the dataset to $a \times b$ sections. The figure shows that by dividing the data to more sections, the runtime of the *pipeline* integration method will drop (Figure 5a) and the error rate will generally increase (Figure 5b). The *pipeline* integration method relies on the spatial module as the first model and when the section size is too small, the spatial module cannot capture the relation between data points. For the *ensemble* integration method, the runtime increases after a certain point (10 data sections in Figure 5a), this is because the increase rate of runtime of the Random Forest model is higher than the decrease rate in the runtime of GWR. As the number of sections increases, the training data fed to the Random Forest model increases. Since Random Forest is the first module of *ensemble*, it captures most of the relation between data points. Thus, more sections and consequently more data increase the accuracy.

6.2 Model Performance

We perform a series of experiments to evaluate the performance of A-GWR, showing the separate impact of S-MGWR as well as the impact of combining it with the other parts of A-GWR framework. In all datasets except for the largest two (kingHousePrices, NYCAirBnb), we average the results over 5 independent train-test splits of the original training data (80%-20%). This is important to ensure reproducibility as we observed that accuracy has large variability for small datasets. See Appendix for further details.

Evaluating S-MGWR performance. To compare the performance of S-MGWR with MGWR and GWR, we use the *synthData1* dataset. We have the true values of coefficients for this dataset and we can compute how good each method is computing the coefficient. For a generated dataset we split the data to train and test sections and train the S-MGWR, MGWR, and GWR over the train section. We then report the $1 - R^2$ error over the test section and the root mean square error (RMSE) of the coefficients of training points. Since the dataset is generated randomly, we generated ten different

	synthData1		synthData2		kingHousePrices		NYCAirBnb	
	Time (s)	1 - R^2	Time (s)	1 - R^2	Time (s)	1 - R^2	Time (s)	1 - R^2
GWR	1.5558	0.0121	5.0826	0.9101	512.0763	0.1441	2064.8780	0.4284
A-GWR w\GWR	4.3137	0.0031	9.7359	0.1962	138.563	0.1288	345.7216	0.3932
MGWR	35.7843	0.0016	58.3185	0.8868	80501.9860	0.1378	103713.3542	0.4378
A-GWR w\MGWR	41.0223	0.0016	63.8316	0.1830	5574.1644	0.1143	21460.6177	0.4125

Table 2: Evaluating A-GWR as a framework.

	pGeorgia	pTokyo	pClrwatr	pBerlin	synthData1	synthData2	kingHousePrices	NYCAirBnb
	1 - R^2	1 - R^2	1 - R^2	1 - R^2	1 - R^2	1 - R^2	1 - R^2	1 - R^2
A-GWR (pipelined)	0.4161	**0.0164**	**0.9226**	0.6821	**0.0137**	**0.0610**	**0.1109**	0.4181
MGWR	0.4913	0.0226	0.9622	0.6965	0.0139	0.0999	0.1341	0.4378
GWR	0.3914	0.0215	0.9265	0.6876	0.0805	0.1168	0.1295	0.4284
Random Forest	**0.3878**	0.0352	0.9276	**0.5825**	0.1788	0.1240	0.1216	**0.3927**
XGBoost	0.6229	0.0359	1.0304	0.6942	0.1634	0.1042	0.1165	0.4092

Table 3: Performance of A-GWR, MGWR, GWR, and Random Forest over different datasets

Model	Time (s)	Coeff. RMSE	1 - R^2 ($\times 10^{-3}$)
GWR	1.18	3651.77	9.5
MGWR	42.88	467.75	1.2
S-MGWR	28.69	742.42	1.7
S-MGWR*	-	468.20	1.2

Table 4: Performance of S-MGWR, GWR, and MGWR. S-MGWR* is the S-MGWR model with bandwidth computed by MGWR.

	Time (s)	
Cache	synthData1	synthData2
Enabled	1133.2258	764.5523
Disabled	1612.3177	1070.3291

Table 5: Runtime of S-MGWR with and without cache

datasets and report in Table 4 the average values of runtime, $1 - R^2$ error, and the coefficients RMSE over the ten simulations.

Table 4 shows that S-MGWR fills the gap between GWR and MGWR. It fits the data more accurately than GWR due to its flexibility and it has a better runtime than MGWR due to the ability to use advanced black-box optimization algorithms. Also, running S-MGWR with the bandwidths computed by MGWR (denoted as S-MGWR*) shows a small difference in coefficients RMSE while maintaining the best $1 - R^2$ error value. The table does not report runtime for S-MGWR* as it has no training phase and uses the bandwidths computed by MGWR directly. S-MGWR* shows that given a good bandwidth selection method, being stateless does not affect the accuracy. This shows even though S-MGWR does not have the access to MGWR bandwidth history, its performance is comparable to MGWR. This makes it a perfect candidate to be used as part of high-level models such as A-GWR or parallelized models.

