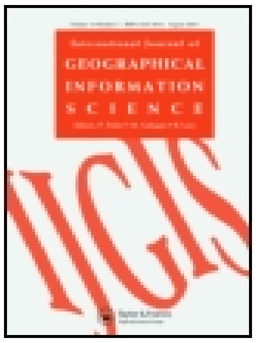

International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: https://www.tandfonline.com/loi/tgis20

Computational improvements to multi-scale geographically weighted regression

Ziqi Li & A. Stewart Fotheringham

To cite this article: Ziqi Li & A. Stewart Fotheringham (2020): Computational improvements to multi-scale geographically weighted regression, International Journal of Geographical Information Science, DOI: 10.1080/13658816.2020.1720692

To link to this article: https://doi.org/10.1080/13658816.2020.1720692

INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE
https://doi.org/10.1080/13658816.2020.1720692

RESEARCH ARTICLE

Computational improvements to multi-scale geographically weighted regression

Ziqi Li and A. Stewart Fotheringham

Spatial Analysis Research Center, School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ, USA

ABSTRACT
Geographically Weighted Regression (GWR) has been broadly used in various fields to model spatially non-stationary relationships. Multi-scale Geographically Weighted Regression (MGWR) is a recent advancement to the classic GWR model. MGWR is superior in capturing multi-scale processes over the traditional single-scale GWR model by using different bandwidths for each covariate. However, the multiscale property of MGWR brings additional computation costs. The calibration process of MGWR involves iterative back-fitting under the additive model (AM) framework. Currently, MGWR can only be applied on small datasets within a tolerable time and is prohibitively time-consuming to run with moderately large datasets (greater than 5,000 observations). In this paper, we propose a parallel implementation that has crucial computational improvements to the MGWR calibration. This improved computational method reduces both memory footprint and runtime to allow MGWR modelling to be applied to moderate-to-large datasets (up to 100,000 observations). These improvements are integrated into the *mgwr* python package and the MGWR 2.0 software, both of which are freely available to download.

ARTICLE HISTORY
Received 15 June 2019
Accepted 21 January 2020

KEYWORDS
Geographically weighted regression; multiscale; local modelling; spatial analysis; parallel computing

1. Introduction

In recent decades, spatially-varying coefficient (SVC) models have seen increased use in investigating potentially spatially non-stationary relationships. Geographically weighted regression (GWR) is one of the most popular and widely used SVC techniques (Fotheringham *et al*. 2002). It incorporates a data-borrowing mechanism at each location to obtain location-specific parameter estimates. Other alternative SVC frameworks include Bayesian spatially varying coefficient models (Gelfand *et al*. 2003, Finley 2011) and spatial eigenvector filtering (Murakami and Griffith 2015). Despite different model formulations and calibrations, all three methods are reported in the literature to yield similar local parameter estimates (Finley 2011, Oshan and Fotheringham 2018, Wolf *et al*. 2018). However, GWR has been the most widely adopted method due to its interpretability and accessibility. Various applications of GWR can be found in a wide range of fields, such as house price modelling (Bitter *et al*. 2007), atmospheric science (Hu *et al*. 2013), criminology (Troy *et al*. 2012) and public health (Comber *et al*. 2011).

CONTACT Ziqi Li lziqi@asu.edu
Supplemental data of this article can be accessed here.

Classic GWR is considered as a single-scale model that is based on one bandwidth parameter which controls the amount of distance-decay in weighting neighbouring data around each location. The single bandwidth in GWR assumes that processes (relationships between the response variable and the predictor variables) all operate at the same scale. The optimal bandwidth in GWR can be considered as the best 'average' bandwidth across all the covariate-specific processes. However, this posits a limitation in modelling potentially multi-scale processes which are likely to occur in the real world. For example, the measured ambient temperature of a location is affected by the built environment, regional weather and global warming, all of which operate at different scales. A recent advancement to GWR, termed Multi-scale GWR (MGWR), removes the single bandwidth assumption and allows covariate-specific bandwidths to be optimized (Fotheringham *et al*. 2017). This results in each parameter surface being allowed to have a different degree of spatial variation, reflecting variation across covariate-specific processes. In this way, MGWR has the capability to differentiate local, regional and global processes by optimizing a different bandwidth for each covariate. Additionally, bandwidths in MGWR become explicit indicators of the scale at which various processes operate. An illustration of local, regional and global spatial processes can be seen in Figure 1. In GWR, because only one bandwidth is used for all covariates, all processes are assumed to have similar spatial variation and GWR will over-smooth local process and under-smooth global processes. However, in MGWR, with a correctly specified bandwidth for each covariate, the local (global) processes can be accurately captured by small (large) bandwidths.

The advance from a single-scale local model (GWR) to a multi-scale local model (MGWR) brings flexibility and better model fit at a cost of additional computation. The latter arises because multi-scale models have more parameters to be estimated than single-scale models. MGWR, which is formulated as a Generalized Additive

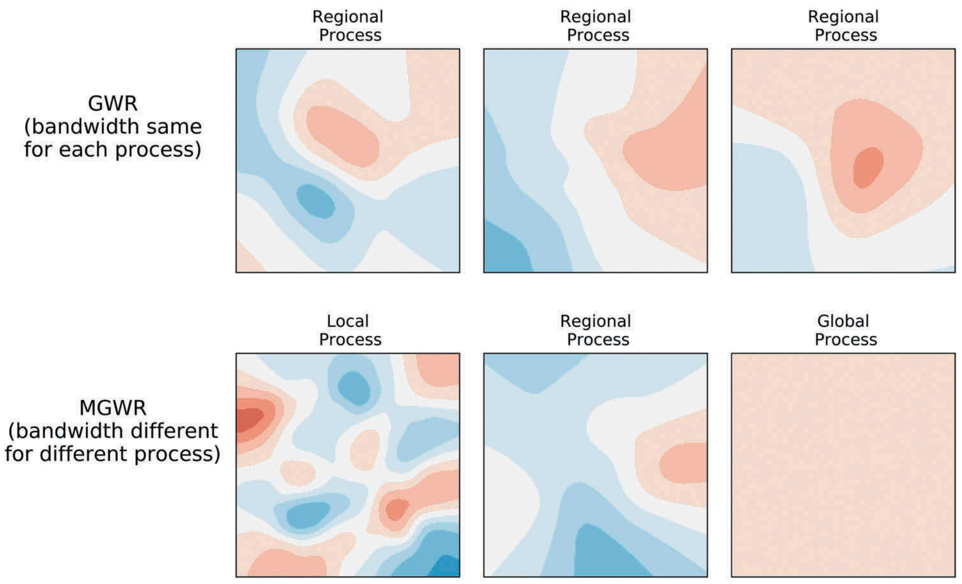

Figure 1. Illustration of local, regional and global spatial processes in GWR and MGWR.

Model (Buja *et al*. 1989, Hastie and Tibshirani 1990), utilizes a back-fitting algorithm for calibration which involves many univariate GWR calibrations. GWR calibration is not cheap in terms of computational cost since locally weighed regressions need to be calibrated at all locations. From the existing literature, studies using GWR often have fewer than 15,000 units of analysis (Li *et al*. 2019). Two major reasons why GWR has not generally been applied to large-scale data are (1) memory limitations of the hardware; and (2) inefficiency of the available software. Though there have been previous efforts to improve GWR calibration efficiency (Harris *et al*. 2010, Zhang 2010, Tran *et al*. 2016), few of these have obtained drastic improvements or developed accessible operational software. Recently, however, a parallel GWR computational improvement algorithm called FastGWR has been developed to solve various computational issues in GWR calibration (Li *et al*. 2019). Based on an example dataset with 15,000 spatial locations, FastGWR is around 350 times faster than the widely used R package *GWmodel* on a normal desktop without losing accuracy in parameter estimation. Additionally, the FastGWR parallel implementation addresses memory limitation and enables calibration of a GWR model with hundreds of thousands of locations within a reasonable time on a desktop machine. This brings new opportunities for geographically weighted regression models to be applied to large datasets. Because MGWR involves many GWR calibrations, this development can naturally benefit MGWR calibration by lowering runtime and memory.

In this study, our objective is to develop an efficient implementation of the MGWR model and to provide the spatial local modelling community with fast, scalable, reliable and accessible software that can be used in applied studies. This paper is organized as follows. First, we review the basics of the MGWR model which includes its back-fitting calibration process and statistical inference. Second, the computational challenges of MGWR are presented. Third, improvements to both the speed and memory of MGWR are described. Fourth, we propose a data-generating process to synthesize a dataset used for evaluating MGWR computation. Fifth, comparisons and evaluations of MGWR computation are presented. The paper concludes with possible future directions.

