INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE
2024, VOL. 38, NO. 5, 803–824
https://doi.org/10.1080/13658816.2024.2326445

Taylor & Francis
Taylor & Francis Group

RESEARCH ARTICLE

SPASTC: a Spatial Partitioning Algorithm for Scalable Travel-time Computation

A. C. Michels[a,b,c], J. Park[a,e], J.-Y. Kang[d] and S. Wang[a,b,c]

[a]CyberGIS Center for Advanced Digital and Spatial Studies, Urbana, IL, USA; [b]Informatics Program, University of Illinois Urbana-Champaign, Urbana, IL, USA; [c]Department of Geography and Geographic Information Science, University of Illinois Urbana-Champaign, Urbana, IL, USA; [d]Department of Geography, Kyung Hee University, Seoul, South Korea; [e]Department of Geography and Geographic Information Science, University of North Dakota, Grand Forks, ND, USA

ABSTRACT

Travel-time computation with large transportation networks is often computationally intensive for two main reasons: 1) large computer memory is required to handle large networks; and 2) calculating shortest-distance paths over large networks is computing intensive. Therefore, previous research tends to limit their spatial extent to reduce computational intensity or resolve computational intensity with advanced cyberinfrastructure. In this context, this article describes a new Spatial Partitioning Algorithm for Scalable Travel-time Computation (SPASTC) that is designed based on spatial domain decomposition with computer memory limit explicitly considered. SPASTC preserves spatial relationships required for travel-time computation and respects a user-specified memory limit, which allows efficient and large-scale travel-time computation within the given memory limit. We demonstrate SPASTC by computing spatial accessibility to hospital beds across the conterminous United States. Our case study shows that SPASTC achieves significant efficiency and scalability making the travel-time computation tens of times faster.

ARTICLE HISTORY
Received 15 June 2023
Accepted 28 February 2024

KEYWORDS
Accessibility; cyberGIS; parallel computing; spatial domain decomposition

1. Introduction

Travel-time computation is an important research topic in geographic information science as it is foundational to addressing a variety of geospatial problems such as spatial accessibility and transportation optimization. This research focuses on demonstrating a new spatial partitioning approach to representing spatial context in travel-time computation through enabling large-scale analysis of spatial accessibility that aims to quantify access and availability of resources and services across space (Guagliardo 2004). Work exploring spatial accessibility has received significant attention with researchers examining access to healthcare resources (Guagliardo 2004, Luo 2004, Luo and Qi 2009, Kang *et al.* 2020, Park *et al.* 2023), supermarkets (Zenk *et al.* 2005), playgrounds (Smoyer-Tomic *et al.* 2004), electric vehicle (EV) charging stations

CONTACT S. Wang shaowen@illinois.edu

(Park *et al.* 2021) and green spaces (Zhang *et al.* 2011, Dai 2011). While there are a variety of models for computing spatial accessibility, most modern approaches – gravity models (Joseph and Kuby 2011), floating catchment methods (Luo and Wang 2003, Luo 2004, Luo and Qi 2009, Luo and Whippo 2012) and Rational Agent Access Model (RAAM) (Saxon and Snow 2020) – rely on some form of travel-time computation.

Travel-time calculations are computationally intensive because they rely on loading large travel networks (memory intensive) and calculating shortest paths between demand and supply locations on these networks (computing intensive). While commercial APIs exist with temporally rich travel-time data, they are prohibitively expensive and the results often cannot be shared (Saxon and Snow 2020). The computational burden presented by travel-time calculations has resulted in studies exploring spatial accessibility to focus on a single city (Smoyer-Tomic *et al.* 2004, Park *et al.* 2021), a single county (Dony *et al.* 2015), a handful of counties (Luo and Qi 2009, Dai 2011) or a single state (Kang *et al.* 2020). Studies addressing scalability have used Euclidean distance measures instead of more accurate travel time-based distance (Zhang *et al.* 2011), calculated travel-time in parallel using the network of the entire spatial extent (Kang *et al.* 2020) or used commercial cloud computing to calculate travel-time catchments for each county in parallel (Saxon and Snow 2020). While high-performance computing can help to overcome the computational burden, it adds a significant technical barrier and does not resolve the root problem.

Spatial partitioning methods have been used to tackle large-scale geospatial computing problems, but existing methods may not be suitable for this problem. These methods are employed for spatial domain decomposition (Wang and Armstrong 2009) – partitioning data and work into computationally feasible chunks – which supports distributed data storage and processing (Zhou *et al.* 2007, Aji *et al.* 2013, Eldawy *et al.* 2015, Yu *et al.* 2015) and enabling large-scale spatial modeling (Parker and Epstein 2011, Shook *et al.* 2013). For travel-time computation, the memory intensity is a function of road network rather than spatial objects, so partitioning spatial objects does not necessarily resolve the computational challenge. There is considerable research on network partitioning (Buluç *et al.* 2016) and spatial network partitioning (Ji and Geroliminis 2012, Lopez *et al.* 2017), but graph partitioning is frequently an *NP*-hard problem (Buluç *et al.* 2016) and may not guarantee that the road network required for each spatial object will be in the same partition. Travel-time computation requires clustering spatial objects (e.g. points in space) and spatial context (e.g. road network) *together*.

In this article, we describe SPASTC, a Spatial Partitioning Algorithm for Scalable Travel-time Computation (SPASTC) designed to calculate driving-time catchments for spatial accessibility analysis. We describe how SPASTC clusters spatial objects and road network together in Section 2. In Section 3, SPASTC is used to measure spatial accessibility to hospital beds for the state of Illinois, the Midwest U.S., and the conterminous United States. We evaluate the algorithm's performance by deriving travel-time catchments for these spatial extents, bench-marking performance with various memory limits against an approach of loading each hospital's local road network individually. Section 4 discusses future work and other geospatial problems which may benefit from the SPASTC approach.