Evaluating A-GWR framework. This experiment shows the benefits of A-GWR as a framework that could adapt different spatial models and learning models. We study A-GWR with *pipeline* model integration where we explore both MGWR and GWR as the spatial model and Random Forest (RF) as the general learning model. The primary motivation behind using *pipeline* over *ensemble* is to promote the spatial model to do most of the prediction. Secondly, since RF is more prone to overfitting, residuals after first fitting with RF may be too small or lose their spatial relation reducing the benefits of the subsequent spatial model. We consider four datasets: *synthData1*, *synthData2*, *kingHousePrices*, and *NYCAirBnb*. Each dataset is divided into sections so that each section contains at most 1000 data points. The synthetic dataset sections are divided into two columns with equal number of points. The *kingHousePrices* dataset is split using a grid to 4×5 grid with equal number of points. Finally, the *NYCAirBnb* dataset is split to a 6×7 grid with equal number of points. The result of A-GWR with MGWR and A-GWR with GWR along with the ordinary versions are shown in Table 2. Both A-GWR versions of GWR and MGWR outperform the ordinary versions in accuracy. In addition, the A-GWR models are considerably faster than the ordinary versions on real datasets. The A-GWR with MGWR performs up to 14.4× faster than MGWR and A-GWR with GWR performs up to 5.9× faster than GWR.

Finally, we compare A-GWR with S-MGWR as a spatial module and Random Forest (RF) as the general learning model with MGWR, GWR, Random Forest, and XGBoost [4]. We use scikit-learn library [32] Table 3 shows the error rate of these methods over different datasets where A-GWR outperforms all other techniques for most of the datasets. As expected MGWR and A-GWR both perform very well on the synthetic dataset 1. However, due to the nonlinearity in synthetic dataset 2, A-GWR performs much better than MGWR. In our experiments, we use Random Forest implemented in the scikit-learn library [32]. For our GWR and MGWR models, we use a slightly modified version of the mgwr package provided in PySAL[30]. The modified version is made available on our Github page [1]. For Random Forest we tuned its hyper-parameters using a random search algorithm to find the best combination of number of estimators, max features, max depth, min samples split, min samples leaf, and bootstrap setting. For the XGBoost model, we used the default values of the library.

Evaluating cache impact. To evaluate the impact of the LRU cache, we compare the runtime of A-GWR over the two synthetic datasets. Runtimes that are listed in Table 5 shows around 30% improvement in time when using cache to store weights. For large datasets, this percentage is considerable and contributes towards faster training phases of A-GWR models.

7 Conclusions

This paper has introduced Augmented Geographically Weighted Regression (A-GWR), a spatial regression framework that generalizes MGWR with stateless training, augmented general-purpose learning models, and parallelization features to handle large and complex spatial datasets. A-GWR introduces Stateless-MGWR (S-MGWR) that is a flexible spatially-varying coefficient (SVC) model with distinct bandwidths for individual features to adjust spatial scale based on the feature impact. S-MGWR is combined with state-of-the-art black-box optimization techniques to find the optimum set of bandwidths efficiently. A-GWR framework achieves higher expressiveness by augmenting powerful machine learning models with spatial models to capture both spatial and non-spatial aspects of the data. In addition, A-GWR achieves scalability by intelligently diving data to smaller sections while preserving the spatial relations between points. Extensive empirical evaluation on eight real and synthetic datasets has shown that A-GWR is a fast, flexible, and scalable algorithm to model spatial data. It achieves better accuracy while performing up to 14.4 times faster than existing models.

References

[1] [n.d.]. https://github.com/mshahneh/AGWR

[2] Stuart Brown, Vincent Versace, L. Laurenson, Jonathon Fawcett, and Scott Salzman. 2012. Assessment of Spatiotemporal Varying Relationships Between Rainfall, Land Cover and Surface Water Area Using Geographically Weighted Regression. *Environmental Modeling & Assessment - ENVIRON MODEL ASSESS* 17 (06 2012).

[3] Osvaldo Daniel Cardozo, Juan Carlos García-Palomares, and Javier Gutiérrez. 2012. Application of geographically weighted regression to the direct forecasting of transit ridership at station-level. *Applied Geography* 34 (2012), 548 – 558.