2. MGWR basics

2.1. MGWR formulation under GAM framework

A Generalized Additive Model (GAM) is formulated as

$$\boldsymbol{y} = \boldsymbol{f_1}(\boldsymbol{X_1}) + \boldsymbol{f_2}(\boldsymbol{X_2}) + \ldots + \boldsymbol{f_k}(\boldsymbol{X_k}) + \varepsilon \tag{1}$$

where $\boldsymbol{y}$ is a column vector of a response variable, $\boldsymbol{f_1}(\boldsymbol{X_1}), \cdots \boldsymbol{f_k}(\boldsymbol{X_k})$ are additive components which are smooth functions of covariates, and ε is a column vector of i.i.d. error (Hastie and Tibshirani 1990). The formulation of MGWR falls under this GAM framework (Fotheringham *et al*. 2017). The response variable $\boldsymbol{y}$ is the data observed over a spatial surface and $\boldsymbol{f_1}, \cdots \boldsymbol{f_k}$ are spatial additive components estimated with covariate-specific bandwidths, so MGWR can be formulated as

$$\boldsymbol{y} = \boldsymbol{f_{bw_1}}(\boldsymbol{X_1}) + \boldsymbol{f_{bw_2}}(\boldsymbol{X_2}) + \ldots + \boldsymbol{f_{bw_k}}(\boldsymbol{X_k}) + \varepsilon \tag{2}$$

and

$$\boldsymbol{f_{bw_j}} = \begin{pmatrix} \beta_{1j}x_{1j} \\ \beta_{2j}x_{2j} \\ \vdots \\ \beta_{nj}x_{nj} \end{pmatrix} \tag{3}$$

where x_{ij} is j^{th} covariate $\boldsymbol{X_j}$ at location i, and β_{ij} is MGWR local parameter of the j^{th} covariate location i. The response variable can be viewed as the sum of multiple spatial layers. To estimate local parameters β, we could re-arrange Equation (2) and knowing ε is independent of $\boldsymbol{X_j}$ gives the following component-wise conditional expectation.

$$\boldsymbol{f_{bw_j}} = E\left(\boldsymbol{y} - \sum_{\boldsymbol{p \neq j}} \boldsymbol{f_{bw_p}} - \varepsilon|\boldsymbol{X_j}\right) = E\left(\boldsymbol{y} - \sum_{\boldsymbol{p \neq j}} \boldsymbol{f_{bw_p}}|\boldsymbol{X_j}\right) = \boldsymbol{A_j}\left(\boldsymbol{y} - \sum_{\boldsymbol{p \neq j}} \boldsymbol{f_{bw_p}}\right) \tag{4}$$

where $\boldsymbol{A_j}$ is $E(\cdot|\boldsymbol{X_j})$ which can be viewed as a projection (hat) matrix from a univariate GWR model of $\boldsymbol{X_j}$ that maps $\boldsymbol{y} - \sum_{\boldsymbol{p \neq j}} \boldsymbol{f_{bw_p}}$ to $\widehat{\boldsymbol{f_{bw_j}}}$This GWR hat matrix $\boldsymbol{A_j}$ is expressed as

$$\boldsymbol{A_j} = \begin{pmatrix} x_{1j}(\boldsymbol{X_j^T W_1 X_j})^{-\boldsymbol{1X_j^T W_1}} \\ \cdots \\ x_{nj}(\boldsymbol{X_j^T W_n X_j})^{-\boldsymbol{1X_j^T W_n}} \end{pmatrix}_{n \times n} \tag{5}$$

where $\boldsymbol{W_i}$ is a diagonal spatial weight matrix computed with covariate-specific bandwidth and a kernel function (e.g. bi-square or Gaussian). Putting all the additive components in Equation (4) into a matrix form gives the following normal equations:

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{A_1} & \cdots & \boldsymbol{A_1} \\ \boldsymbol{A_2} & \boldsymbol{I} & \cdots & \boldsymbol{A_2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A_k} & \boldsymbol{A_k} & \cdots & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{f_{bw_1}} \\ \boldsymbol{f_{bw_2}} \\ \vdots \\ \boldsymbol{f_{bw_k}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A_1} \\ \boldsymbol{A_2} \\ \vdots \\ \boldsymbol{A_k} \end{bmatrix} \boldsymbol{y} \tag{6}$$

Equation (6) can be written in abbreviated form as

$$\boldsymbol{Pf} = \boldsymbol{Qy} \tag{7}$$

where $\boldsymbol{P}$ is an nk by nk matrix and $\boldsymbol{Q}$ matrix is an nk by n matrix. On rearranging, this gives additive components $\boldsymbol{f_{bw_1}}, \cdots \boldsymbol{f_{bw_k}}$ as

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{f_{bw_1}} \\ \boldsymbol{f_{bw_2}} \\ \vdots \\ \boldsymbol{f_{bw_k}} \end{bmatrix} = \boldsymbol{P^{-1}Qy} \tag{8}$$

provided that matrix $\boldsymbol{P}$ is invertible. Then, covariate-specific hat matrices can be obtained by

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \\ \vdots \\ \boldsymbol{R_k} \end{bmatrix} = \boldsymbol{P}^{-1}\boldsymbol{Q} \tag{9}$$

The normal equations in Equation (6) are not described in the MGWR literature but are useful as they provide a closed form of local parameter estimates and their variances as derived in MGWR. From a practical perspective, directly solving Equation (8) takes a time cost of $O(n^3k^3)$ which is feasible for small datasets but is often prohibitive for even moderately large datasets. Another issue is that the solution from Equation (8) assumes that the smoothing matrices $\boldsymbol{A_j}$ are known *a priori*. In the context of MGWR, this means that the bandwidth of each covariate is known. However, this piece of information is usually unknown and MGWR model calibration relies on a data-driven bandwidth search routine to find an optimal-fitted model that minimizes a goodness-of-fit criterion (e.g. AICc, cross-validation, etc.).

2.2. MGWR calibration

The solution to solve Equation (8) is to use the back-fitting process introduced in Buja *et al*. (1989) and explained in Hastie and Tibshirani (1990). Back-fitting estimators will converge to the solution in Equation (8), provided that the inverse of matrix $\boldsymbol{P}$ exists. Fotheringham *et al*. (2017) incorporated this back-fitting algorithm into the calibration process for MGWR with automatic bandwidth searching. The detailed calibration process is as follows (see also Algorithm 1 below). Firstly, the parameter estimates at all locations are initialized from a GWR model containing the response variable and all covariates. Fitted additive terms $\widehat{\boldsymbol{f}}_{\boldsymbol{1}\ldots\boldsymbol{k}}$ are computed accordingly by element-wise multiplying parameter estimates with covariates. The current model residuals $\widehat{\varepsilon}$ can be computed by $\widehat{\varepsilon} = \boldsymbol{y} - \widehat{\boldsymbol{y}}$. Then, a univariate GWR model is calibrated using the residuals $\widehat{\varepsilon}$ plus the first additive term $\widehat{\boldsymbol{f}}_{\boldsymbol{1}}$ as a response variable and first covariate $\boldsymbol{X_1}$.

$$\text{Univariate GWR} : \widehat{\boldsymbol{f}}_{\boldsymbol{1}} + \widehat{\varepsilon} \sim \boldsymbol{X_1} \tag{10}$$

The optimal bandwidth found in the univariate GWR model then temporarily become the optimal bandwidth bw_1 for the first covariate. With that bandwidth bw_1, $\widehat{\boldsymbol{f}}_{\boldsymbol{1}}$ and residuals $\widehat{\varepsilon}$ are updated. This procedure is then repeated for covariate $\boldsymbol{X_2}$ to update the second additive term $\widehat{\boldsymbol{f}}_{\boldsymbol{2}}$ and residuals $\widehat{\varepsilon}$. The process continues in this way through to the final covariate $\boldsymbol{X_k}$ when $\widehat{\boldsymbol{f}}_{\boldsymbol{k}}$ and $\widehat{\varepsilon}$ are updated. This finishes the first-round iteration and the second-round starts from the first additive term using the new values for the estimated $\widehat{\boldsymbol{f}}_{\boldsymbol{k}}$ and $\widehat{\varepsilon}$. The iterations continue until the change in a convergence indicator (such as the residual sum of squares) becomes sufficiently small between successive iterations. When the convergence threshold is met, parameter estimates and the final set of bandwidths $bw_{1\ \ldots\ k}$ can be obtained from the last iteration of back-fitting.