2. Methods

This section describes the SPASTC algorithm for clustering spatial objects and their associated road network together, explicitly representing each object's spatial context. Partitioning spatial objects in space is well-studied (Eldawy *et al.* 2015), but for travel-time computation partitioning objects without considering their spatial context can produce partitions where the spatial context data required by partitions of objects are infeasibly large. For example, a clustering algorithm may group together all the hospitals in the greater New York City area, but the combined road network around all the hospitals can become too data intensive to handle. Previous research on partitioning spatial objects based on their spatial context generally focuses on the co-locality of objects or administrative boundaries (Al Jawarneh *et al.* 2020), but many geospatial problems have complex spatial contexts that should be explicitly represented in computation. On the other hand, partitioning spatial context data (e.g. road network in this study) can be computationally intensive (Buluç *et al.* 2016) and may produce partitions of the spatial data such that the spatial context of one or more objects is split between partitions.

SPASTC partitions a set of objects based on the memory intensity of their spatial context data, yielding partitions that are computationally feasible and speeding up spatial analysis by clustering objects with similar spatial contexts. SPASTC goes through three main steps illustrated in Figure 1: identifying the spatial context of each spatial object (i.e. point in this study) (Section 2.1), combining regions with shared spatial contexts (Section 2.2) and clustering regions up to a memory limit (Section 2.3). Section 2.4 provides computational complexity and intensity analysis of the algorithm.

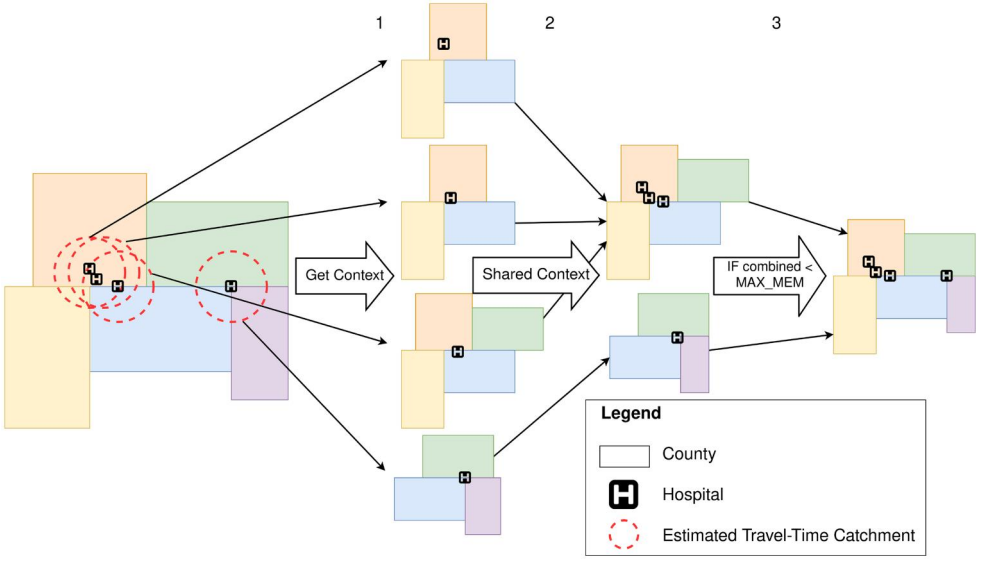

Figure 1. An example of how hospitals are partitioned across space using SPASTC: (1) for each hospital we use a Euclidean buffer to determine which county's road networks are required to calculate the hospital's driving-time catchment, (2) we combine these regions if one is a subset of another and (3) we combine overlapping regions if their estimated combined size in memory is less than a user-specified parameter.

2.1. Identifying spatial context

The first step of SPASTC is to identify or estimate the spatial context around each spatial object. This step is entirely dependent on a use case: for some applications this is straightforward, but more difficult for others (Kwan 2012). For calculating travel-time catchments, we can use an Euclidean buffer with an overestimated maximum distance to achieve this step. For example, for a 30 min travel-time catchment in an area with a maximum speed limit of 70 miles/h, we know that an Euclidean buffer of 36 miles should suffice (greater than 70 miles/h $\times$ 0.5 h). We refer to each object's spatial context as its *primary region*.

2.2. Clustering regions by spatial context

To cluster together primary regions based on shared spatial context, we employ the disjoint-set data structure as described by Galler and Fisher (1964). The disjoint-set data structure is designed to contain a collection of non-overlapping sets and specifically for computing with equivalence classes (Galler and Fisher 1964). First, we add each spatial object to the disjoint-set data structure (lines 1–4 in Algorithm 1). Then we cluster sets of objects if they share the same spatial context by comparing the primary region of each object to the primary region of each other object (lines 5–15 in Algorithm 1) If object A's primary region is a subset of object B's, we cluster object A and B, keeping object B's primary region to represent the new set. Algorithm 1 lines 1 through 15 detail how this clustering is accomplished with the disjoint-set data-structure. This step is illustrated in Figure 1 under Step 2 'Shared Context' where we see that the top two hospitals' primary regions are subsets of the third hospital's primary region and are thus clustered. Multiple objects being clustered into a single region means that these objects have the same spatial context and can be processed together without increasing their memory requirements.