[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *CoRR* abs/1603.02754 (2016). http://arxiv.org/abs/1603.02754

[5] Henry Crosby, Paul Davis, Theodoros Damoulas, and Stephen A. Jarvis. 2016. A Spatio-temporal, Gaussian Process Regression, Real-estate Price Predictor. In *ACM SIGSPATIAL*.

[6] Dgomonov. 2019. New York City Airbnb Open Data. https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data/metadata

[7] Yadolah Dodge. 2008. *Weighted Least-Squares Method.* Springer New York, 566–569.

[8] Zhenhong Du, Zhongyi Wang, Sensen Wu, Feng Zhang, and Renyi Liu. 2020. Geographically neural network weighted regression for the accurate estimation of spatial non-stationarity. *International Journal of Geographical Information Science* 34, 7 (2020), 1353–1377.

[9] Taylor M. Oshan et. al. 2021. PySAL Datasets. https://sgsup.asu.edu/sparc/multiscale-gwr

[10] T. Feuillet, H. Commenges, M. Menai, P. Salze, C. Perchoux, R. Reuillon, E. Kesse-Guyot, C. Enaux, J.-A. Nazare, S. Hercberg, C. Simon, H. Charreire, and J.M. Oppert. 2018. A massive geographically weighted regression model of walking-environment relationships. *Journal of Transport Geography* 68 (2018), 118–129.

[11] Alexander Fotheringham, Chris Brunsdon, and Martin Charlton. 2002. Geographically Weighted Regression: The Analysis of Spatially Varying Relationships. *John Wiley & Sons* 13 (01 2002).

[12] A. Fotheringham, Ricardo Crespo, and Jing Yao. 2015. Geographical and Temporal Weighted Regression (GTWR). *Geographical Analysis* 47 (04 2015).

[13] Alexander Fotheringham, Wenbai Yang, and Wei Kang. 2017. Multiscale Geographically Weighted Regression (MGWR). *Annals of the American Association of Geographers* 107 (08 2017), 1247–1265.

[14] Shuhui Gong, John Cartlidge, Yang Yue, Guoping Qiu, Qingquan Li, and Jingyu Xin. 2017. Geographical Huff Model Calibration Using Taxi Trajectory Data. In *Proceedings of the 10th ACM SIGSPATIAL Workshop on Computational Transportation Science (IWCTS'17)*. Association for Computing Machinery, 30–35.

[15] Julian Hagenauer and Marco Helbich. 2021. A geographically weighted artificial neural network. *International Journal of Geographical Information Science* 0, 0 (2021), 1–21.

[16] harlfoxem. 2016. House Sales in King County, USA. https://www.kaggle.com/harlfoxem/housesalesprediction/metadata

[17] Richard Harris, Alexander Singleton, Daniel Grose, Chris Brunsdon, and Paul Longley. 2010. Grid-Enabling Geographically Weighted Regression: A Case Study of Participation in Higher Education in England. *T. GIS* 14 (02 2010), 43–61.

[18] M. Irfan, A. Koj, H. Thomas, and M. Sedighi. 2016. Geographical General Regression Neural Network (GGRNN) tool for geographically weighted regression analysis. In *The Eighth International Conference on Advanced Geographic Information Systems, Applications, and Services, Venice, GEOProcessing 2016.*

[19] Kevin G. Jamieson and Ameet Talwalkar. 2015. Non-stochastic Best Arm Identification and Hyperparameter Optimization. *CoRR* abs/1502.07943 (2015).

[20] Yin-Yee Leong and Jack Yue. 2017. A modification to geographically weighted regression. *International Journal of Health Geographics* 16 (12 2017).

[21] Yin-Yee Leong and Jack C. Yue. 2017. A modification to geographically weighted regression. *International Journal of Health Geographics* 16, 1 (31 Mar 2017), 11.

[22] Lianfa Li. 2019. Geographically Weighted Machine Learning and Downscaling for High-Resolution Spatiotemporal Estimations of Wind Speed. *Remote Sensing* 11, 11 (2019).

[23] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits. *CoRR* abs/1603.06560 (2016).

[24] Ziqi Li, Alexander Fotheringham, Wenwen Li, and Taylor Oshan. 2018. Fast Geographically Weighted Regression (FastGWR): A Scalable Algorithm to Investigate Spatial Process Heterogeneity in Millions of Observations. *International Journal of Geographical Information Science* (10 2018).

