Parallel Computing for Fast Spatiotemporal Weighted Regression

Xiang Que [a,b], Chao Ma[b,c*], Xiaogang Ma[b,*], Qiyu Chen[d]

[a] Computer and Information College, Fujian Agriculture and Forestry University, Fuzhou, Fujian, China

[b] Department of Computer Science, University of Idaho, 875 Perimeter Drive MS 1010, Moscow, ID 83844-1010, USA

[c] State Key Laboratory of Oil and Gas Reservoir Geology and Exploitation, Chengdu University of Technology, Chengdu 610059, China

[d] School of Computer Science, China University of Geosciences (Wuhan), 388 Lumo Road, Wuhan 430074, China

[*]Corresponding author. Email: machao@cdut.edu.cn (Chao Ma); max@uidaho.edu (Xiaogang Ma)

Abstract:

The Spatiotemporal Weighted Regression (STWR) model is an extension of the Geographically Weighted Regression (GWR) model for exploring the heterogeneity of spatiotemporal processes. A key feature of STWR is that it utilizes the data points observed at previous time stages to make better fit and prediction at the latest time stage. Because the temporal bandwidths and a few other parameters need to be optimized in STWR, the model calibration is computationally intensive. In particular, when the data amount is large, the calibration of STWR becomes heavily time-consuming. For example, with 10000 points in 10 time stages, it takes about 2307 seconds for a single-core PC to process the calibration of STWR. Both the distance and the weighted matrix in STWR are memory intensive,

which may easily cause memory insufficiency as data amount increases. To improve the efficiency of computing, we developed a parallel computing method for STWR by employing the Message Passing Interface (MPI). A cache in the MPI processing approach was proposed for the calibration routine. Also, a matrix splitting strategy was designed to address the problem of memory insufficiency. We named the overall design as Fast STWR (F-STWR). In the experiment, we tested F-STWR in a High-Performance Computing (HPC) environment with a total number of 204611 observations in 19 years. The results show that F-STWR can significantly improve STWR's capability of processing large-scale spatiotemporal data.

Keywords: Spatiotemporal Weighted Regression; Parallel computing; Geographically Weighted Regression; Spatial analysis; Spatiotemporal non-stationarity

1. Introduction

Spatiotemporal data are a topic of interest in many fields of research. The amount of spatiotemporal data has significantly accumulated in recent years due to improvements in data acquisition methods (Wikle, 2019). Many statistical models, such as the Bayesian hierarchical model (BHM Berliner,1996) and the spatiotemporal kriging (Pebesma et al., 2012, 2016), have been proposed to facilitate understanding how (and ultimately why) data vary in space and time. Nevertheless, the modeling of spatiotemporal processes still faces challenges caused by the spatiotemporal non-stationarity (i.e., relationships between variables change when the locations change and time variation).

The Geographically and Temporally Weighted Regression (GTWR) (Huang et al., 2010), an extended model of the Geographically Weighted Regression (GWR) (Brunsdon et al. 1996;

Fotheringham et al. 2003), is developed to analyze the spatiotemporal non-stationarity. Although it addressed the issues to some extent (Wu et al., 2013; Wrenn and Sam 2014), calculating distance in three dimensions (two-dimension in space and one dimension in time) in GTWR still can be improved. Location and time are usually measured at different scales, and their units and impacts on regression points are fundamentally different. Although GTWR uses the adjustment factor τ to synthesize the distance between time and space, a sole measure integrating the spatial and temporal distances is not enough (Fotheringham et al., 2015). Moreover, the performances of GTWR in fitting and prediction are sometimes even inferior to the traditional GWR models (Que et al., 2020).

A new GTWR (Fotheringham et al., 2015) model was proposed to use a set of time-isolated spatial bandwidths to capture the local effects from observation points on regression points in space and time. But the calibration process is cumbersome and cannot simultaneously optimize the bandwidths of time and space. Furthermore, both GTWR models (Huang et al., 2010; Fotheringham et al., 2015) regard the time interval as the temporal distance, resulting in that all observations from different locations recorded at the same previous time having an equal temporal weight on a regression point. However, the magnitudes of value variation during the time interval in different observations have different influences on the regression point (Que et al., 2020). The more significant the value changes during the time interval, the higher its impact is on the regression point. The variation rate of attribute values (non-stationarity in time) also has heterogeneity in space. Using the time intervals as the temporal distances for calculating temporal weights cannot fully capture the local temporal effects from observations to the regression point. Based on this idea, the Spatiotemporal Weighted Regression (STWR) model (Que et al., 2020) was developed to capture better the combined effects of a non-stationary spatiotemporal process from observed data.

67 STWR treats the time distance as the rate of value variation through a time interval rather than the
68 time interval itself, which is more suitable to measure the degree of temporal impact from each
69 observation point to a regression point. Besides, it utilizes a weighted average form to calculate the
70 spatiotemporal kernel rather than the multiplication form in GTWR (Huang et al., 2010; Fotheringham
71 et al., 2015), thus, avoiding potential underestimation of combined spatiotemporal effects (Que et al.,
72 2020). Compared with GWR and GTWR, these new features in STWR significantly improve model
73 fitting and prediction capabilities for the latest time stage (Que et al., 2020). However, because the
74 temporal bandwidths and several other parameters need to be optimized, the calibration of STWR is
75 more complicated and time-consuming than the original GWR model. It requires intensive computation
76 and memory usage. It is a challenging issue and may seriously limit the application of STWR in large-
77 scale spatiotemporal data processing. Inspired by FastGWR (Li et al., 2019), a parallel implementation
78 of GWR, we developed a parallel computing version of STWR, named Fast STWR (F-STWR), to scale
79 up the capacity of STWR.

80 In the remainder of this paper, Section 2 will overview STWR and its differences with GTWR.
81 Section 3 will introduce the parallel calibration approach developed for STWR. Section 4 will present
82 the experimental results. Section 5 will compare the characteristics of F-STWR and FastGWR. Finally,
83 Section 6 will summarize the highlights of F-STWR.

84 **2. The calibration and optimization procedure of STWR**

85 **2.1 Model formulation of STWR and its differences with GTWR**

86 Derived from the basic GWR framework (Brunsdon et al. 1996; Fotheringham et al. 2003), the STWR
87 formula is shown in Equation 1.