Algorithm 1: Clustering Regions by Spatial Context

```
Input: objects, objectToSpatialContextMap
Output: regionToObject, regionToSpatialContexts
1  ds ← disjointSet()
2    for object ∈ objects do
3      ds.find(object)              //Adds object to disjoint set datastructure
4    end
5    for object1 ∈ objects do
6      for object2 ∈ objects do
7        primaryRegion1 ← objectToSpatialContextMap [object1]
8        primaryRegion2 ← objectToSpatialContextMap [object2]
9        if primaryRegion1 == primaryRegion2 then
10           ds.union(object1, object2)
11         else if primaryRegion2 ⊂ primaryRegion1 then
12           ds.union(object1, object2)
         /* only need to update this if the regions are not the same */
```

```
13            objectToSpatialContextMap[object2] ← primaryRegion1
14        end
15    end
16 regionToObject ← new map(regionID → list of objects)
/* iterate over the roots and their classes in Disjoint Set */
17    for root, class ∈ ds.itersets(with_canonical_elements = True) do
18        regionToObject[root] ← []
19        for object ∈ class do
20            regionToObject[root].append(object)
21        end
22    end
23 return regionToObject, objectToSpatialContextMap
```

The regions left over after this step are referred to as *secondary regions*. To keep track of which objects correspond to a given secondary region, two maps are created: (1) a map of region IDs to object IDs and (2) a map of region IDs to a list of geographic units/network files needed to calculate the travel-time catchment area for the region's objects. For our use case, this should be: (1) a map of region IDs to a list of hospital IDs and (2) a map of region IDs to a list of counties that are needed to calculate driving-time catchment areas for the hospitals in the region. The pseudocode for this operation is provided in Algorithm 1 lines 16 through 23 where these maps are created by looping over each region (the root) updating the list of objects within our region (the class). This allows us to easily lookup the objects associated with each region and the secondary region of any object. Lastly, the function returns the two maps describing the region and spatial context of each spatial object.

2.3. Clustering regions by memory

SPASTC allows users to specify a memory limit `MAX_MEM` on the memory usage of spatial context data, with the restriction that `MAX_MEM` should be greater or equal to the maximum memory usage of the secondary regions. This restriction is put into place because secondary regions are by design the smallest spatial extent required for at least one object in the region. If `MAX_MEM` is less than the memory usage of some secondary region then we could not load the data for the spatial extent required for at least one object's computation. To estimate the memory usage, SPASTC requires a pre-processing step of obtaining an estimate for the memory usage of each spatial unit. For our use case, we estimated the memory usage of each county's road network in memory by loading each county's road network into memory individually and recording the memory usage of the process.

Algorithm 2: Clustering Regions by Memory Usage

```
Input: regionToSpatialContexts, regionToObject, SpatialContextToMemory, MAX_MEM
Output: regionToObjectIDs, regionToSpatialContexts
1 stopIterating ← False
2 while not stopIterating do
```

```
3     to_merge ← list()
4     for region A in regions do
5       B ← region s. t. (A ∩ B ≠ ∅) and mem. usage of (A ∪ B) is minimized
6       combined_mem ← memory usage of (A ∪ B)
7       to_merge.append((combined_mem, A, B))
8     end
9     to_merge ← sorted(to_merge)            //sort by combined memory
      ascending
10    if to_merge[0][0] > MAX_MEM then
11      stopIterating ← True                 //lowest memory merge too big,
        exit
12    end
13    seen ← set()                  //track regions merged this iteration
14    for (combined_mem, region A, region B) in to_merge do
15      if combined_mem ≤ MAX_MEM and A, B∉ seen then
16        seen ← seen ∪{A, B}                //mark A, B as seen
17        merge(A, B)                  //updates data structures to combine
18      else if combined_mem > MAX_MEM then
19        break                  //rest of merges this iteration too big
20    end
21  end
```

Given this memory limit, we combine secondary regions using a greedy algorithm outlined in Algorithm 2. During each step of the algorithm, for each region A, we determine the non-disjoint region B such that the memory usage of the set difference between B and A ($mem(B - A)$) is minimized. This step is not memory intensive and is embarrassingly parallel. We only consider non-disjoint region A and B because if they are disjoint, no computation is saved by loading the regions together. This gives us the region B such that we minimize the memory usage of the union of A and B ($A \cup B$). Depending on the data and use case, we may expect $mem(A \cup B) < mem(A) + mem(B - A)$, especially if the pieces of data have a buffer around them to allow for composition, but this approach provides a reasonable upper-bound that is not dependent on the data or use case and the alternative would be to load, compose, and evaluate the memory usage of regions for each possible $A \cup B$ which would be extraordinarily slow.

Once an optimal region B is found for each region A, we return a tuple $(mem(A \cup B), A, B)$ for each region A in our list of secondary regions. These tuples represent suggested merges: the memory usage of the merged network and the two regions to merge. We sort the list of tuples in ascending order by the estimated combined memory usage ($mem(A \cup B)$), and for each tuple $(mem(A \cup B), A, B)$, if neither A or B has been merged with this iteration (tracked by `seen`) and $mem(A \cup B) < MAX\_MEM$, we merge regions A and B. The outer loop repeats once the list of tuples is fully iterated through or once the next suggested merge has a combined memory usage greater than `MAX_MEM`. The outer loop terminates once the first suggested merge in a step is larger than `MAX_MEM` because this signals that no more merges can be made without

exceeding MAX_MEM. These merges are tracked by continually updating our maps of region IDs to objects and region IDs to spatial units.