Algorithm 1: MGWR Back-fitting Calibration

```
1: Calibrate a GWR model of $\boldsymbol{y} \sim \boldsymbol{X}$ and obtain fitted additive terms $\widehat{\boldsymbol{f}}_{\boldsymbol{1} \ldots \boldsymbol{k}}$ and model
   residual $\widehat{\varepsilon}$.
2: Do until $\widehat{\boldsymbol{f}}_{\boldsymbol{1} \ldots \boldsymbol{k}}$ converge:
3:     For each term $j$ from $1$ to $k$:
           Calibrate univariate GWR model of $\widehat{\boldsymbol{f}}_{\boldsymbol{j}} + \widehat{\varepsilon} \sim \boldsymbol{X}_{\boldsymbol{j}}$ and get optimal bandwidth $bw_j$,
           new fitted
           term $\widehat{\boldsymbol{f}}_{\boldsymbol{j}}^*$, and residual $\widehat{\varepsilon}^*$.
           Update $\widehat{\boldsymbol{f}}_{\boldsymbol{j}} \leftarrow \widehat{\boldsymbol{f}}_{\boldsymbol{j}}^*$ and $\widehat{\varepsilon} \leftarrow \widehat{\varepsilon}^*$.
4:     End for
5:     End do
```

2.3. MGWR inference

Yu *et al*. (2019) analytically derive the hat matrix for MGWR and the standard errors of the MGWR parameter estimates, which provides an inferential framework for MGWR under a GAM framework. In this section, we will follow the notations used in Yu *et al*. (2019) to describe the computation of inference. The principal is to express each fitted component as follows:

$$\widehat{\boldsymbol{f}}_{\boldsymbol{j}} = \boldsymbol{R}_{\boldsymbol{j}} \boldsymbol{y} \tag{11}$$

where $\boldsymbol{R}_{\boldsymbol{j}}$ is an n by n covariate-specific hat matrix that maps response variable $\boldsymbol{y}$ to each fitted additive component $\widehat{\boldsymbol{f}}_{\boldsymbol{j}}$. Then, the covariance of $\widehat{\boldsymbol{f}}_{\boldsymbol{j}}$ can be obtained by $\boldsymbol{R}_{\boldsymbol{j}} \boldsymbol{R}_{\boldsymbol{j}}^{\boldsymbol{T}} \sigma^2$ where σ^2 is the error variance of the MGWR model. The covariate-specific hat matrix $\boldsymbol{R}_{\boldsymbol{j}}$ is computed within the back-fitting process after each univariate GWR by updating

$$\boldsymbol{R}_{\boldsymbol{j}} \leftarrow \boldsymbol{A}_{\boldsymbol{j}} \left(\boldsymbol{I} - \sum\nolimits_{\boldsymbol{p} \neq \boldsymbol{j}}^{\boldsymbol{k}} \boldsymbol{R}_{\boldsymbol{p}} \right) \tag{12}$$

where $\boldsymbol{A}_{\boldsymbol{j}}$ is the hat matrix of each univariate GWR model (Equation (4)). Conveniently, let $\boldsymbol{I} - \sum\nolimits_{\boldsymbol{p} \neq \boldsymbol{j}}^{\boldsymbol{k}} \boldsymbol{R}_{\boldsymbol{p}}$ be denoted as $\boldsymbol{R}_{\boldsymbol{j}}^{-}$ so that Equation (12) becomes

$$\boldsymbol{R}_{\boldsymbol{j}} \leftarrow \boldsymbol{A}_{\boldsymbol{j}} \boldsymbol{R}_{\boldsymbol{j}}^{-} \tag{13}$$

Then, the MGWR hat matrix $\boldsymbol{S}$ that maps $\boldsymbol{y}$ onto $\widehat{\boldsymbol{y}}$ can be obtained by summing the $\boldsymbol{R}$ matrices from the last iteration in the back-fitting once the convergence threshold is satisfied.

$$\boldsymbol{S} = \sum\nolimits_{\boldsymbol{1}}^{\boldsymbol{k}} \boldsymbol{R}_{\boldsymbol{j}} \tag{14}$$

With the $\boldsymbol{R}$ and $\boldsymbol{S}$ matrices computed, important model diagnostics can then be obtained as follows:

(1) Covariate-specific Effective Number of Parameters (ENP):

$$ENP_j = tr\left(\boldsymbol{R}_{\boldsymbol{j}}\right) \tag{15}$$

(2) Model Effective Number of Parameters:

$$ENP_{model} = tr(\boldsymbol{S}) \tag{16}$$

(3) Model *AICc*:

$$AIC_c = 2n\ln\left(\frac{RSS}{n}\right) + n\ln(2\pi) + n\left(\frac{n + tr(\boldsymbol{S})}{n - 2 - tr(\boldsymbol{S})}\right) \tag{17}$$

where *RSS* is the residual sum of squares.

(4) The standard errors of the parameters can be obtained by:

$$SE\left(\widehat{\beta}_j\right) = \sqrt{diag\left(\boldsymbol{C}\boldsymbol{C}^T\hat{\sigma}^2\right)} \tag{18}$$

where $\boldsymbol{C} = \left[diag(\boldsymbol{X_j})\right]^{-1}\boldsymbol{R_j}$ and $\left[diag(\boldsymbol{X_j})\right]^{-1}$ is the inverse of a diagonal matrix with $\boldsymbol{X_j}$ being its diagonal elements.
Detailed descriptions can be found in Yu *et al*. (2019).

2.4. Available software for calibrating MGWR model

Currently, there are two software packages that are available to calibrate an MGWR model, and both are actively maintained. One is *GWmodel* in the R environment (Gollini *et al*. 2015)[1] which provides an array of geographically weighted models. With the latest update in February 2019 (version 2.0–8), *GWmodel* is able to perform an MGWR calibration with inference based on the work of Fotheringham *et al*. (2017) and Yu *et al*. (2019). It also has the capability of calibrating Parameter-specific Distance Metrics (PSDM) models (Lu *et al*. 2018), which is a special use case of MGWR when different distance metrics are desired. A detailed comment on PSDM and MGWR can be found in Oshan *et al*. (2019b). The second is MGWR 1.0 software released in November 2018. It is built on the *mgwr* python package[2] (Oshan *et al*. 2019a) and has a user-friendly graphical interface and is available for both MacOS and Windows platforms. The computational improvements developed in this paper are integrated into the *mgwr* python package (version 2.1.0) and built into the MGWR 2.0 software[3] which supersedes the previous MGWR 1.0 version.

3. Computational cost of MGWR

MGWR is computationally intensive due to the iterative back-fitting algorithm that involves many univariate GWR calibrations. The total number of univariate GWR calibrations needed is $k \times d$, where k is the number of covariates and d is the number of iterations before convergence is reached. Though d is hard to predict, there are three general rules of thumb from experiments which show that:

(1) The greater the number of covariates, the more iterations are needed for convergence.

(2) Higher levels of multicollinearity among covariates increase the number of iterations needed before convergence.
(3) Local processes need more iterations to converge than regional or global processes.

Further analysis is needed to provide confirmatory evidence of these rules. In addition, standardization of both response and covariates is suggested before running the MGWR model (Fotheringham *et al.* 2017). This is not only for better comparison of parameter estimates and associated bandwidths, but it also speeds up convergence. Li *et al.* (2019) show that for a multivariate GWR model with k covariates, the time complexity of the GWR algorithm is $O(k^3n^2logn)$ where the $\log(n)$ comes from a golden-section search for the optimal bandwidth. Because MGWR only needs univariate GWR calibrations in its back-fitting, the time complexity of an MGWR calibration is approximately $O(kdn^2logn\ )$ as shown in Table 1. The computational cost of the inference calculation is much more intensive because after each univariate GWR during the back-fitting, a multiplication of two n by n matrices $\boldsymbol{A_j}$ and $\boldsymbol{R_j}^-$ (Equation (13)) is needed at a cost of $O(n^3)$ time complexity. In total, there would be $k \times d$ numbers of $O(n^3)$ operations needed. The time for updating $\boldsymbol{R_j}$ is negligible when n is relatively small (e.g. n < 5,000). However, as n increases, this operation does not scale well and will dominate the runtime of the back-fitting more than the parameter estimation procedure. Further evidence of this statement will be shown in the Results section. On the other hand, regarding the memory constraint, storing the $k$by n by n matrix $\boldsymbol{R}$ has a memory complexity of $O(kn^2)$. Figure 2 shows the theoretical memory allocation needed with varying n and k based on double-precision (64-bit for each number) matrices. Considering today's

Table 1. Computational complexity of MGWR calibration.