88 $$y_i^t = \beta_0^t(u_i, v_i) + \sum_k \beta_k^t(u_i, v_i) x_{ik}^t + \varepsilon_i^t \quad (1)$$

In this equation, y_i^t denotes the i^{th} dependent variable at the regression point (u_i, v_i) at time t. x_{ik}^t denotes the k^{th} independent variable. ε_i^t is the independent random error term with distribution $N(0, \sigma^2)$ (STWR assumes that all random error terms at different time stages meet the same independent and identical distribution). $\beta_0^t(u_i, v_i)$ and $\beta_k^t(u_i, v_i)$ denote the constant item and coefficient at location (u_i, v_i), respectively.

Equation 2 shows the estimator for the coefficient.

$$\hat{\beta}^t(u_i, v_i) = [(X_{s_{\Delta t}}^T W_{\Delta t}(u_i, v_i) X_{s_{\Delta t}})^{-1} X_{s_{\Delta t}} W_{\Delta t}(u_i, v_i)] y_{s_{\Delta t}} \quad (2)$$

In this equation, $X_{s_{\Delta t}}$ is a matrix of all observed independent variables within the time interval Δt. $y_{s_{\Delta t}}$ is the corresponding vector of dependent variables. $X_{s_{\Delta t}}^T$ is the transpose of $X_{s_{\Delta t}}$. $W_{\Delta t}(u_i, v_i)$ denotes the spatiotemporal weight matrix of observations from different locations and time stages within Δt, which will be used for the regression point $i(u_i, v_i)$ at t.

In $W_{\Delta t}(u_i, v_i)$, an element w_{ijST}^t is used to reflect the spatiotemporal impact from other observations. Both STWR and GTWR model have their own spatiotemporal kernel functions to calculate the weight value w_{ijST}^t, but their designs of the spatiotemporal kernel are different. The spatiotemporal kernel of GTWR is a multiplication form of the spatial kernel and the temporal kernel, as shown in Equation 3 (Fotheringham et al., 2015).

$$w_{ijST}^t = k_s(d_{sij}, b_S) \times k_T(d_{tij}, b_T) \quad (3)$$

In this equation, the weight w_{ijST}^t denotes the impact from observed data point j to the regression point i at time stage t. k_s and k_T are spatial kernel and temporal kernel, respectively. d_{sij} and d_{tij} denote the Euclidean distance and time distance from an observed data point j to the regression point i, respectively. b_S, and b_T are the spatial bandwidth and temporal bandwidth, respectively.

In comparison, STWR applies a new weighted average form of spatiotemporal kernel for calculating the w_{ijST}^t, as shown in Equation 4 (Que et al., 2020).

$$w_{ijST}^t = (1-\alpha) k_s(d_{sij}, b_{ST}) + \alpha k_T(d_{tij}, b_T), 0 \le \alpha \le 1 \quad (4)$$

In this equation, b_{ST} denotes the spatial bandwidth at t. Outputs of both spatial and temporal kernel functions range from 0 to 1. α is an adjustable parameter for balancing (enlarging or reducing) the effects from time and space.

To obtain the combined weight value w_{ijST}^{t}, the calculation of the time distance (d_{tij}) is also critical. In GTWR, the d_{tij} is the time interval (difference) between two observed time stages. However, the time distance (d_{tij}) in STWR is not the time interval or time difference, but the rate of value variation between an observed point and a regression point through a time interval. The temporal kernel k_T is defined in Equations 5 (Que et al., 2020).

$$w_{ij\Delta t}^{t} = \begin{cases} \left[\frac{2}{1+exp(-d_{ij\Delta t}^{t})} - 1\right], & if\ 0 < \Delta t < b_T \\ 0, & otherwise \end{cases} \quad (5)$$

$$\text{where, } d_{ij\Delta t}^{t} = \begin{cases} \frac{\left|\left(y_{i(t)} - y_{j(t-q)}\right)/y_{j(t-q)}\right|}{\Delta t/b_T}, & if\ 0 < \Delta t < b_T \\ 0, & otherwise \end{cases} \quad (6)$$

In Equation 6, $y_{i(t)} - y_{j(t-q)}$ is the value variation from regression point i at t to point j at $t-q$, which denotes the value change during the time interval Δt. Δt is the time difference between t and $t-q$. When the time interval Δt is out of the range $(0, b_T)$, the weight is set to zero.

Like most distance-decay weighting strategy, it is usually assumed that the spatial bandwidth becomes narrower with the time distance increases. The spatial bandwidth b_{ST} in STWR is assumed to be linearly changing along with time from the current time stage to the previous stages, as shown in Equation 7 (Que et al., 2020).

$$b_{ST} = b_{St} - \tan\theta * \Delta t,\ -\frac{\pi}{2} < \theta < \frac{\pi}{2} \quad (7)$$

In this equation, $\tan\theta$ denotes the slope, and b_{St} denotes the initial bandwidth at the latest time stage t. The other spatial bandwidth b_{ST} in the past time stage is derived from b_{St}. Combining Equations 4 and 7 with the bi-square or Gaussian spatial kernel function (Fotheringham et al., 2002), the

spatiotemporal kernel in STWR (Equation 4) can be further derived. As shown below, Equations 8 and 9 are based on the bi-square and Gaussian kernel, respectively.

$$w_{ijST}^{t}=\begin{cases}\left[(1-\alpha)*\left[1-\left(\frac{d_{sij}}{b_{St}-\tan\theta*\Delta t}\right)^{2}\right]^{2}+\alpha*\left(2/\left(1+\exp\left(-\frac{\left|(y_{i(t)}-y_{j(t-q)})/y_{j(t-q)}\right|}{\Delta t/b_{T}}\right)\right)-1\right)\right], \\ \quad if\ \Delta t<b_{T},\ \ and\ \ d_{sij}<(b_{St}-\tan\theta*\Delta t) \\ 0,\ \ otherwise\end{cases} \tag{8}$$

$$w_{ijST}^{t}=\begin{cases}\left[(1-\alpha)*\exp\left[-\frac{1}{2}\left(\frac{d_{sij}}{b_{St}-\tan\theta*\Delta t}\right)^{2}\right]+\alpha*\left(2/\left(1+\exp\left(-\frac{\left|(y_{i(t)}-y_{j(t-q)})/y_{j(t-q)}\right|}{\Delta t/b_{T}}\right)\right)-1\right)\right], \\ \quad if\ \Delta t<b_{T},\ \ and\ \ d_{sij}<(b_{St}-\tan\theta*\Delta t) \\ 0,\ \ otherwise\end{cases} \tag{9}$$

In GWR, only the single spatial bandwidth b_{ST} needs to be optimized. In comparison (as shown in Equations 8 and 9), STWR needs to get the optimized initial spatial bandwidth b_{St}, temporal bandwidth b_T, and the optimized parameters α and θ. Thus, besides the increased volume of data, STWR also needs to optimize more parameters in its calibration procedure.