The resulting regions from this step are referred to as *final regions*. This algorithm is designed such that the data on the spatial extent for the final regions have a few critical characteristics that make computation feasible and efficient: (1) the size in memory is less than MAX_MEM and (2) no two final regions *A* and *B* can be merged while keeping memory usage under MAX_MEM. These partitions are ideal for distributed computation: SPASTC has split the job into *n* pieces each with data that requires MAX_MEM to load.

2.4. Computational complexity and intensity analysis of SPASTC

SPASTC employs two partitioning steps detailed in Algorithms 1 and 2. Algorithm 1, Merging Regions by Spatial Context, has complexity $O(n^2 s)$, where n is the initial number of spatial objects and s the number of partitions of spatial data, to complete comparisons between regions. This is because each of the n primary regions is compared to each of the other $n - 1$ primary regions with $O(s)$ time for each comparison. The second step of the algorithm, Merging Regions by Memory Usage, has the worst-case performance of $O(m^3 s)$ where m is the number of secondary regions. The worst-case performance arises in the case where at each step every region attempts to merge with the same region, thus only one merge happens per iteration, taking at most $m - 1$ iterations to terminate. However, typical performance is $O(m^2 \log(m) s)$ because approximately half of the regions are merged each iterations, thus taking $\log(m)$ iterations of the outer-loop. Relating the number of spatial objects n to the number of primary regions m depends on the data, but clearly the number of secondary regions is less than or equal to the number of primary regions ($m \leq n$).

However, the Big-*O* complexity analysis does not reveal the practical benefits of the approach. While these runtimes may seem large, it is important to note that for many applications the spatial context data – in our case road networks – is often much larger than the point data we are partitioning or the spatial unit data used for partitioning and this discrepancy results in massive savings. This will be demonstrated in Section 3 where the road networks are measured in tens of millions of nodes while the number of hospitals and spatial units are only in the thousands. Using the theoretical analysis from Wang and Armstrong (2009) we can gain insight into SPASTC's efficiency. Loading and composing networks requires some computation for network composition, but it is primarily data-centric with the memory and I/O costs being limiting factors. Having identified these costs as our bottleneck, SPASTC allows a user to specify a memory limit and then seeks to minimize I/O cost which is also the most time-consuming part of the computation. This analysis helps to explain the benchmarked performance in Section 3.4. Additionally, the spatial domain decomposition performed by SPASTC enables travel-time computation in parallel given adequate cyberinfrastructure support.

3. Use case: travel-time catchments and spatial accessibility

To illustrate the utility and scalability of SPASTC, we calculated spatial accessibility to hospital beds across Illinois, the Midwest United States and the conterminous United States with the E2SFCA method (Luo and Qi 2009).

3.1. Spatial accessibility method (E2SFCA)

To calculate spatial accessibility, we employ a spatial interaction model of accessibility called the Enhanced Two-Step Floating Catchment Area (E2SFCA) method (Luo and Qi 2009). This model is adapted from the Two-Step Floating Catchment Area (2SFCA) method, a special case of the gravity model which was designed to be more intuitive (Luo and Wang 2003). The 2SFCA method relies on two steps: (1) calculate a supply-to-demand ratio for each supply location and (2) each demand location sums the supply-to-demand ratios for supply locations such that the demand location is within the supply location's catchment (Luo and Wang 2003). The result is a supply-to-demand ratio for all demand points in space. For example, if demand is the general population and supply is the number of physicians, the method would calculate a ratio of physicians-to-people across the spatial extent.

E2SFCA extends the 2SFCA method by considering multiple travel-time catchments and distance decay (Luo and Qi 2009). In 2SFCA, a population is either in the travel-time catchment or not, but E2SFCA calculates multiple travel-time zones (0–10, 10–20 and 20–30 min in Luo and Qi (2009)) and applies different weights to each zone to account for distance decay. In E2SFCA's first step, we calculate the supply-to-demand ratio (R_j) for each supply location j with a weighted sum of population:

$$R_j = \frac{S_j}{\sum_{k \in \{d_{kj} \in D_r\}} P_k W_r} \tag{1}$$

where S_j is the supply at supply location j, P_k is the demand or population at demand location k and W_r is the weight for travel-time zone r (Luo and Qi 2009). Equation (1) says that the supply-to-demand ratio for each supply point j (R_j) is the supply at j (S_j) divided by the weighted sum of demand ($\sum P_k W_r$) over demand locations k that are within the rth travel-time catchment (D_r) of the supply location ($k \in \{d_{kj} \in D_r\}$) (Luo and Qi 2009).

Step 2 of the E2SFCA method is similar to Step 2 of the 2SFCA method, with each demand location summing supply-to-demand ratios, but again applies the weights to account for distance decay. The equation for Step 2 of the E2SFCA method is:

$$A_i^F = \sum_{j \in \{d_{ij} \in D_r\}} R_j W_r \tag{2}$$

where A_i^F is accessibility at demand location i, R_j is the ratio from Equation (1), and W_r are the same weights used in Equation (1) (Luo and Qi 2009). Equation (2) means that the accessibility at demand point i is the sum of the weighted supply-to-demand ratios ($R_j W_r$) over supply points j such that the distance between i and j is within a travel-time catchment of j ($j \in \{d_{ij} \in D_r\}$). The result is a ratio of supply-to-demand at each population location. We employed E2SFCA to calculate accessibility to hospital beds using travel-time catchments derived from SPASTC.