Back-fitting		Inference	
Time	Memory	Time	Memory
$O(kdn^2logn)$	$O(kn)$	$O(kdn^3)$	$O(kn^2)$

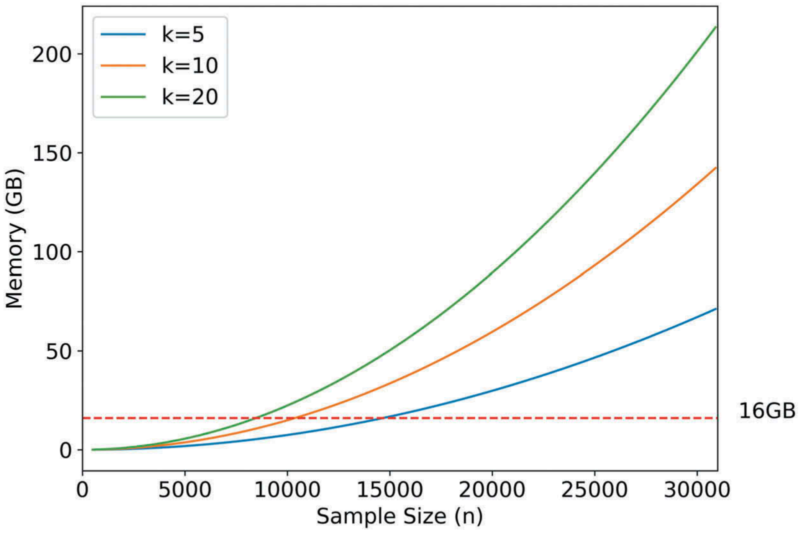

Figure 2. Theoretical memory demand of MGWR calibration with varying numbers of observations and covariates.

hardware, this will place a severe limitation in MGWR for calibrating a moderately large model (>10,000 locations). For example, a desktop computer with 16 GB of available memory would fail to calibrate an MGWR model with more than 15,000 locations and with more than 5 covariates. What is more, in practice, because intermediate results are needed to be cached during the calibration, we often need more memory space than the theoretical memory.

4. Solutions to address the computational issues in MGWR

MGWR calibration can be separated into two steps. The first is to estimate parameters and find the set of optimal bandwidths during a back-fitting procedure. The second is to compute MGWR inference to obtain parameter uncertainties, covariate-specific and overall model effective number of parameters (ENP) and model diagnostics (e.g. AICc). The first step of an MGWR calibration mainly involves univariate GWR calibrations; thus, here we can take advantage of the parallelization and optimization methods described in FastGWR (Li *et al*. 2019). For inference computation, we need to develop a novel parallel implementation that computes covariate-specific hat matrices $\boldsymbol{R_j}$ in column chunks to reduce the memory demand. We now describe these two developments.

4.1. MGWR back-fitting with FastGWR

Traditional GWR calibration consists of independent locally weighted least square regression at each location. Then, local statistics are combined to compute model diagnostics such as AICc to assess model fit. This process is repeated for different bandwidths to find which bandwidth yields the best model fit using, for example, AICc. FastGWR utilizes the parallelizable nature of GWR and computes all local statistics in parallel without storing large matrices such as the weight matrix and hat matrix. Then, AICc can be efficiently computed and compared across different bandwidths with the help of a golden-section search routine. The approach is described in detail in the context of MGWR. For a given bandwidth, there are two local statistics that are needed to compute AICc: the local influence value r_i (also the diagonal element of hat matrix) and the local residual $\hat{\varepsilon}_i$. Both can be computed as follows:

$$r_i = x_{ij}\left(\boldsymbol{X_j^T W_i X_j}\right)^{-\boldsymbol{1}}\boldsymbol{X_j^T W_i} \tag{19}$$

$$\hat{\beta}_{ij}{}^{*} = \boldsymbol{X_j}\left(\boldsymbol{X_j^T W_i X_j}\right)^{-\boldsymbol{1}}\boldsymbol{X_j^T W_i}\left(\widehat{\boldsymbol{f}}_{\boldsymbol{j}} + \widehat{\varepsilon}\right) \tag{20}$$

$$\hat{\varepsilon}_i{}^{*} \;= \hat{\beta}_{ij}x_{ij} - \hat{\beta}_{ij}{}^{*}x_{ij} + \hat{\varepsilon}_i \tag{21}$$

where $x_{ij}$is the measurement of the j^{th} covariate at location i, $\boldsymbol{X_j}$ is a column vector of the j^{th} covariate, $\boldsymbol{W_i}$ is a diagonal weight matrix (can be stored as a column vector to reduce memory) computed locally using the current bandwidth as a pre-specified kernel (e.g. bi-square or Gaussian), $\hat{\beta}_{ij}$ and $\hat{\varepsilon}_i$ are the parameter estimate and residual from the last iteration of back-fitting. The computation of three local statistics can be evenly assigned to multiple processors for parallel computing. Then, once all processors finish their assigned tasks, covariate-specific AICc values can be computed as

$$AICc = 2n\ln\left(\frac{\sum_1^n \hat{\varepsilon}_i{}^*}{n}\right) + n\ln(2\pi) + n\left(\frac{n + \sum_1^n r_i}{n - 2 - \sum_1^n r_i)}\right) \tag{21}$$

This procedure is repeated along with the golden-section routine as detailed in Algorithm 2.

Algorithm 2: MGWR Back-fitting Calibration using parallel implementation from FastGWR

```
1: Calibrate a GWR model of $\boldsymbol{y} \sim \boldsymbol{X}$ and obtain fitted additive terms $\widehat{\boldsymbol{f}_{1\ldots k}}$ and model
   residual $\widehat{\varepsilon}$.
2: Do until $\widehat{\boldsymbol{f}_{1\ldots k}}$ converge:
3:    For each term $j$ from $1$ to $k$:
4:       For $bw$ in golden-section search:
5:          Parallel computing local statistics $\hat{\varepsilon}_i{}^*$, $r_i$, and $\hat{\beta}_{ij}{}^*$ for all locations i.
6:          Compute and assess covariate-specific $AICc$.
7:       Obtain the optimal bandwidth as $bw_j$ with minimum $AICc$ and update $\widehat{\boldsymbol{f}_j} \leftarrow \widehat{\beta}_j^* \boldsymbol{X}_j$
         and $\hat{\varepsilon} \leftarrow \hat{\varepsilon}^*$
4: End for
5:    End do
```

4.2. Chunk-wise parallel computing of MGWR inference

From the previous discussion, it is known that the inference computation in MGWR requires a large memory allocation for the computation of the covariate-specific hat matrices $\boldsymbol{R}$ (dimension n by n by k), which can easily exceed the memory limit of a standard desktop computer; even for a moderately sized dataset. In this section, we propose a partitioning method that can compute the $\boldsymbol{R}$ matrix in small chunks of columns, each of which can easily fit into the memory of even a modest computer. From Equation (13) we can see that each covariate-specific hat matrix $\boldsymbol{R}_j$ is computed as $\boldsymbol{R}_j \leftarrow \boldsymbol{A}_j\boldsymbol{R}_j{}^-$ during back-fitting, where A_j is a univariate GWR hat matrix that projects $\boldsymbol{R}_j{}^-$ to $\boldsymbol{R}_j$with the dimension $n$by n, and $\boldsymbol{R}_j{}^-$ is $\boldsymbol{I} - \sum_{\boldsymbol{p}\neq \boldsymbol{j}}^{\boldsymbol{k}} \boldsymbol{R}_{\boldsymbol{p}}$ of shape $n$by n. From the property of a projection matrix, it is known that updating each column of $\boldsymbol{R}_j$ is independent of updating other columns of $\boldsymbol{R}_j$. This provides the opportunity to compute R_j by chunks of columns. Since both $\boldsymbol{A}_j$ and $\boldsymbol{R}_j{}^-$ are $n$by n matrices, in the case when both cannot fit into memory, we partition $\boldsymbol{A}_j$ by rows and $\boldsymbol{R}_j{}^-$ by columns to update a block of $\boldsymbol{R}_j$. The updating is illustrated in Figure 3 where we evenly split $\boldsymbol{A}_j$ and $\boldsymbol{R}_j{}^-$ into two chunks. This procedure is repeated in the back-fitting algorithm, and when the back-fitting converges we get a column chunk of $\boldsymbol{R}_j$, which can be denoted as $\boldsymbol{R}_{jc}$. Partial inference can be computed based on the chunk of $\boldsymbol{R}_{jc}$, such as the calculation of the effective number of parameters and the matrix $\boldsymbol{CC}^T$ in Equation (18) to further compute parameter uncertainties. After that, we iterate to the second chunk of $\boldsymbol{R}_j$, and obtain the corresponding partial inference. Each chunk can be computed independently in parallel to fully utilize a multi-processor system. Once all chunks are computed, full MGWR inference can be combined using partial inference obtained from each chunk, and the formulas are given as follows:

$$ENP_{jc} = \sum\nolimits_{c=1}^{q} partialtrace(\boldsymbol{R}_{jc}) \tag{22}$$