2.2 Procedure for model calibration and parameter optimization in STWR

Similar to the general GWR, some popular goodness-of-fit diagnostics such as the Cross-Validation (CV) (Cleveland, 1979), the Akaike Information Criterion (AIC) (Akaike 1973; Hurvich et al., 1998), and the corrected AIC (AICc, Cavanaugh, 1997) (AICc performs better than AIC with small sample sizes) are also appropriate for STWR. Once we get the weighted matrix $W_{\Delta t}$, the row r of that matrix can be calculated using Equation 10.

$$r_{it}=X_{it}(X_{\Delta t}^{T}W_{i\Delta t}X_{\Delta t})^{-1}X_{\Delta t}W_{i\Delta t} \tag{10}$$

In this equation, X_{it} is the i^{th} row of the independent variable matrix at t. $X_{\Delta t}$ is a matrix of all the observed independent variables during a certain time interval Δt, and $W_{i\Delta t}$ is the i^{th} row of the

weighted matrix $W_{\Delta t}$.

The optimization is a procedure of searching through multiple trials by utilizing these diagnostics scores. Specifically, for each regression point $p_i^{(t)}$, STWR needs to traverse two loops to obtain the optimized values for α, θ, the number of time stages used by STWR, and the initial spatial bandwidth b_{St} at time stage t. The searching range of the temporal bandwidth b_T is limited to a discrete set $BT_\lambda = \{\Delta t_1, \Delta t_2, \ldots \Delta t_\lambda\}$, in which the element Δt_λ is the time interval from t to $t - \lambda$. The initial spatial bandwidth b_{St} is also limited to a discrete set $BS_{Nt} = \{D_{k+1}, D_{k+2}, \ldots D_{N_t}\}$, in which the element $D_U, U \in \{k+1, k+2, \ldots, N_t\}$ denotes the distance from $p_i^{(t)}$ to the U^{th} nearest neighbor, and k equals to the number of independent variables. BT_λ and BS_{Nt} are determined by the number of total observation time stages and the number of observation points at time stage t, respectively. In the two loops for STWR parameter optimization, the outer loop is to traverse the set of BT_λ, and the inner loop is to traverse the set of BS_{Nt}. If the current traversed time interval element is Δt_q (q = 1, 2, …, λ) in BT_λ and the initial spatial bandwidth is D_{k+p} (p = 1, 2, …, N_t), then the range of θ can be calculated for each past time stage $t - s$ (s = 1, 2, …, q) in Δt_q, because the spatial bandwidth b_{ST} at $t - s$ is not farther than the distance D_{k+p} (i.e., spatial bandwidth value decreases over the time in the past). With user-specified maximum number of iterations and searching methods (e.g., Golden-section or equal interval searching), the weighted matrix $W_{\Delta t}(u_i, v_i)$ can be calculated for each iteration, and its corresponding $\hat{\beta}^t(u_i, v_i)$ (Equation 2) can be solved by employing the Iteratively Reweighted Least Squares (IRLS) method. Finally, the CV or AICc scores can be obtained at each iteration during the traverse.

To improve the applicability of STWR, we designed a matrix splitting calculation method and a parallel search method to save memory usage and speed up the procedure of parameter optimization in

STWR. The development and implementation of those methods were based on the mpi4py package in Python. More technical details are presented in the next section.

3. The method of parallel calibration and its implementation in F-STWR

3.1 Design of the parallel algorithm

Message Passing Interface (MPI) is a common protocol for parallel computing, widely adopted in many fields (Schmidt et al., 2002; Gabriel et al., 2004; Neese 2012; Wu et al., 2013). For STWR, the most time-consuming calculation lies in the procedure of parameter optimization, which is conducted by searching through multiple trials. Intuitively, we can consider scattering those trials to different processors evenly (i.e., a parallel form) to improve the efficiency. To put that idea into practice, we need to look into the detailed procedure in STWR. In the basic GWR model, the diagnostics score only depends on all regression points' local standard errors, whose parallel calculation form refers to the method described by Li et al. (2019). In STWR, however, we should also code the traversal procedures of all the time stages into the parallel form. In a multi-processor system, MPI identifies processes according to their relative ranks in a group (Snir et al., 1998). A unique integer identifier (i.e., within the range from 0 to group size - 1) is assigned to each processor(rank). The group size is the number of tasks in a multi-processor system. A rank is also called a “task ID” in some studies (Barney, 2012). For F-STWR, we use rank 0 as the rank root in MPI, responsible for assigning tasks and collecting results. To implement MPI in STWR, we should make sure that each rank's traversing progress keeps pace with the MPI rank root consistently. Moreover, during the optimization procedure, we should combine some results for further calculation, such as gathering all the points’ impacts from different observed time stages to the rank root to calculate each trial's scores. The details of the algorithm to compute the diagnostics scores are summarized in Algorithm 1.

195

Algorithm 1: Parallel implementation of F-STWR for model fitting and parameter optimization

1: Root (rank 0) reads the data, the searching method, and diagnostics type, then calculates the initial spatial bandwidth according to the number of the latest observation points

2: Broadcast data to other processes with ranks not equal to 0. Each process gets its own chunk of index numbers of the latest observation points. Divide the total number of latest observations by the size of MPI processes and then get the ceiling value m_ck, and each process keeps a chunk of index numbers ranging from rank * m_ck to rank * (m_ck+1)

3: For each element in set BT_{λ}:

4: The root changes current Cache_Type to 0 and broadcasts the Cache_Type

5: For each element in BS_{Nt}:

6: Calculate the range of θ

7: For iteration element in θ:

8: Execute searching function (α):

9: (1) Root executes function F1

10: (2) Each rank executes searching function mpi_CalCriterion

11: (3) Root executes function F2

12: Root collects all diagnostics scores and then outputs the best score and its corresponding parameters

196

197 The parameter Cache_Type and its two control functions (F1 and F2) are used here to keep the pace
198 consistent with the rank root and avoid unnecessary iterative calculations. A different cache type stands

for a different computing status. The mpi_CalCriterion function will retrieve matrixes from different memory (if cached) or make different matrixes cached (if not cached). Table 1 outlines the status of different cache types and their corresponding descriptions.