3.2. Data

Our supply dataset covers hospitals pulled from the Homeland Infrastructure Foundation-Level Data (HIFLD) website[1]. The dataset represents 7596 hospitals in the

United States. Of these hospitals, 7437 hospitals were in the conterminous United States and only 6822 hospitals were in the conterminous United States with a valid 'BEDS' field. For the travel-time analysis, we included all of the hospitals in the conterminous United States, but for the spatial accessibility, we could only use the hospitals with valid 'BEDS' fields. For demand, we used the American Community Survey's 2014–2018 (5-year) population estimates at the census tract level.

The road network data was obtained from OpenStreetMap using the Python OSMnx package (Boeing 2017). To achieve this, we used data on the cartographic boundaries for each county in the United States obtained from the Census Bureau[2]. We pulled the driving network for each county in the conterminous U.S. using the OSMnx `graph_from_polygon` function. Each file was saved using the GEOID for the county allowing us to programmatically access them. To obtain a memory estimate for each county network, we loaded each county road network into memory with OSMnx and recorded the memory usage of the Python script's memory usage. By getting the memory usage before and after loading the network, we were able to estimate the memory usage of each county road network loaded with OSMnx. While OpenStreetMap data is not perfect (Kaur *et al.* 2017), the data quality is comparable to propriety geospatial data (Sehra *et al.* 2014), provides near complete coverage in the United States (Barrington-Leigh and Millard-Ball 2017), and is widely used throughout the spatial accessibility (Kang *et al.* 2020, Saxon and Snow 2020) and geographic information science (Yan *et al.* 2020) literature.

3.3. Results

SPASTC partitions the set of hospitals and determines the necessary road network for each partition. For each final region, we load the required partition of the network, compose them together *via* the `networkx` compose operator (Hagberg *et al.* 2008). Next, we remove strongly and weakly connected components of fewer than 20 nodes in an attempt to remove erroneous data and increase the accuracy of the routing calculations. Weakly connected components are completely disconnected from the rest of the road network whereas nodes in strongly connected components do not have routes to and from the main graph (Boeing 2017). Then for each object in the final region, we calculate an ego-centric network around the object for each driving time required. Travel times on the network are determined using the `osmnx add_edge_travel_times` method which calculates free-flow travel time along each edge of the network using the length of the edge and the speed limit on the edge in kph (Boeing 2017). This means that our routing also does not consider traffic congestion and complications like rail crossings, but rather assumes that individuals are able to drive consistently at the speed limit. Once we determine the egocentric graph for each object, we calculate the convex hull around the nodes in the subgraph to obtain a driving-time polygon. Given these driving-time catchments, spatial accessibility can be calculated using any method that depends on driving-time catchments.

To perform the analysis, we utilized a SLURM (Yoo *et al.* 2003) cluster which allows us to specify memory limits on each job. One job performs the partitioning of the hospitals and then submits a job for each partition of hospitals to calculate the travel-

Table 1. Memory requirements for each region for (1) the largest primary region, the minimum amount of memory required to load each network, (2) the SPASTC algorithm, regions are compiled such that none exceeds this limit and (3) the SLURM cluster, the compute cluster will kill any job that exceeds this memory limit.

Region	Largest Primary Region	SPASTC	SLURM
Illinois	5.077 GB	5.1 GB	12 GB
Midwest	6.175 GB	6.2 GB	14 GB
Conterminous U.S.	9.402 GB	9.5 GB	20 GB

time catchments for all hospitals within the partition. Note that our `MAX_MEM` parameter in the SPASTC algorithm only describes the amount of memory required to load each region's road network and does not account for any other operations, so we must allocate more than `MAX_MEM` to each job. Additional memory requirements arise in our use case due to the `networkx` compose operator copying the network data structures, Dijkstra's algorithm (Dijkstra 1959), and storing results so to accommodate this memory usage we gave SLURM a memory limit of $2 \times \lceil MAX\_MEM \rceil$. The memory for each analysis is given in Table 1. Because this analysis already required additional memory, we utilized it to load batches of network partitions in parallel using 8 processes.

3.3.1. Illinois

Spatial accessibility of the general population to hospital beds in Illinois is given in Figure 2. While the maximum number of hospital beds per thousand people was around 37, we capped the color bar at 20 to better illustrate the variation in spatial accessibility across space. For all maps, grey indicates no access meaning that the corresponding census tract's centroid was not contained in any hospital's catchment. Most of Illinois has less than 10 hospital beds per thousand people, but Greene County in western central Illinois stands out with values in excess of 15 hospital beds per thousand people.

3.3.2. Midwest U.S

Next, we demonstrate the method's scalability by calculating E2SFCA to hospital beds for the Midwest U.S. The Midwest region used for this project is the U.S Census Bureau's Midwest Region (IA, IL, IN, KS, MI, MN, MO, ND, NE, OH, SD and WI) plus Kentucky for a total of thirteen states. Kentucky is included in our 'Midwest' to include all of the Illinois' neighbors as we scale up. Figure 3 gives hospital beds per thousand people for the Midwest using E2SFCA with grey indicating no access. We see maximum values in western Kansas, western central Illinois and south-west Ohio, with pockets of relatively high access spread throughout the Midwest.