[25] Ziqi Li and A. Stewart Fotheringham. 2020. Computational improvements to multi-scale geographically weighted regression. *International Journal of Geographical Information Science* 34, 7 (2020), 1378–1397.

[26] Binbin Lu, Chris Brunsdon, Martin Charlton, and Paul Harris. 2017. Geographically weighted regression with parameter-specific distance metrics. *International Journal of Geographical Information Science* 31, 5 (2017), 982–998.

[27] Arabinda Maiti, Qi Zhang, Srikanta Sannigrahi, Suvamoy Pramanik, Suman Chakraborti, Artemi Cerda, and Francesco Pilla. 2021. Exploring spatiotemporal effects of the driving factors on COVID-19 incidences in the contiguous United States. *Sustainable Cities and Society* 68 (2021), 102784.

[28] Mansour Ndiath, Badara Cissé, Jean Ndiaye, Jules Gomis, Ousmane Bathiery, Anta Dia, Oumar Gaye, and Babacar Faye. 2015. Application of geographically-weighted regression analysis to assess risk factors for malaria hotspots in Keur Soce health and demographic surveillance site. *Malaria journal* 14 (11 2015), 463.

[29] Fernando Nogueira. 2014–. Bayesian Optimization: Open source constrained global optimization tool for Python. https://github.com/fmfn/BayesianOptimization

[30] Taylor M. Oshan, Ziqi Li, Wei Kang, Levi John Wolf, and Alexander Stewart Fotheringham. 2018. *mgwr: A Python implementation of multiscale geographically weighted regression for investigating process spatial heterogeneity and scale.* OSF Preprints bphw9. Center for Open Science.

[31] Taylor M. Oshan, Ziqi Li, Wei Kang, Levi J. Wolf, and A. Stewart Fotheringham. 2019. mgwr: A Python Implementation of Multiscale Geographically Weighted Regression for Investigating Process Spatial Heterogeneity and Scale. *ISPRS International Journal of Geo-Information* 8, 6 (2019).

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[33] Sergio J. Rey and Luc Anselin. 2007. PySAL: A Python Library of Spatial Analytical Methods. *The Review of Regional Studies* 37, 1 (2007), 5–27.

[34] Ibrahim Sabek, Mashaal Musleh, and Mohamed F. Mokbel. 2018. TurboReg: a Framework for Scaling Up Spatial Logistic Regression Models. In *SIGSPATIAL*.

[35] Masamichi Shimosaka, Takeshi Tsukiji, Hideyuki Wada, and Kota Tsubouchi. 2018. Predictive Population Behavior Analysis from Multiple Contexts with Multilinear Poisson Regression. In *ACM SIGSPATIAL*.

[36] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 2951–2959.

[37] J.C. Spall. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Automat. Control* 37, 3 (1992), 332–341.

[38] W. R. Tobler. 1970. A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography* 46 (1970), 234–240.

[39] Austin Troy, J. Morgan Grove, and Jarlath O'Neil-Dunne. 2012. The relationship between tree canopy and crime rates across an urban–rural gradient in the greater Baltimore region. *Landscape and Urban Planning* 106, 3 (2012), 262 – 270.

[40] Wenbai Yang, Alexander Fotheringham, and Paul Harris. 2012. An Extension of Geographically Weighted Regression with Flexible Bandwidths. In *the 20th Annual GIS Research UK (GISRUK) conference.*

[41] Quan-shi Zhang and Song-Chun Zhu. 2018. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering* 19, 1 (2018), 27–39.

[42] Bin Zou, Qiang Pu, Muhammad Bilal, Qihao Weng, Liang Zhai, and Janet E. Nichol. 2016. High-Resolution Satellite Mapping of Fine Particulates Based on Geographically Weighted Regression. *IEEE Geoscience and Remote Sensing Letters* 13, 4 (2016), 495–499.

Appendix

A S-MGWR Bandwidth Search Strategies

As discussed in Section 4.2, S-MGWR can use different bandwidth search strategies to explore the huge bandwidth search space. This appendix briefly discusses each of these methods.

◇ **Hyperband:** Hyperband [23] is an extension of successive halving algorithms [19] where a number of different bandwidths are generated and then evaluated using a fraction of training data. Then the top performing combinations are selected and evaluated using a bigger fraction of training data. This method takes three parameters, *starting_configs*, *min_budget*, and η. The *starting_configs* determines the number of different combinations to start from. The *min_budget* determines the minimum number of points for evaluating each set of bandwidths. The η controls the proportion of configurations discarded in each round of successive halving [23].