Table 1. Status and description of each cache type

Cache_Type	Status and description
0	No content cached and unstable. Automatically change to M in the next step. If the previous status is 4, M will be 2, otherwise M is its default value 1
1	Both the spatial distance (SD) and temporal distance (TD) matrixes are cached and stable but should be truncated before recalculating the spatiotemporal weight matrix W
2	Both SD and TD are cached and unstable, but they should be further truncated, masked and cached. Automatically change to 3 after the masked SD and TD are cached.
3	Both masked SD and TD are cached and stable. Only need to adjust the factor α to enlarge/reduce the spatiotemporal effect (W)
4	Auxiliary status to control the M value

The rank root mainly controls the status of Cache_Type except for some unstable status (0 or 2). There are three different scenarios for the mpi_CalCriterion function in Algorithm 1: (1) Traverse to a new element in set BT_{λ}. The total time interval that the model used will be changed, so all the matrixes (if cached) will be invalid. So, the Cache_Type will be set to 0. (2) Two different branches before we

execute the mpi_CalCriterion function (in F1): One is that our program's Cache_Type is 0, and we should change its status to 4. This is because in our searching function (Line 8 in Algorithm 1), we only need to adjust α, and there is no need to recalculate all the distance matrixes. The other is that our program's Cache_Type is 1, which means that we have only cached the distance matrixes without masking. The matrix should be masked, so the Cache_Type is set to be 2. (3) After the mpi_CalCriterion function (in F2): If our program's Cache_Type is 3, we may execute another iteration, and the masked matrixes will be invalid. So, the Cache_Type will be reset to 1. After executing the F1 and F2 functions, the rank root should broadcast the current Cache_Type to all other ranks.

Next, the mpi_CalCriterion function is inspected, which will obtain a diagnostic score through the rank root. In the implementation, the rank root gathers results from all other ranks, which calculate with their own chunk of data. The detail is summarized in Algorithm 2.

Algorithm 2: mpi_CalCriterion function for calculating diagnostics scores of different cache types

```
1: Given a chunk of index numbers CK
2: Check cached matrixes and set M value (described in Table 1)
3: switch Cache_Type:
4:         case 0:
5:                 Clear cached matrixes
6:                 For index in CK:
7:                         Execute function local_CalSearch to get component value
8:                 Root gathers all the component values
9:                 Root calculates diagnostics score (Li et al., 2019)
```

10: Submit distance matrixes (corresponding to chunk) into cached matrixes

11: Set current Cache_Type to M

12: case 1:

13 Copy and sort current cached distance matrixes

14: Truncate matrixes according to current bandwidth (element in BS_{Nt})

15: Calculate the weighted matrix W according to the truncated matrixes

16: For index in CK:

17: Calculate component value according to corresponding index in W

18: Root gathers all the component values

19: Root calculates diagnostics score (see Line 9)

20: case 2:

21: Execute same as Line 13 to 19

22: Cache current masked distance matrixes (truncated)

23: Set current Cache_Type to 3

24: case 3:

25: Calculate the weighted matrix W according to the cached masked matrixes

26: Execute same as Line 16 to 19

The function local_CalSearch (Line 7 in Algorithm 2) calculates the component (in a pointwise way), which is gathered by the rank root for calculating the diagnostics scores. The details of diagnostics score calculation (such as CV and AICc) were adopted from Li et al. (2019). Here we describe the local_CalSearch in Algorithm 3.

225

Algorithm 3 local_CalSearch function for calculating component value of each regression point

1：Given current regression point $p_i^{(t)}$ corresponding to the index number in chunk

2：For past time stages $t-s$ in Δt_q (element in BT_λ):

3: Calculate spatial distance and temporal distance (Equation 6) matrixes to $p_i^{(t)}$

4: Calculate weight matrix from Equation 8 or 9

5: Calculate component value according to the diagnostics type:

(a) CV score: calculate the coefficient matrix (Equation 2) and get the residuals

(b) AICc score: calculate residual squared εi^2 and the diagonal element of the hat matrix (Hoaglin and Welsch, 1978) from Equation 10

6: Return component value and spatial and temporal distance matrixes

226 **3.2 Strategy of splitting matrix for calculation**

227 The implementation of a parallel algorithm reduces memory usage (Li et al., 2019). However, in
228 FastGWR studies, the memory reduction only happens during the procedure of parameter optimization.
229 For F-STWR, a new design is needed because when the optimized parameters are used for predicting
230 unobserved points, STWR needs to calculate or build up new big matrixes. At that stage, if the total
231 number of points is too large, the STWR program may break down. We can alleviate this problem by
232 adopting the strategy of matrix splitting for calculation. The idea of avoiding big matrixes in STWR is
233 to split them into small ones. A strategy was proposed and applied in the F-STWR program.

234 (1) Before splitting the matrix, we should first know how many data points (or memory) are used
235 for calculation. Users specify the maximum number of points (max_tol). If the available memory reduces,
236 our model will need more iterations, increasing the total time for calculation. So, it is suggested to specify

a proper value for splitting according to the user's system. If the max_tol is set, the minimum memory size must be greater than Mem_min, given in Equation 11.

$$Mem\_min = rank\_num \times \max\ \_tol^2 \times Mem\_dtype \quad (11)$$

In this equation, $rank\_num$ is the number of ranks we plan to use, Mem_dtype is the memory size of the data type of matrix, e.g., float32 in python is 4 bytes.

(2) The next step is to determine how many parts we should have. This issue is solved by obtaining the ceiling value of the result in a division calculation (i.e., the number of observations divided by max_tol). We should also calculate each part's row number, which is determined by the number of points at the latest time stage (suppose the number is N_latest). In the calibration of STWR, the distance matrix (or weighted matrix) is from the past observation points to each regression point at the latest time stage. If the total number of past points is N_past, then the dimension of this matrix is N_latest by N_past, which is larger than the limitation of max_tol by max_tol. In this case, the N_latest is split into several parts to ensure that every part's size is smaller than the limitation. A note here is that the matrix should not be split by N_past, which may cause missing data when calculating weights from observations to each regression point.

(3) Instead of directly calculating the spatial and temporal distance matrixes for all the regression points, each part's matrixes are calculated. Then the weights of regression points in each part are calculated. All the rows in each part are assigned a number to keep a consistent order with the points observed at the latest time stage.

(4) Last, a list of weight values is built to replace the original big matrix. To this end, all the data are gathered for calculating summary information such as AICc, R-squared, Effective number of parameters (trace(S)), etc., by tracking the assigned number.

4. Experiments and comparison of results

We use the mpi4py package (Dalcin et al., 2008) in Python to call the OpenMPI program (which is an open-source implementation of MPI that can be executed on a single CPU computer and High-Performance Computing clusters) to execute STWR. Our parallel algorithm (F-STWR) was deployed on the University of Idaho's IBEST Computational Resources Core, which has a total of 1500 processor cores. We ran our experiments on the SuperMocro rack server (named Watson & Crick) with 48 logical cores and 256 GB of system memory. Crick features 4 Intel Xeon Phi 5110p Co-processor cards. Watson has 2 NVIDIA P100 cards.