3.3.3. Conterminous U.S

Lastly, we applied SPASTC to the Conterminous United States. While they may be hard to see at this scale, Figure 4 gives the 10-min driving-time catchments illustrated in red, 20-min driving-time catchments are illustrated in green, and the 30-min driving-time catchments are shown in blue. Using these catchments, we calculated spatial accessibility as shown in Figure 5. We see that the western U.S. except the coast has

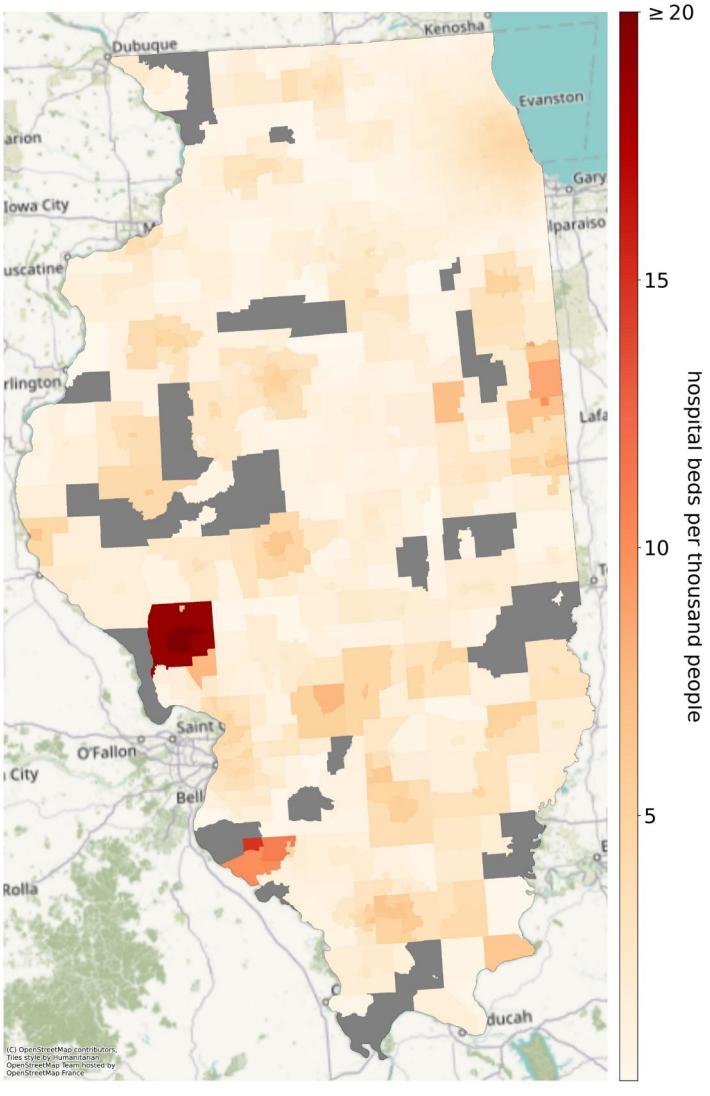

Figure 2. Enhanced Two-Step Floating Catchment Area (E2SFCA) of general population to hospital beds in Illinois. Hospitals beds per 1000 people. Grey indicates no access.

very low levels of access with pockets of high access in places that have hospitals but relatively low population density.

3.4. Performance of SPASTC

To illustrate the effectiveness of SPASTC, we compared it against loading each hospital's network separately without any spatial partitioning. Figure 6 gives the number of times each county's road network would have to be loaded to calculate the driving-time catchments to hospitals in the conterminous United States without SPASTC (a, log base 2 scale) and with SPASTC given a 16 GB limit (b, linear scale). We see that

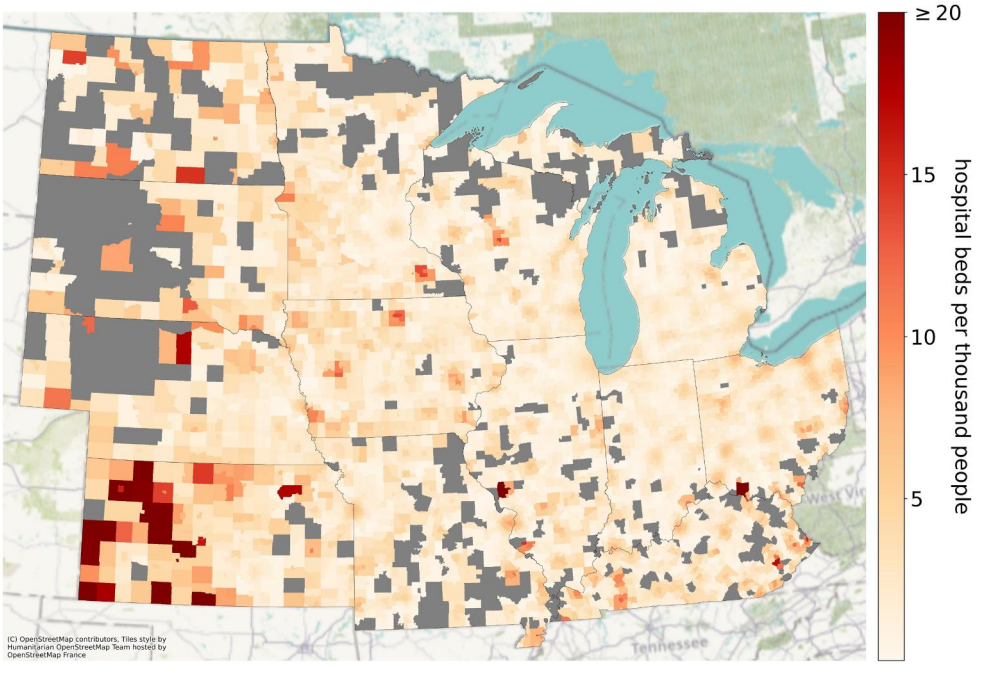

Figure 3. Enhanced Two-Step Floating Catchment Area (E2SFCA) of general population to hospital beds in the Midwest. Hospitals beds per 1000 people. Grey indicates no access.