◇ **Simulated Annealing:** Simulated annealing is a well-known probabilistic optimization algorithm that simulates temperature-like heating and cooling moves in the search space. It takes parameters T, α, *steps*, *updates*, μ, and σ. T and α determine the temperature. The neighboring states are generated based on a normal distribution $N(\mu, \sigma)$. For each state, the number of *steps* is reduced by one. If the new state is better than the best state found so far, the number of *updates* is reduced by one. The algorithm returns the best value when *steps* or *updates* reach zero.

◇ **Hill Climbing:** Starting from an arbitrary solution, at each iteration of the Hill Climbing method, only one of the features' bandwidth is changed. If the new combination has a better validation result, it is replaced as the answer so far. This method is very similar to the *Simulated Annealing* method. The only difference is in the way of generating neighboring states.

◇ **Bayesian Optimization:** It is a black-box optimization algorithm. It creates a probabilistic model (called surrogate model) of the objective function; with each new point, the surrogate model is updated. To pick a new point for evaluating, an acquisition function is used. At each step, the next point is determined by the acquisition function and after evaluating that point the model is updated. We use the BayesianOptimization library for python [29] and pass the *random_count* and *iter_count* to it. When the *is_local* variable is true, only values in the *locality_range* of best answer so far is considered.

◇ **SPSA:** SPSA is a stochastic approximation algorithm that is capable of finding the global optimum. At each step, it approximates the gradient by measuring the objective function at only two points. Then, it moves in the direction that minimizes the error. The fact that SPSA is independent of the dimension of the features makes it a great choice for our application where data have more than one feature and computing the gradient for all these takes a lot of time. SPSA takes three parameters, *alpha_decay*, *gamma_decay*, and *steps*. The details for this method are presented in Algorithm 4.

◇ **Combined Search Strategy**: Our approach uses a combination of all the methods discussed. SPSA method is used to explore the space as broadly as possible in a short time frame. After that, the Bayesian Optimization is performed to explore the space around the SPSA's result. Finally, to find a local optima, our algorithm performs

Algorithm 4: SPSA method

Input: steps, gamma_decay, alpha_decay
Output: A vector of bandwidths
while *iter* < *steps* **do**
  $c_{iter} = \frac{1}{10}(\text{gamma\_decay}^{(iter+1)})$
  if *base is None* **then**
    $a_{iter} = \frac{1}{\max(best\_error, 10^{-5})}(\text{alpha\_decay}^{(iter+1)})$
  else
    $a_{iter} = \frac{1}{\max(base, 10^{-5})}(\text{alpha\_decay}^{(iter+1)})$
  end
  Δ = a random perturbation vector
  $g_{iter} = \frac{evaluate(state + c_{iter}\Delta) - evaluate(state - c_{iter}\Delta)}{2c_{iter}\Delta}$
  if *base is None* **then**
    base = max value in g_{iter} vector
  end
  state = state - $(a_{iter} \times g_{iter})$
end
return state

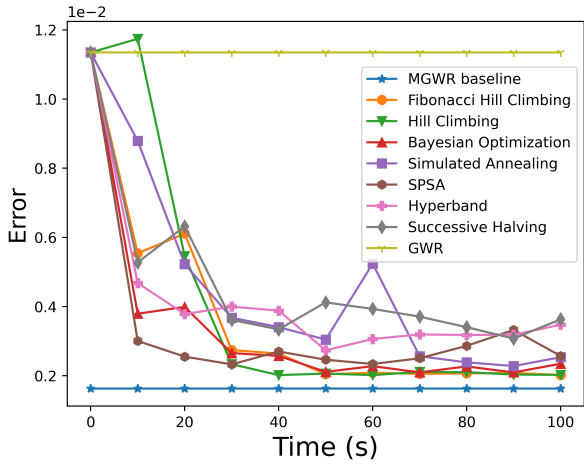


Figure 6: Timeline of error reduction for different bandwidth search methods.

the Hill Climbing search. Combining different bandwidth search methods is founded by analyzing the error improvement of each method over time. Figure 6 shows the error reduction for different methods over 100 seconds of time. The x-axis represents a timeline, from $t = 0$ to $t = 100$, and the y-axis represents the error value. The figure shows GWR as the fixed upper bound (at error $1.15x10^{-2}$) and MGWR as the fixed lower bound (at error $0.16x10^{-2}$), and every method starts with error equals GWR error and converges to an error close to MGWR. It is noticeable that different methods converge to the best error (MGWR error) at different speeds. This guides the *combined search strategy* to use the faster convergence methods first.