4.1 Verification of the calibration results of the parallel algorithm

To verify the consistency of the fitting results of the parallel algorithm F-STWR, we used the same real-world data in Que et al. (2020) for comparison. The data are the daily mean precipitation hydrogen isotopes (δ^2H) of three consecutive days in Northeastern United States. To be consistent with the previous setting in the STWR experiment (Que et al., 2020), we set θ in Equation 7 to zero for the F-STWR algorithm. A comparison of the results in F-STWR and STWR is shown in Table 2. All values are the same except for some slight differences in the sum of squared errors (SSE) of the second (D2) and third (D3) days. The differences are most likely caused by rounding the decimal point in assigning parallel tasks and collecting results from the parallel cores.

Table 2. Comparing model calibration results of F-STWR and STWR.

Model	Model Parameters & Diagnostic information	STWR	F-STWR
	SSE	**24022.226**	**24022.195**
	R2	0.834	0.834

D2	AICc	977.181	977.181
	Init Bandwidth	16	16
	alpha	0	0
	Time stages model used	2	2
	SSE	**25118.096**	**25118.082**
	R2	0.763	0.763
D3	AICc	669.648	669.648
	Init Bandwidth	16	16
	alpha	0	0
	Time stages model used	2	2

277 **4.2 Efficiency test of F-STWR**

278 The parallel algorithm and matrix splitting approach described earlier were employed to calibrate the
279 STWR model using house price data in London obtained through the Nationwide Building Society
280 (Fotheringham et al., 2015). Interested readers can contact the authors of Fotheringham et al. (2015) for
281 the data. The data consists of a set of annual house prices from 1980 to 1998 with 204,611 observation
282 points, and each data point contains 21 independent variables. Here, the in-depth analysis of the data is
283 omitted. Instead, we present the experimental results demonstrating parallel computing ability in F-
284 STWR for tackling large-scale spatiotemporal data.

285 In the first experiment, we compared our parallel algorithm F-STWR with STWR for different numbers
286 of samples (1K to 10K points, K denotes 1000). The results show that F-STWR is much less time-
287 consuming than STWR. As Fig. 1 shows, the average runtime (in seconds) of STWR increases from
288 12.68s of 1K points to 258.85s of 5K points, and then up to 2307.68s of 10K points. In contrast, F-

STWR_15, F-STWR_30, and F-STWR_45 (F-STWR_15, F-STWR_30, and F-STWR_45 denote running F-STWR with 15, 30, and 45 MPI processors, respectively) are much faster than the original STWR, especially when the number of observations increases. For 10K points, the average runtime of F-STWR_15, F-STWR_30, and F-STWR_45 are 136.75s, 108.35s, and 112.29s, respectively. Those results are almost just one-twentieth of the runtime in STWR. The F-STWR_45 is slightly slower than F-STWR_30. This may because there are more internal communications within F-STWR_45 (45 processors) than F-STWR_30 (30 processors) during the parallel processing.

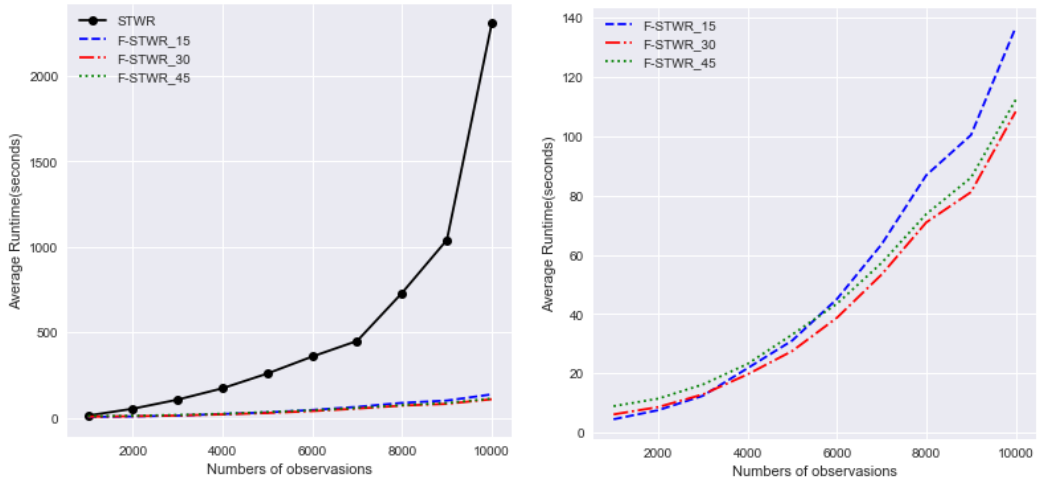

Fig. 1. Comparing average runtime of scalability as data size grows. (a) Comparison results of STWR and F-STWR with multiple processors (b) Comparison of F-STWR with different numbers of employed MPI processors. F-STWR_15, F-STWR_30, and F-STWR_45 denote F-STWR model run with 15, 30, and 45 processors, respectively.

The second experiment was with a fixed number (10K) of observation points and an increasing number of MPI processors. As show in Fig. 2, the runtime decreases significantly as the number of MPI processors increases, especially in the first 20 processors. The runtime (in seconds) drops from 307.26s of 5 processors to 103.32s of 35 processors. When the number of processors is above 35 the runtime does not further decrease, and even slightly increases. As mentioned in the last paragraph, this may be caused

by the increased internal communications.

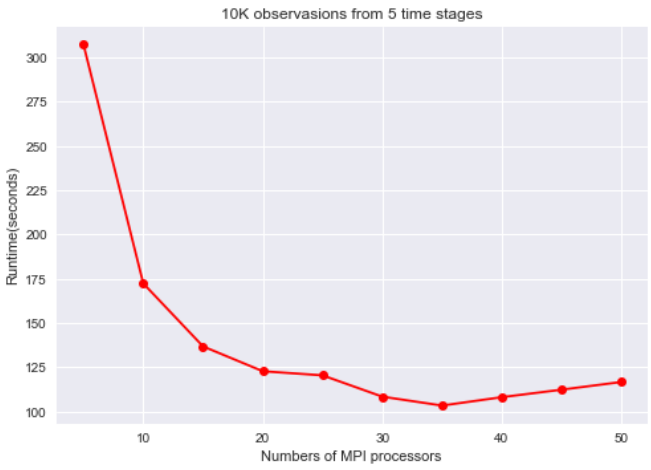

Fig. 2. Runtime of F-STWR for utilizing an increasing number of MPI processors with a fixed number of 10K data points.