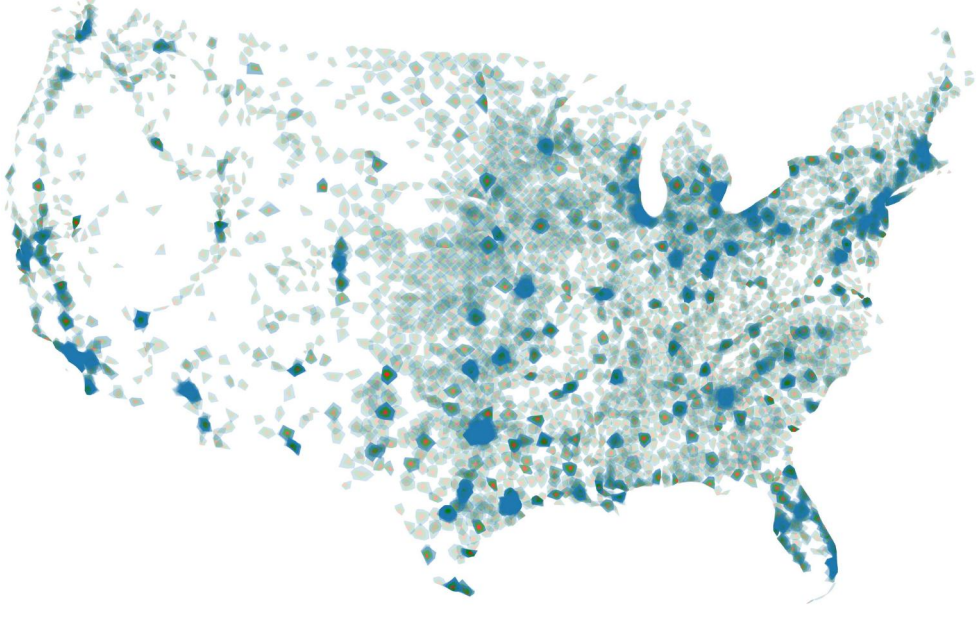

Figure 4. The calculated travel-time catchments for the conterminous United States. 10-min driving-time catchments are illustrated in red, 20-min driving-time catchments are illustrated in green and 30-min driving-time catchments are illustrated in blue.

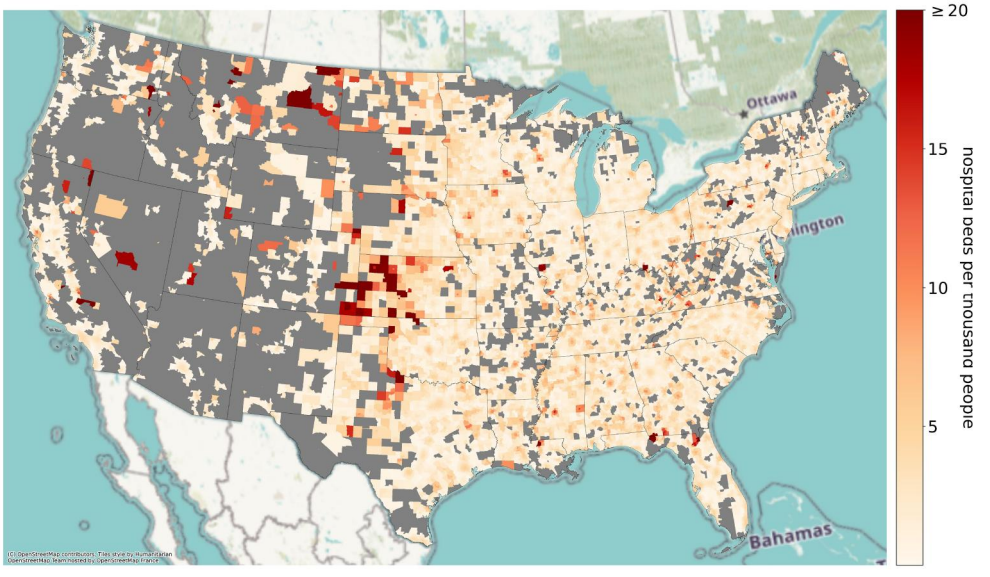

Figure 5. Enhanced Two-Step Floating Catchment Area (E2SFCA) of general population to hospital beds in the conterminous United States. Hospitals beds per 1000 people. Grey indicates no access.

without SPASTC, some road networks are loaded in excess of $2^8 = 256$ times whereas the SPASTC approach loads counties a maximum of 5 times even with a 16GB memory limit. Figures demonstrating this analysis in more depth can be seen in Appendix A.

To demonstrate the computational performance gains that SPASTC achieves, we benchmarked the performance of computing travel-time catchments for the conterminous United States using SPASTC against the same analysis without SPASTC. Both approaches used the structure laid out in Section 3: a script to load the data, determine the spatial context of each hospital, apply SPASTC or not, and then submit a job to the SLURM cluster to compute the travel-time catchments for each hospital/cluster. For benchmarking the approach without SPASTC, we submitted Bash scripts that would perform the analysis for 20 hospitals each to avoid overwhelming the job scheduler and reduce the amount of time spent submitting jobs (7437 jobs with a 1 s pause between submissions would be more than 2 h alone), but only recorded time spent in the Python script that calculates travel-time for each hospital.

In the results, we refer to this first script as 'Partitioning & Job Submission' and the travel-time calculations are referred to as 'Travel Analysis.' We had the script pause for one second between each job submission to not overwhelm the job scheduler which is included in the 'Partitioning & Job Submission' portion. We tested SPASTC with multiple memory limits shown in Table 1. Note that the memory limit given to the SPASTC algorithm is half of the limit given to SLURM except 20 GB which has a threshold of 9.5 GB; more details on this are given in Section 3.3. For both options, we allowed networks to be loaded in parallel using 8 processes. Each method was run 10 times each on the San Diego Supercomputer Center's Expanse high-performance computer.