B Implementation Notes

B.1 Library Notes

For the multiprocessing, we used the *multiprocessing* library in Python. We set the number of learners to the minimum value of the number of sections and the number of CPU threads. The number of learners indicates the number of pools to create. Each pool is responsible for a data section. For the spatial data, features are separated from location attributes and these two properties are passed to the spatial model. For the random forest, the features and locations are concatenated and used as the training data.

PySAL implementation of GWR does not allow multiple prediction calls. This is specifically problematic for the pipeline method where a prediction call happens during the training (to compute the residuals) and one prediction call for the test data. Due to this problem, if GWR is passed as the spatial module, after the prediction is done in the training process, we train the spatial module again. This does not affect the accuracy as GWR is deterministic and produces the same result given the same input. However, the time reported for A-GWR using GWR module can be further improved using another implementation.

B.2 S-MGWR Notes

S-MGWR evaluates the bandwidths using k-fold cross-validation. For the k-fold cross-validation, where $k > 1$, for each set of bandwidths, the validation error is computed and the mean of k values is reported as the error for the bandwidths set. There is also a feature called *information sharing* that can be enabled or disabled when S-MGWR is employed within A-GWR. If information sharing is enabled, different sections can interact and pass information to each other. In essence, this warm starts the bandwidth search and accelerates the optimization. For example, when a section finds the best set of bandwidths, any section that has not started the search process can use that set of bandwidths as the initial search point. This helps a faster convergence as for the same dataset, different data sections are expected to have similar localities. We used *Manager* module from the *multiprocessing* library in Python for communication between our different processes.

B.3 Experiments Notes

We used different settings and search algorithms in S-MGWR for different experiments. The following indicates the parameter values of bandwidth search algorithms for different experiments. For Table 1 we use *simulated_annealing* (3, 0.97, 180, 125, 0, 0.2), *hill_climbing* (185, 110, 0, 0.1), *hyperband* (243, 64, 3), *SPSA* (60), *fibonacci_hill_climbing* (200, 115, 0, 0.1), and *bayesian_optimization* (true, 10, 15, 20). For our combined search strategy method, we use *SPSA* (9), *bayesian_optimization* (false, -, 60, 60), *hill_climbing* (75, 45, 0, 0.1). For Table 5, that shows the caching experiment, we use the following combination: *hyperband* (27, 64, 3), *simulated_annealing* (3, 0.97, 20, 15, 0, 2), and *hill_climbing* (100, 50, 0,1).

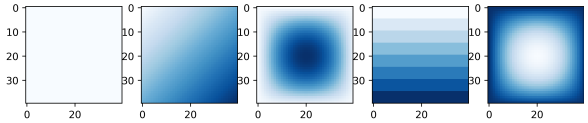


Figure 7: Illustration of the covariate functions. β_0 to β_4 are displayed from left to right, respectively

B.4 Dataset Notes

The covariate surface functions (β_0 to β_4) for Equation 5 are defined as follows:

$$\beta_0 = 3, \quad \beta_1 = 1 + \frac{1}{12}(u+v), \quad \beta_2 = 5 + \frac{1}{4}\lfloor\frac{u}{5}\rfloor, \tag{6}$$
$$\beta_3 = 1 + \frac{1}{\frac{l}{2}^2}\left[\frac{l}{4}^2 - (\frac{l}{4} - \frac{u}{2})^2\right]\left[\frac{l}{4}^2 - (\frac{l}{4} - \frac{v}{2})^2\right],$$
$$\beta_4 = l - \frac{1}{500}\left[\frac{l}{4}^2 - (\frac{l}{4} - \frac{u}{2})^2\right]\left[\frac{l}{4}^2 - (\frac{l}{4} - \frac{v}{2})^2\right].$$

Where u_i and v_i are the horizontal and vertical location of each point in the grid, $0 \leq u, v < l$. Figure 7 is an illustration of the surface functions.

For the large datasets (New York City Airbnb Open Data and House Sales in King County, USA) we split the data randomly to train and test splits. Training dataset contains 80 percent of the data and the rest are assigned to the test set. For other datasets, due to their small size, we created 5 different random splits with the same 80%-20% ratio. For each set, we used a different preset random seed (seeds from 1 to 5 were used) and for each seed, the data is randomly divided into the train and the test set. The results reported on these datasets is the mean of the results over each different split. We also verified that the standard errors are reasonably small.