In the third experiment, all the 204,611 observation points were used in a step-wise process. As shown in Fig. 3, when the total number of accumulated data points increases along the years, the running time of F-SWTR, however, shows an interesting pattern of ups and downs. Overall, the running time of F-STWR at a time stage (shown in blue square spots) is consistent with the number of observation points at that time stage (shown in red round spots). Our thoughts for the patterns in Fig.3 are that the running time is affected primarily by the number of points at the latest time stage and secondly by the cumulative number of data points. In this step-wise and accumulative process, each year becomes the latest time stage in a sequence. On one hand, an intuitive understanding is that the increasing number of time stages and accumulated data points will put more workload in each parallel core, which will increase the running time. On the other hand, only the data points at the latest time stage were used as regression points, which determines the size of the workload in each parallel core. A smaller number of regression points means lighter workload and thus less running time (e.g., at time stages 1992, 1993, 1995 and 1998). We may raise a hypothesis that the number of observation points at the latest time stage is the key controlling

factor for the running time of F-STWR. Nevertheless, here the experiment was done with only one dataset. To verify that hypothesis, we plan to test F-STWR with more datasets in the future work.

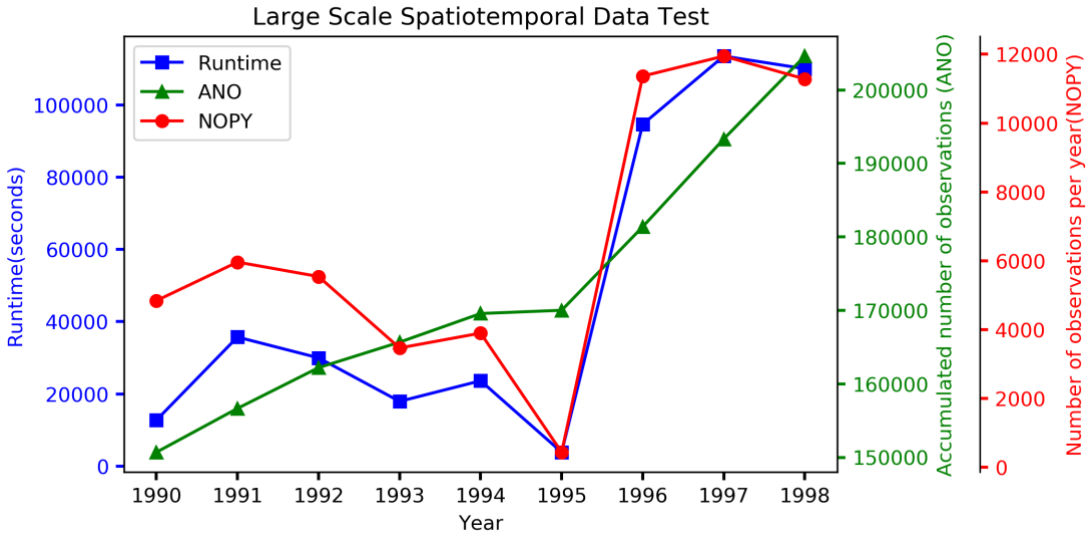

Fig. 3. Runtime test of F-STWR on each time stage of large-scale data points (50 MPI processors employed).

When the whole dataset (204,611 points) was tested with 50 MPI processors (time stage 1998 in Fig. 3), it took 110001.49 seconds (30.5 hours) to run F-STWR. As shown in Table 3, the results are compared with the ordinary least squares (OLS) method and GWR applied to the latest time stage (11,282 points at 1998). The temporal bandwidth (optimized time stages) in F-STWR is 5, which means that a total of 38,936 observation points in the recent 5 time stages were employed to fit the F-STWR. Different metrics in Table 3 show that F-STWR is better than the general GWR and OLS in this experiment. The Root Sum Square (RSS) of F-STWR is smaller than OLS and GWR. Also, both the AICc and R-squared (R2) of F-STWR are better. The estimated standard errors (Sigma) of F-STWR are less than two-thirds of GWR.

Table 3. Comparison of model performance.

	RSS	AICc	R2	Sigma	Bandwidth

OLS	16852704643550.800	270390.408	0.701		
GWR	11383507478243.400	266278.067	0.798	32015.787	3604.000
F-STWR	8475299504397.160	263500.640	0.850	19287.224	649

338

339 **4.3 Memory usage test on F-STWR and STWR**

340 In most desktop computers, the original STWR is not able to be calibrated for large datasets (such as
341 the one used in section 4.2) due to the limitation of memory size. F-STWR can further employ the
342 strategy of matrix splitting to address the issue. In our work we utilized the function “tracemalloc” in
343 python 3.6.8 to test the memory usage of F-STWR and STWR. For easier comparison, we prepared a
344 dataset of 1000 points and 5 time stages, and we recorded the peak memory usage during the model
345 calibration of both F-STWR and STWR. The comparison result is listed in Table 4. The column F-
346 STWR shows the result of F-STWR without matrix splitting. The F-STWR_SP1, F-STWR_SP2, F-
347 STWR_SP3 are three cases in which we used matrix splitting, and their parameter “max_tol” were set
348 to 90000(300*300),160000(400*400) and 250000 (500*500), respectively. The peak memory usage of
349 STWR is the largest (23.182 MB), which is about 162% much more than F-STWR (14.329 MB), and
350 about 326% much more than F-STWR_SP1(7.109 MB).

351 In general, this result verifies the effectiveness of the matrix splitting strategy. Nevertheless, not all
352 the settings of matrix splitting can facilitate memory saving. For instance, the peak memory usage of F-
353 STWR_SP3 (14.332 MB) is not less than F-STWR (14.329 MB). Moreover, the results show that both
354 the peak memory usage of F-STWR (14.329 MB) and F-STWR_SP3 (14.332 MB) are close to the current
355 memory usage of STWR (14.206 MB). It seems that the maximum amount of memory usage in our
356 parallel algorithm is a little larger than the STWR model after calibration. This can be a topic for more

exploration in our future work.

Table 4. Comparison of memory usage and runtime of F-STWR and STWR

	STWR	F-STWR	F-STWR_SP1	F-STWR_SP2	F-STWR_SP3
Current (MB)	14.206	5.034	3.512	3.445	5.041
Peak (MB)	23.182	14.329	7.109	8.891	14.332
Runtime(seconds)	705.180	106.470	95.831	98.254	101.090

F-STWR, F-STWR_SP1, F-STWR_SP2, F-STWR_SP3 run on 12 processors.