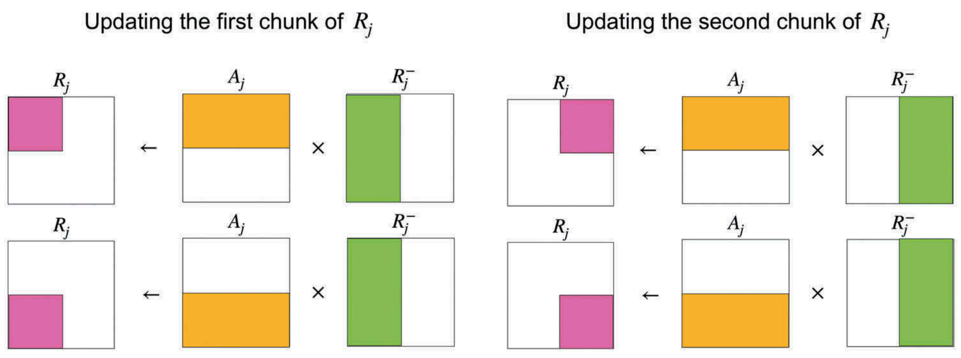
Updating the first chunk of R_j
Updating the second chunk of R_j
R_j
A_j
R_j^-
←
×

Figure 3. Illustration of the block update covariate-specific matrix R_j by chunks (e.g. the number of chunks = 2).

$$\boldsymbol{CC}_{\boldsymbol{jc}}^{\boldsymbol{T}} = \sum\nolimits_{c=1}^{q} \left(\frac{\boldsymbol{R}_{\boldsymbol{jc}}}{X_j}\right)^2_{axis=1} \tag{23}$$

where c is the current chunk and q is the total number of chunks and the subscript $axis = 1$ indicates that the summation is applied to columns and reduces to a column vector. Details are shown in Algorithm 3.

Algorithm 3: Chunk-wise parallel computing of MGWR inference

```
1: Initialize $\boldsymbol{R_j}$ from GWR, $ENP_j = 0$, and $\boldsymbol{CC_j^T} = 0_{n\times1}$
2: Parallel computing each chunk $c$ in the total number of $q$ chunks
3:    Given bandwidths history in back-fitting iterations from parameter estimation
4:        For each term $j$ from $1$ to $k$:
5:            Block-update $\boldsymbol{R_j}$ and get a column chunk of $\boldsymbol{R_j}$
6:            Update column chunk of $\boldsymbol{R_j}^-$
7: For each term $j$ from $1$ to $k$:
8:        $ENP_j = ENP_j + ENP_{jc}$
9:        $\boldsymbol{CC_j^T} = \boldsymbol{CC_j^T} + \boldsymbol{CC_{jc}^T}$
10:     End for
11: End do
```

Splitting covariate-specific hat matrices $\boldsymbol{R}$ into two chunks (q = 2) should reduce the memory needed by approximately half. For example, if the allocation of $\boldsymbol{R}$ needs 32 GB of memory, if we use the chunk-wise computation described above and split $\boldsymbol{R}$ into two chunks, then memory allocation needed will be reduced to around 16 GB. Splitting $\boldsymbol{R}$ into q chunks is expected to reduce memory by a factor of q. However, it worth noting that since $\boldsymbol{A_j}$ is not stored and will be computed repeatedly for each chunk of $\boldsymbol{R_j}$, it will add additional cost if we specify the number of chunks greater than 1. In this way, we need to decide the minimum number of chunks that can make the chunk of $\boldsymbol{R}$ fit into the available memory. A simple calculation to decide q would be

$$q = \frac{8 \times k \times n \times n}{m \times 1024^3} \tag{24}$$

where m is the available memory in the hardware, and we assume we use double-precision matrices (64-bit). For example, if the available memory is 16 GB on a machine, and we have an MGWR model with 40,000 data points and 10 covariates, q will need to be at least 15 chunks to avoid memory problems.

5. Testing dataset and environment

5.1. A data-generating process for deriving a test dataset

For testing the performance of the improved MGWR software, we develop a data-generating process (DGP) that can automatically synthesize multi-scale spatial processes for a varying number of covariates and observations. Detailed steps can be found in the supplementary materials. In this study, we generated 15 parameter surface grids with a dimension of 200 by 200 in a total of 40, 000 locations. The resulting parameter surfaces $\boldsymbol{b_1}, \boldsymbol{b_2}, \ldots, \boldsymbol{b_{15}}$ are spatial random fields with different heterogeneities, which can be seen in Figure 4. The response variable y^* is constructed by $\boldsymbol{y}^* = \sum_{j=1}^{15} \boldsymbol{b_j x_j} + \varepsilon$ where each covariate $\boldsymbol{x_j}$ and the error term ε are randomly drawn from the normal distribution *N(0, 1)*.

5.2. Testing environment

The single desktop used in comparing performance across MGWR software packages is equipped with an Intel i7-4790 4-core CPU at 3.60 GHz and 16 GB RAM. The Python environment used is version 3.6. The R environment used is version R 3.5.2. Runtimes are measured based on Python and R's built-in *time* modules. Memory usages are measured based on the *memory_profiler*[4] module for Python and the built-in *memory* module for R. To enhance reproducibility, all testing codes, datasets used and results can be found publicly in a Github repository.[5]

6. Results

6.1. Runtime comparison with the original MGWR implementation

First, we benchmark our improved algorithm with the original implementation of MGWR used in Fotheringham *et al.* (2017) to show that the improved method reduces various issues of computation. To demonstrate this, we reproduce MGWR models and record runtimes using the example datasets in Fotheringham *et al.* (2017).[6] Figure 5 shows the comparison of runtimes from MGWR 2.0 and runtimes explicitly reported in, Fotheringham *et al.* (2017) using two example datasets. The simulation 1 dataset used in, 2017 consists of three covariates with 625 observations. For this synthetic dataset, the calibration takes 4.8 min in Fotheringham *et al.* (2017, p. 1257), while it only takes 7.4 s after the improvements described here. For the other example used in that paper, the Irish potato famine dataset with 2317 locations and 9 covariates, the paper reported that the calibration process took 51 h (Fotheringham *et al.* 2017,p. 1260). In comparison, using the improved method proposed in this paper, the runtime is reduced to around 6 min, which is 510 times faster than the original implementation. For both examples, our improved method shows a crucial

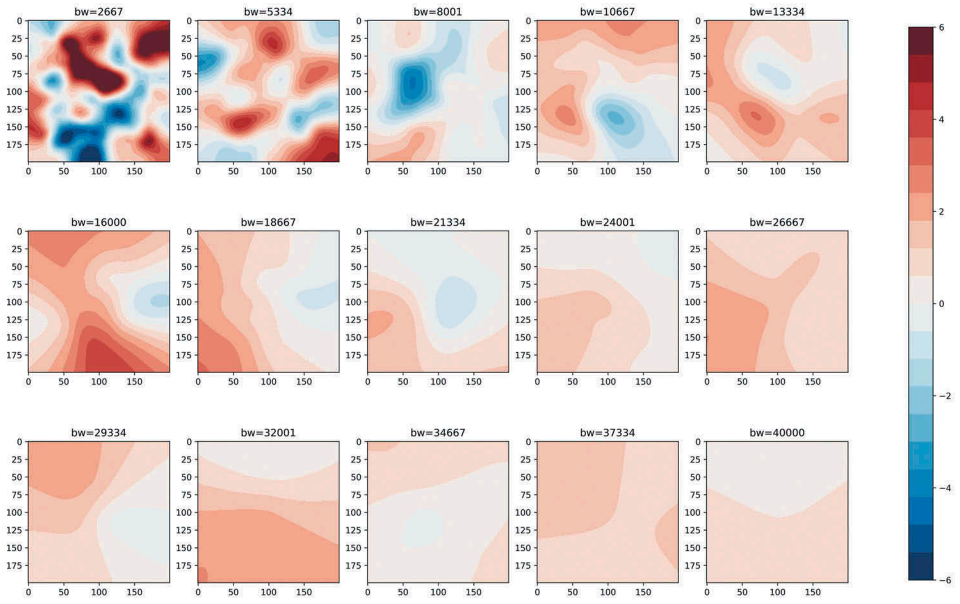

Figure 4. Synthesized known parameter surfaces with varying scales from local to global.

performance increase. Also, the speed-up is projected to be greater for more complicated models which contain more observations and covariates.

6.2. *Comparison against GWmodel*

Second, we compare our proposed improvement against *GWmodel* package's MGWR implementation. We randomly draw a subset of data points from the synthetic dataset described in Section 5.1 and calibrate MGWR models using both software packages to record the runtimes and maximum memory usage. Results are presented in Table 2. With a sample size of N = 1,000, MGWR 2.0 is around 16 times faster than *GWmodel*. When N = 2,000, MGWR is around 30 times faster than *GWmodel* and when N increases to 5,000, MGWR 2.0 is around 140 times faster than *GWmodel*. When the sample size increases to 10,000, *GWmodel* generates

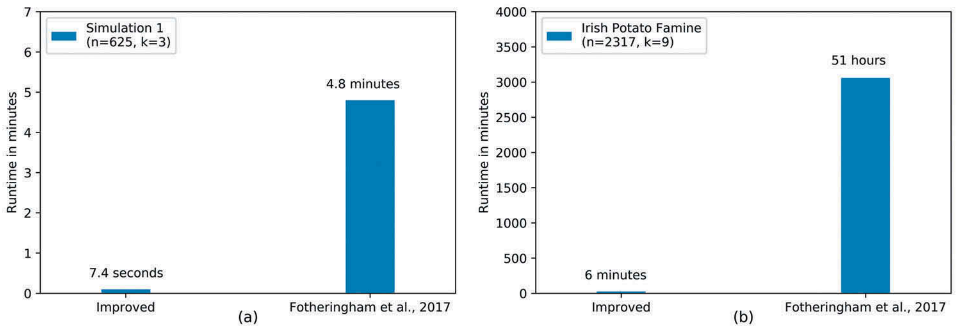

Figure 5. Runtime comparison based on two example datasets from Fotheringham *et al*. (2017).

Table 2. Runtime and memory comparison between MGWR 2.0 and *GWmodel*.

	MGWR calibration with inference (10 covariates)			
Sample size	MGWR 2.0			GWmodel
N	Time (Seconds)	Memory	Time	Memory
1000	50	1.3 GB	802	0.5 GB
2000	127	2.0 GB	3818	0.8 GB
5000	370	3.2 GB	52,119 (~15 h)	5.5 GB
10,000	2652	10 GB	Memory error	NA
15,000	10,068 (~3 h)	11 GB	Memory error	NA
20,000	23,236 (~6.5 h)	11 GB	Memory error	NA
40,000	301,126 (~4 day)	11 GB	Memory error	NA

a memory allocation error on the testing machine but MGWR 2.0 is able to complete the calibration within an hour. For sample size N = 20,000, MGWR 2.0 takes around 6.5 h to complete and for N = 40,000, MGWR 2.0 takes around 4 days to finish. This is because memory allocation exceeds the memory limit of the testing machine so the software has to use more chunks to split the inference computation in order to fit in the memory, which slows the calibration. If a machine with more memory were used, the calibration by MGWR 2.0 would be accelerated by several orders of magnitude. From a memory perspective, MGWR 2.0 caps out the memory usage at around 11 GB (approximate available memory on the test machine) when N is great than around 15,000 and the software starts to bring the chunk parameter into action to avoid memory allocation error.

6.3. Effectiveness of memory reduction in inference computation

With the improved implementation borrowed from FastGWR, memory is not an issue in the parameter estimation component of MGWR. In the inference component, we need to use the method described in Section 3.4 to separately compute covariate-specific hat matrices in chunks of columns. To demonstrate the effectiveness of partitioning this computation into chunks, we calibrate two MGWR models with sample sizes of N = 10,000 and N = 20,000 and 10 covariates, and record the memory usage with varying numbers of chunks. Figure 6(a) shows how memory is reduced when the number of chunks is increased. When the number of chunks is 1, the inference computation does not involve any partitioning and covariate-specific matrices are computed as described in Yu *et al*. (2019). When the number of chunks increases to 2, memory is reduced by approximately half, and this behaviour is consistent for both sample sizes. For example, when N = 20,000, the memory demand is reduced from around 40 GB to around 20 GB. When the number of chunks is increased to 4, the memory requirement is reduced further to around 10 GB. This implies that if a machine has a RAM of 16 GB, it is not able to compute MGWR inference for a sample size of 20,000 due to the memory needed, but if we use the improved method presented in this paper, we could increase the number of chunks and compute inference chunk-by-chunk to get around the memory constraint. However, increasing the number of chunks comes with the cost of increasing calibration time. Figure 6(b) shows how runtime is increased when the number of chunks is increased. As can be seen, when two chunks are being used, it only adds a small amount of extra runtime. When more chunks are used, runtime goes up but only slowly. Figure 6(a,b) combined show that using more chunks does not add too much extra computation time but does decrease memory demand drastically.

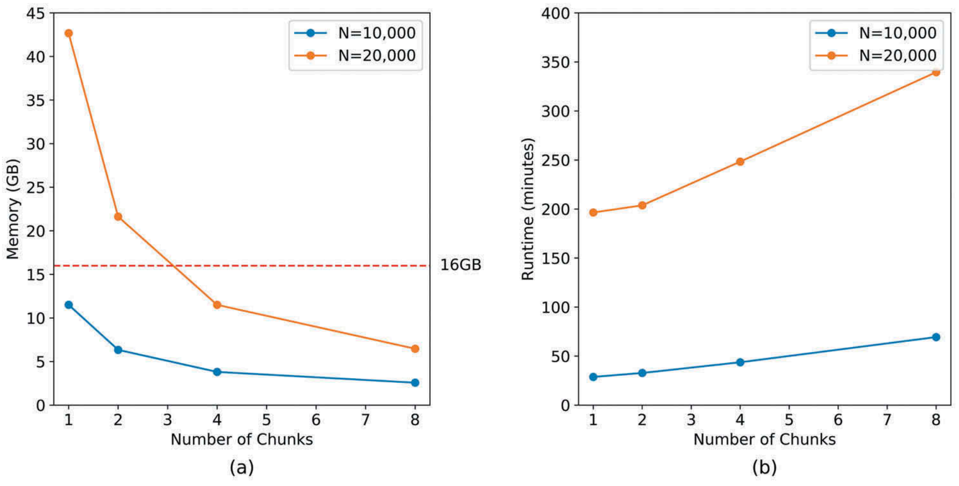

Figure 6. Memory demand and runtime change with varying numbers of chunks used.

6.4. On the scalability of MGWR parallelization

In this section, we examine the scalability of the parallelization used in the MGWR calibration. First, we calibrate MGWR models with sample sizes of N = 5,000 and N = 10,000 with varying numbers of cores. Since the parallelization mechanism is different for parameter estimation and inference, we separately record the runtimes for the two components. In Figure 7, stacked bar plots (a) and (c) show the actual runtimes, respectively, for parameter estimation and inference computation for the two sample sizes using 1 to 4 processors. It can be seen that when N = 5,000, parameter estimation takes more time than inference computation, while when N = 10,000, inference computation takes more time than parameter estimation. As more processors are used, runtimes decrease logarithmically for both sample sizes. The line plots in Figure 7(b,d) show the speed-up factors for parameter estimation, inference and a combination of both (total) based on runtime using a single processor so that the speed-up factor is 1 when using only 1 processor. Parameter estimation has strong scalability when using more processors, and using 4 processors generates a speed-up factor close to 3. Inference computation scales less well than parameter estimation but still the use of 4 processors generates a speed-up factor around 2. When combined, the MGWR calibration overall is around 2.4 times faster when using 4 processors than when using only a single processor. Since the testing machine only has four physical cores, it is hard to infer the scalability if a machine has more processors. Thus, we used another desktop machine with eight physical cores to examine both the compatibility and scalability of the MGWR parallelization. The results can be seen in Figure 8 where it can be seen that using 8 processors speeds-up the overall MGWR algorithm by a factor of 4.8. The speed-up factor of parameter estimation is around 6 and speed-up factor of inference computation is around 4. Generally speaking, the scalability of MGWR parallelization is reasonable. There are two reasons for not getting perfect linear scalability. First, based on Amdahl's law about parallel computing that the speed-up of a program is limited by its serial part, the back-fitting procedure of the MGWR calibration involves sequential updating of parameter estimates and residuals which reduces performance. Second, parallelization can be implemented by either multi-threading or multi-