5. Discussion

5.1 A reflection on the applicability of F-STWR and STWR

The current F-STWR only considers how to parallelize STWR and solve the issues of computational efficiency and memory usage. We have not considered the influence of different types of data on the algorithm. For example, to employ the current F-STWR algorithm to analyze large-scale grid data, it still needs to calculate in a pointwise procedure (see above-mentioned Algorithms 2 and 3), although points can be assigned to different nodes. We can not simply split the grid data subjectively to achieve parallelism because it might destroy the spatial continuity of the study area, causing issues such as the modifiable areal unit problem (Wong 2004; Dark et al., 2007). One possible solution is to use clustering or other methods to find the best solution for dividing grid data before assigning them to different parallel cores.

In the Introduction section we have briefly compared STWR, GWR, and GTWR. Based on the presented examples of F-STWR, we can further extend the comparison with a few

more algorithms in spatial and spatiotemporal data analysis, especially with the kriging-based models. The kriging-based approach and GWR-based approach each has its own focuses and characteristics. First, the kriging-based approaches mainly aim at interpolation, and they do not require the input data to be multivariate in the spatial (or spatiotemporal) coordinates. In contrast, the GWR-based approaches require their input data having explanatory (independent) variables and response (dependent) variables. Second, the kriging-based approaches assume that a study area has various degrees of stationarity. Take the simple kriging as an example, it assumes stationarity of the first moment over the entire domain with a known mean. However, GWR-based approaches mainly target at exploring the non-stationarity relationships between the explanatory and response variables. Third, the kriging-based approaches can obtain the Best Linear Unbiased Prediction (BLUP), while in GWR-based approaches, Fotheringham et al., (2003) frame the choice of bandwidth as a trade-off between bias and variance. Thus, in our understanding, GWR-based approaches are perhaps more like exploratory methods rather than methods used to draw inference.

With the above-mentioned main differences, the running time of GWR, in fact, is not comparable to the general kriging model. Although the basic GWR is more time consuming than standard kriging, GWR-based approaches can handle some problems that the current kriging method cannot, such as constructing the spatially varying coefficients (SVC) between the explanatory variables and the response variables. Some studies have tried to combine GWR with kriging and apply it to specific topics, such as mapping soil organic carbon stock (Kumar et al., 2012) and topsoil electrical conductivity (Yang et al, 2019). For our future work, we have the interest to consider how to combine the current STWR model with the

kriging (or spatiotemporal kriging) approach. But in this paper, we mainly focus on the parallel calibration of STWR model for exploring the non-stationarity relationships of large-scale data points between variables in space and time.

5.2 Comparison between F-STWR and FastGWR

Without parallel processing, the current open-source GWR software can handle up to approximately 15,000 observations, which is a severe limitation (Li et al., 2019). A parallel implementation of GWR, called FastGWR (Li et al., 2019), was proposed to address the calibration problem of large-scale data in GWR. FastGWR reshaped the optimization algorithm of GWR and employed the MPI method for parallel computing. However, FastGWR is only able to handle data without a time dimension. Its algorithm design is for GWR, which cannot be used directly to the calibration procedure of STWR. Moreover, the spatiotemporal kernel of STWR is much more complicated than the spatial kernel in the original GWR, which increases the difficulty of parallel implementation of STWR.

Although our idea of applying parallel implementation to STWR was inspired by FastGWR, F-SWTR was very different from FastGWR, mostly due to the difference between STWR and GWR. On one hand, the weighting method of the STWR model is different from GWR, and its kernel function and calibration procedures are more challenging, because STWR needs to weigh the combined effect of time and space and to optimize the spatial bandwidths at different time stages, as well as the temporal bandwidths. On the other hand, the current FastGWR does not consider the combination matrix of weighted calculated by each parallel core, which is needed for prediction and may further cause insufficiency memory usage.

To address these limitations, we designed a parallel implementation algorithm for STWR together with a strategy of matrix splitting. We named the overall design as Fast STWR (F-STWR). There are

five aspects to show that F-STWR is different from FastGWR. (1) F-STWR provides spatiotemporal kernel functions for calculating the combined weights from time and space, while FastGWR only has the spatial kernel for calculating weights from space. (2) F-STWR can optimize the best number of time stages for the model to use, which is not considered in FastGWR. (3) The weight matrixes of each time stage in F-STWR are not necessarily the same because the number of observation points in each time stage is different. In comparison, FastGWR does not need to consider this issue. (4) F-STWR uses a method of caching on the MPI nodes, which can avoid repeatedly calculating the spatial distance matrix and the time distance matrix during the calibration process. To obtain the optimized adjustable parameter (a parameter for weighting the influence from local space and time on the regression point) in STWR, the F-STWR adopts a searching method (e.g., golden section or equal interval) to try and calculate the combined spatiotemporal weight values. In the process, the weight value needs to be calculated by the updated and the spatial distance and temporal distance matrixes for each step. If there is not a strategy for caching, there will be many unnecessary and repeated calculations. In comparison, FastGWR does not consider the combined spatiotemporal effect, so there is not caching design in its parallel algorithm. (5) In addition to parallel computing, F-STWR adopts a matrix splitting strategy for memory saving, and this strategy can help generate combined matrixes to be used for model prediction. The matrix splitting strategy is not considered in current FastGWR.

6. Conclusions

The parallel calibration routine developed by this study improves the ability of STWR for processing large-scale spatiotemporal data. The resulting F-STWR can leverage existing computing resources (processor cores and memory) and solve many challenges that large-scale data bring to the original STWR model (Que et al., 2020). Some features of F-STWR and STWR are summarized below.

(1) Runtime may be related to the number of observation points in the latest time stage.

(2) STWR can easily reach its bottleneck during the calibration procedure because it is computationally intensive and memory intensive. The parallel computing algorithm F-STWR is a better choice for handling datasets with a large number of observation points.

(3) With appropriate settings of the matrix splitting strategy, F-STWR can save more memory usage than STWR, making it more flexible in different memory configurations.

(4) F-STWR can obtain the consistent results of STWR, which means that it can combine the weights calculated by different parallel cores into the calibration weighted matrixes and enable further predictions out of sample points.

Extending the parallel calibration routine for multiple bandwidths of space and time will be one of the most challenging issues in our future work. Another topic of interest is to try other strategies to further speed-up the calculation of F-STWR.

Author Contribution

X.Q., X.M. and C.M. designed the algorithm together. X.Q. developed the code. Q.C. contributed to the discussion and revision of the algorithm. All authors contributed to the writing of the manuscript.