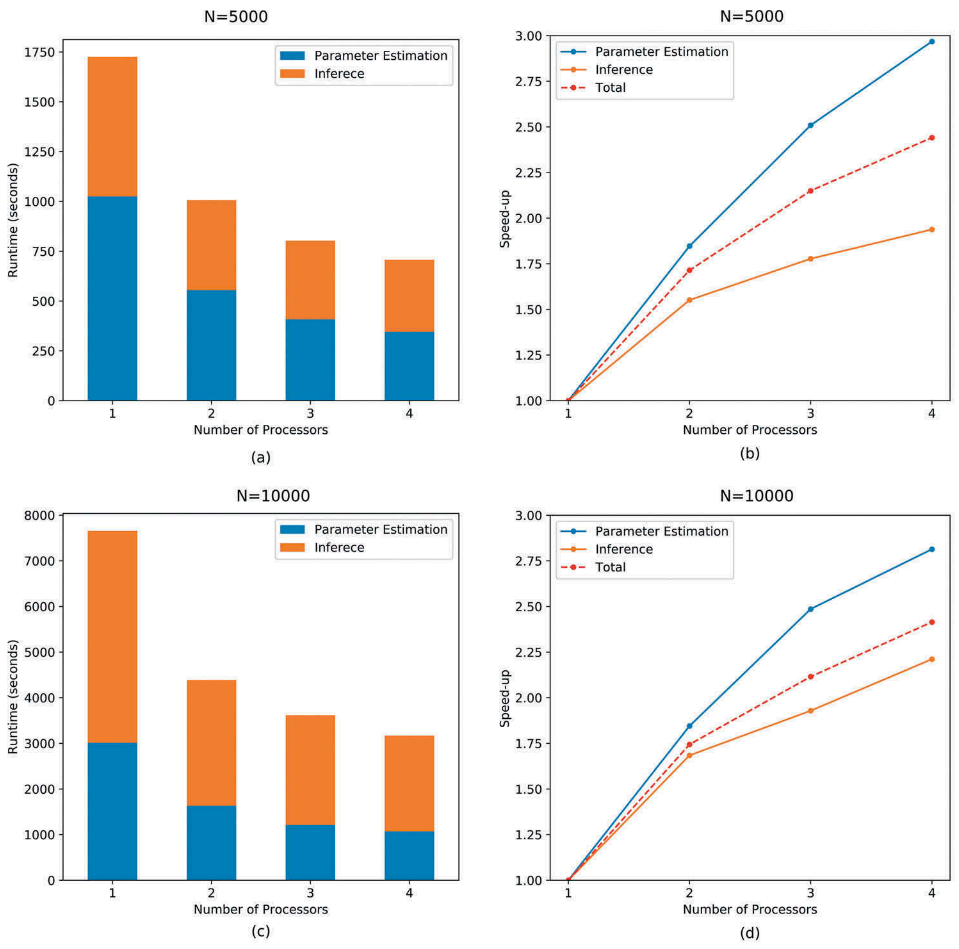

Figure 7. Runtime and speed-up factor of the new MGWR calibration for parameter estimation, inference and total calibration on a 4-processor machine based on a sample size of 10,000.

processing. Multi-processing is more desirable in this situation because multi-threading is limited by the Global Interpreter Lock (GIL) of *Python*. However, the downside of using multi-processing is that there is a noticeable amount of time needed for data copying to different processors, even though we limit this by minimizing the amount of data being copied.

6.5. Computational evaluation of MGWR

In this section, we evaluate the computation of MGWR using the synthesized dataset described in Section 5.1. We calibrate MGWR models including inference for a varying number of sample sizes and with 5, 10 and 15 covariates. Given that convergence rates may vary in empirical studies, we also include three categories of convergence rate which are: fast convergence (10 back-fitting iterations); moderate convergence (50 back-fitting iterations); and slow convergence (100 back-fitting iterations). Figure 9 shows the approximate time needed to complete an MGWR calibration (with parameter estimation and inference) under the three different

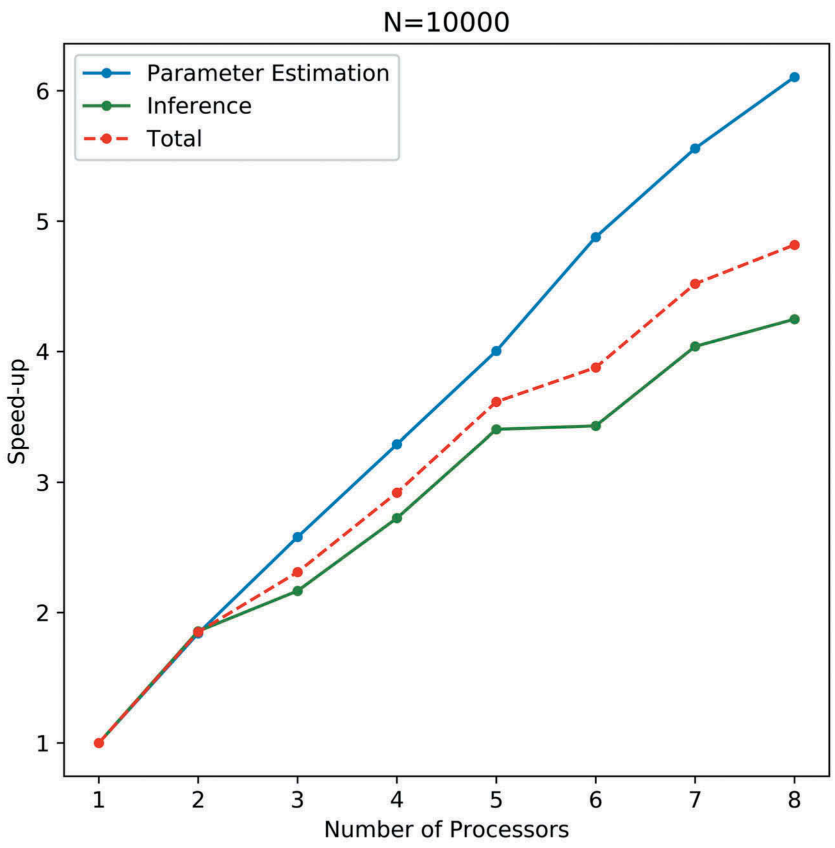

Figure 8. Speed-up factor of the new MGWR calibration for parameter estimation, inference and total calibration on an 8-processor machine based on a sample size of 10,000.

convergence rates and for varying numbers of both observations and covariates.[7] The blue areas on each figure show what model size can be fitted in under 10 min; the green areas show the model size that can be calibrated within 1 h; the yellow areas show the model size can be calibrated within 6 h; the red areas show the model size that can be calibrated within 1 day; and the purple areas show the model size needing more than 1 day for calibration. For example, for 10 covariates, a model with a sample size of 2000 can be fitted in under 10 min if it converges at a moderate rate. In general, we can see that MGWR is able to be calibrated on models with fewer than 20,000 observations within a reasonable time (less than a day) if it converges moderately. When the sample size is larger than 20,000 or more covariates are involved, it might take several days to calibrate an MGWR model on a normal desktop computer. However, using a workstation with more processors and memory will accelerate MGWR calibration several times so the task would probably be completed within 1 day. The ability of MGWR 2.0 to utilize more computing resources opens the possibility for handling up-to 100,000 observations in an MGWR calibration.

7. Conclusions

In this paper, we make critical computational improvements to the newly developed Multi-scale Geographically Weighted Regression (MGWR) model (Fotheringham *et al*. 2017). We

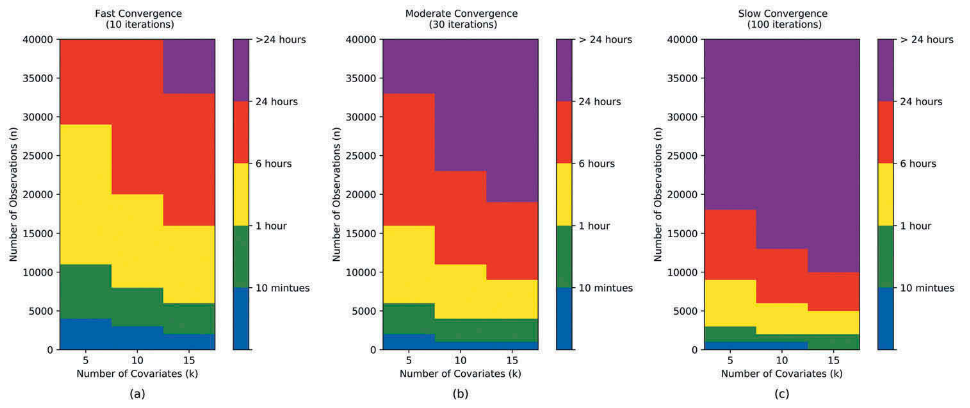

Figure 9. Approximate runtime needed to complete an MGWR calibration (with parameter estimation and inference) under three different convergence rates and for varying numbers of both observations and covariates.

extend the parallelization framework introduced in the FastGWR algorithm (Li *et al*. 2019) to speed-up each univariate GWR model within the MGWR back-fitting parameter estimation. Also, using the recently developed MGWR inference framework (Yu *et al*. 2019), we introduce a parallel chunk-wise partitioning method for computing MGWR inference to solve memory problems. We compare our improved calibration method with the original implementation used in Fotheringham *et al*. (2017) and find it is around 500 times faster. Also, we compare our new method with the *GWmodel* package in the R environment (Gollini 2015) and find it to be 30 to 140 times faster than *GWmodel* for sample sizes ranges from N = 1000 to N = 5000. For sample sizes larger than 5000, only our improved method is able to calibrate an MGWR model. We further demonstrate that our new algorithm scales well on multi-processor machines with a speed-up ratio of 2.4 on a 4-processor machine and 4.8 on an 8-processor machine. With the current improvement, an MGWR model with around 20,000 observations can be calibrated within a day on a normal desktop machine with 4 processors and 16 GB of memory. With a workstation with more processors and memory, it is expected that it is feasible to calibrate an MGWR model with up to 100,000 data points.