Competing Interests

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

Acknowledgement

The research presented in this paper was partially supported by the National Science Foundation under Grants No. 1835717 and No. 2019609, the China Scholarship Council under Grant No. 201807870006, the Fujian Provincial Department of Education under Grant No. JT180130, the special projects for local science and technology development guided by the central government under grant no. 2020L3006, and the Digital Fujian Environmental Monitoring Internet of Things Laboratory open fund no. 202008. The authors thank Prof. Stewart Fotheringham and other colleagues at the Spatial Analysis Research Center (SPARC) of Arizona State University for their insightful comments and suggestions during a seminar about the STWR model. We also thank three anonymous reviewers for their constructive comments and suggestions on an earlier version of the manuscript.

Code and Data Availability

The code for F-STWR and the installer of an executable program for Windows are shared on GitHub: https://github.com/quexiang/Fast-STWR/releases.

The test datasets, examples in Jupyter Notebook format, and the operation manual are also shared on GitHub: https://github.com/quexiang/Fast-STWR.

References:

Akaike, H., 1973. Maximum likelihood identification of Gaussian autoregressive moving average models. Biometrika 60, 255-265. DOI: 10.1093/biomet/60.2.255

Barney, B., 2012. OpenMP. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/mpi/

Berliner, L.M., 1996. Hierarchical Bayesian time series models. In Maximum entropy and Bayesian

methods (pp. 15-22). Springer, Dordrecht. DOI: 10.1007/978-94-011-5430-7_3

Brunsdon, C., Fotheringham, A.S. and Charlton, M.E., 1996. Geographically weighted regression: a method for exploring spatial nonstationarity. Geographical analysis, 28(4), pp.281-298. DOI: 10.1111/1467-9884.00145.

Cavanaugh, J.E., 1997. Unifying the derivations of the Akaike and corrected Akaike information criteria, Statistics & Probability Letters, 31 (2): 201–208. DOI: 10.1016/s0167-7152(96)00128-9

Cleveland, W.S., 1979. Robust locally weighted regression and smoothing scatterplots. Journal of the American statistical association 74, 829-836. DOI: 10.1080/01621459.1979.10481038

Dalcín, L., Paz, R., Storti, M. and D'Elía, J., 2008. MPI for Python: Performance improvements and MPI-2 extensions. Journal of Parallel and Distributed Computing, 68(5), pp.655-662. DOI: 10.1016/j.jpdc.2007.09.005

Dark, S.J. and Bram, D., 2007. The modifiable areal unit problem (MAUP) in physical geography. Progress in Physical Geography, 31(5), pp.471-479.

Fotheringham, A. S., Brunsdon, C., and Charlton, M., 2003. Geographically Weighted Regression: The Analysis of Spatially Varying Relationships, John Wiley & Sons.

Fotheringham, A.S., Crespo, R., Yao, J., 2015. Geographical and temporal weighted regression (GTWR). Geographical Analysis 47, 431-452. DOI: 10.1111/gean.12071

Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A. and Castain, R.H., 2004. September. Open MPI: Goals, concept, and design of a next generation MPI implementation. In European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting (pp. 97-104). Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-540-30218-6_19

507 Hoaglin, D.C. and Welsch, R.E., 1978. The hat matrix in regression and ANOVA. The American
508 Statistician, 32(1), pp.17-22.
509 Huang, B., Wu, B. and Barry, M., 2010. Geographically and temporally weighted regression for
510 modeling spatio-temporal variation in house prices. International Journal of Geographical
511 Information Science, 24(3), 383-401.
512 Hurvich, C.M., Simonoff, J.S., Tsai, C.L., 1998. Smoothing parameter selection in nonparametric
513 regression using an improved Akaike information criterion. Journal of the Royal Statistical
514 Society: Series B (Statistical Methodology) 60, 271-293. DOI: 10.1111/1467-9868.00125
515 Kumar, S., Lal, R. and Liu, D., 2012. A geographically weighted regression kriging approach for
516 mapping soil organic carbon stock. Geoderma, 189, pp.627-634.
517 https://doi.org/10.1016/j.geoderma.2012.05.022
518 Li, Z., Fotheringham, A.S., Li, W. and Oshan, T., 2019. Fast Geographically Weighted Regression
519 (FastGWR): a scalable algorithm to investigate spatial process heterogeneity in millions of
520 observations. International Journal of Geographical Information Science, 33(1), pp.155-175.
521 Neese, F., 2012. The ORCA program system. Wiley Interdisciplinary Reviews: Computational
522 Molecular Science, 2(1), pp.73-78. DOI: 10.1002/wcms.81
523 Pebesma, E., 2012. spacetime: Spatio-temporal data in R. Journal of Statistical Software,51, 1–30.
524 Pebesma, E. and Heuvelink, G., 2016. Spatio-temporal interpolation using gstat. RFID Journal, 8(1),
525 pp.204-218.
526 Que, X., Ma, X., Ma, C. and Chen, Q., 2020. A spatiotemporal weighted regression model (STWR v1.
527 0) for analyzing local nonstationarity in space and time. Geoscientific Model Development,
528 13(12), pp.6149-6164.

529 Schmidt, H.A., Strimmer, K., Vingron, M. and von Haeseler, A., 2002. TREE-PUZZLE: maximum
530 likelihood phylogenetic analysis using quartets and parallel computing. Bioinformatics, 18(3),
531 pp.502-504. DOI: 10.1093/bioinformatics/18.3.502
532 Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J., 1998. MPI: The Complete
533 Reference Vol. 1, The MPI Core.
534 Wikle, C.K., Zammit-Mangion, A. and Cressie, N., 2019. Spatio-temporal Statistics with R. CRC
535 Press.
536 Wrenn, D.H. and Sam, A.G., 2014. Geographically and temporally weighted likelihood regression:
537 Exploring the spatiotemporal determinants of land use change. Regional Science and Urban
538 Economics, 44, pp.60-74.
539 Wong, D.W., 2004. The modifiable areal unit problem (MAUP). In WorldMinds: geographical
540 perspectives on 100 problems (pp. 571-575). Springer, Dordrecht.
541 Wu, Y., Li, T., Sun, L. and Chen, J., 2013. Parallelization of a hydrological model using the message
542 passing interface. Environmental modelling & software, 43, pp.124-132. DOI:
543 10.1016/j.envsoft.2013.02.002
544 Yang, S.H. et al, 2019. Mapping topsoil electrical conductivity by a mixed geographically weighted
545 regression kriging: A case study in the Heihe River Basin, northwest China. Ecological Indicators,
546 102, pp.252-264. https://doi.org/10.1016/j.ecolind.2019.02.038