The methodology introduced in this paper has been integrated into the *mgwr* python package and the MGWR 2.0 GUI software, both of which are freely available to download. This not only enhances the accessibility of the MGWR model for new applications to explore multi-scale spatial heterogeneity but also brings the possibility of much larger scale local multi-scale analysis. In addition, the computational improvements described here can also benefit some recent spatial analysis models developed based on the MGWR framework such as the Parameter Specific Distance Metric Model (Lu *et al*. 2018) and the Multiscale Geographically and Temporally Weighted Regression model (Wu *et al*. 2019) both of which have severe computational restrictions.

Notes

1. The GWmodel R library is available at https://cran.r-project.org/web/packages/GWmodel.
2. The *mgwr* python library is hosted at https://github.com/pysal/mgwr.

3. The MGWR 2.0 software is freely available at https://sgsup.asu.edu/sparc/mgwr.
4. https://github.com/pythonprofilers/memory_profiler.
5. https://github.com/c040120/com_improv_mgwr_data_codes.
6. The computer used in Fotheringham *et al*. (2017) has dual 2.6 GHz 8-core CPUs and 64 GB of memory.
7. Results shown are based on testing machine described in Section 5.2; actual runtime may vary on different machines.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the National Science Foundation [1758786].

Notes on contributors

Ziqi Li is a Ph.D.Candidate in the School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ 85281. His research interests include GIScience and spatial statistics. He is currently working on the multi-scale geographically weighted regression mode l(MGWR), covering aspects from computation,inference to applications. E-mail: Li.Ziqi@asu.edu.

A. Stewart Fotheringham is a Regents' Professor of Computational Spatial Science in the School of Geographical Sciences and Urban Planning at Arizona State University, Tempe, AZ 85281. His research interests include spatial data analytics, spatial processes, and spatial interaction modeling. E-mail: stewart.fotheringham@asu.edu.

ORCID

Ziqi Li http://orcid.org/0000-0002-6345-4347

Data and codes availability statement

The data and codes that support the findings of this study are available in figshare.com with the identifier doi:10.6084/m9.figshare.10260881.v1.

References

Bitter, C., Mulligan, G.F., and Dall'erba, S., 2007. Incorporating spatial variation in housing attribute prices: a comparison of geographically weighted regression and the spatial expansion method. *Journal of Geographical Systems*, 9 (1), 7–27. doi:10.1007/s10109-006-0028-7.

Buja, A., Hastie, T., and Tibshirani, R., 1989. Linear smoothers and additive models. *The Annals of Statistics*, 17, 453–510. doi:10.1214/aos/1176347115

Comber, A.J., Brunsdon, C., and Radburn, R., 2011. A spatial analysis of variations in health access: linking geography, socio-economic status and access perceptions. *International Journal of Health Geographics*, 10 (1), 44. doi:10.1186/1476-072X-10-44.

Finley, A.O., 2011. Comparing spatially-varying coefficients models for analysis of ecological data with non-stationary and anisotropic residual dependence. *Methods in Ecology and Evolution*, 2 (2), 143–154. doi:10.1111/j.2041-210X.2010.00060.x.

Fotheringham, A.S., Brunsdon, C., and Charlton, M., 2002. *Geographically weighted regression: the analysis of spatially varying relationships*. Hoboken, NJ: John Wiley & Sons.

Fotheringham, A.S., Yang, W., and Kang, W., 2017. Multiscale Geographically Weighted Regression (MGWR). *Annals of the American Association of Geographers*, 107 (6), 1247–1265. doi:10.1080/24694452.2017.1352480.

Gelfand, A.E., *et al*., 2003. Spatial modeling with spatially varying coefficient processes. *Journal of the American Statistical Association*, 98 (462), 387–396. doi:10.1198/016214503000170.

Gollini, I., *et al*., 2015. GWmodel: an R package for exploring spatial heterogeneity using geographically weighted models. *Journal of Statistical Software*, 63, 17. doi:10.18637/jss.v063.i17

Harris, R., *et al*., 2010. Grid-enabling geographically weighted regression: a case study of participation in higher education in England. *Transactions in GIS*, 14 (1), 43–61. doi:10.1111/tgis.2010.14.issue-1.

Hastie, T.J. and Tibshirani, R.J., 1990. *Generalized additive models*. Vol. 43. CRC Press. Available from: https://www.crcpress.com/Generalized-Additive-Models/Hastie-Tibshirani/p/book/9780412343902.

Hu, X., *et al*., 2013. Estimating ground-level PM2. 5 concentrations in the southeastern US using geographically weighted regression. *Environmental Research*, 121, 1–10. doi:10.1016/j.envres.2012.11.003

Li, Z., *et al*., 2019. Fast Geographically Weighted Regression (FastGWR): a scalable algorithm to investigate spatial process heterogeneity in millions of observations. *International Journal of Geographical Information Science*, 33 (1), 155–175. doi:10.1080/13658816.2018.1521523.

Lu, B., *et al*., 2018. Improvements to the calibration of a geographically weighted regression with parameter-specific distance metrics and bandwidths. *Computers, Environment and Urban Systems*, 71, 41–57. doi:10.1016/j.compenvurbsys.2018.03.012

Murakami, D. and Griffith, D.A., 2015. Random effects specifications in eigenvector spatial filtering: a simulation study. *Journal of Geographical Systems*, 17 (4), 311–331. doi:10.1007/s10109-015-0213-7.

Oshan, T., *et al*., 2019a. mgwr: A Python implementation of multiscale geographically weighted regression for investigating process spatial heterogeneity and scale. *ISPRS International Journal of Geo-Information*, 8 (6), 269. doi:10.3390/ijgi8060269.

Oshan, T., *et al*., 2019b. A comment on geographically weighted regression with parameter-specific distance metrics. *International Journal of Geographical Information Science*. doi:10.1080/13658816.2019.1572895.

Oshan, T.M. and Fotheringham, A.S., 2018. A comparison of spatially varying regression coefficient estimates using geographically weighted and spatial-filter-based techniques. *Geographical Analysis*, 50 (1), 53–75. doi:10.1111/gean.2018.50.issue-1.

Tran, H.T., Nguyen, H.T., and Tran, V.T., 2016. Large-scale geographically weighted regression on spark. *In*: *Knowledge and Systems Engineering (KSE), 2016 eighth international conference on*, 127–132. October. IEEE. doi:10.1177/1753193416669263.

Troy, A., Grove, J.M., and O'Neil-Dunne, J., 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), 262–270. doi:10.1016/j.landurbplan.2012.03.010.

Wolf, L.J., Oshan, T.M., and Fotheringham, A.S., 2018. Single and multiscale models of process spatial heterogeneity. *Geographical Analysis*, 50 (3), 223–246. doi:10.1111/gean.v50.3.

Wu, C., *et al*., 2019. Multiscale geographically and temporally weighted regression: exploring the spatiotemporal determinants of housing prices. *International Journal of Geographical Information Science*, 33 (3), 489–511. doi:10.1080/13658816.2018.1545158.

Yu, H., *et al*., 2019. Inference in multiscale geographically weighted regression. *Geographical Analysis*. doi:10.1111/gean.12189.

Zhang, J., 2010. Towards personal high-performance geospatial computing (HPC-G): perspectives and a case study. *In*: *Proceedings of the ACM SIGSPATIAL international workshop on high performance and distributed geographic information systems*, 3–10. ACM. doi:10.1002/bit.22603.