Summary statistics for the benchmarking can be viewed in Table 2. From the statistics, we see that the mean SPASTC runtime is approximately 37–51x faster than

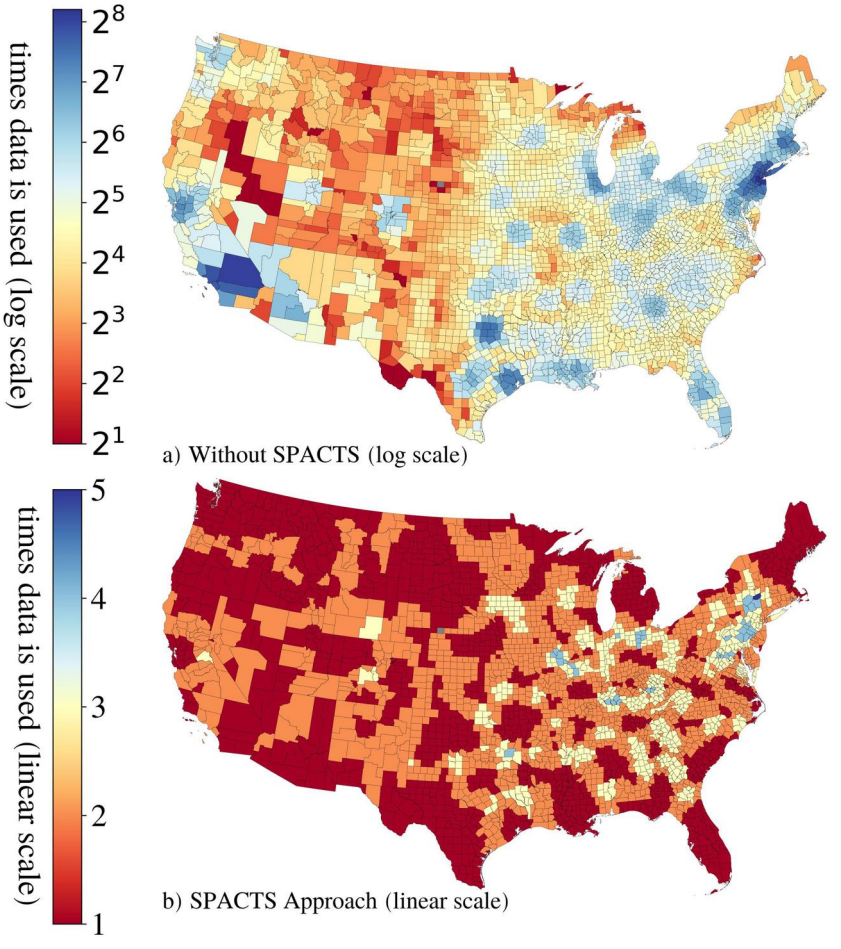

Figure 6. A diagram demonstrating the number of times each county must be loaded for (a) the approach of loading networks for each hospital separately and (b) SPASTC.

performing the analysis without partitioning. Unsurprisingly, there is a large difference in the runtimes for the travel-time analysis, with SPASTC taking a total of 11–15.5 h depending on the memory limit and the same analysis taking more than 23 d on average without SPASTC. This speed-up is due to processing hospitals with shared spatial context at the same time, savings us time in loading and composing networks while still staying under a memory limit. Interestingly, SPASTC was faster in the Job Submission portion across the board as well, meaning that time savings from submitting fewer jobs outweighed the cost of executing the partitioning algorithm for our use case. Therefore, not only does SPASTC speed up the analysis substantially, the Spatial Partitioning Algorithm is fast enough that the time savings from submitting fewer jobs outweighs the time performing the clustering.

When analyzing SPASTC performance as a function of memory – as visualized in Figure 7 – we see a substantial speed-up when moving from 20 or 26 GB to 32 or 48 GB, but there are diminishing returns and analysis time even increased slightly past 48 GB. The diminishing returns are expected because SPASTC prioritizes grouping

Table 2. Runtimes for calculating travel-time catchments at the scale of the Conterminous United States with and without applying SPASTC.

Method	Mean Partit-ioning and Job Submission	Mean travel analysis	Mean total	Turn- around	Speed up
20 G	0:07:38	15:16:10	15:23:49	0:38:17	37.06x
26 G	0:07:36	14:09:17	14:16:53	0:57:35	39.96x
32 G	0:07:09	11:32:36	11:39:45	0:58:35	48.93x
40 G	0:06:56	11:23:34	11:30:30	0:56:40	49.59x
48 G	0:06:25	10:52:46	10:59:11	1:03:11	51.94x
56 G	0:06:39	11:32:41	11:39:20	1:13:56	48.96x
64 G	0:07:01	11:12:26	11:19:27	1:23:30	50.39x
72 G	0:06:52	12:28:06	12:34:58	1:56:54	45.35x
No SPASTC	0:08:00	570:32:51	570:40:50	0:21:59	1.00x

'Partitioning & Job Submission' refers to determining the spatial context of each hospital, partitioning or not, and submitting an analysis job for each hospital/cluster. 'Travel Analysis' refers to total runtime for the travel-time analysis jobs submitted by the 'Partitioning & Job Submission' part of the analysis. 'Turnaround' is the maximum 'Partitioning & Job Submission' plus the maximum 'Travel Analysis' telling us the maximum time we would need to wait for results assuming infinite compute resources.

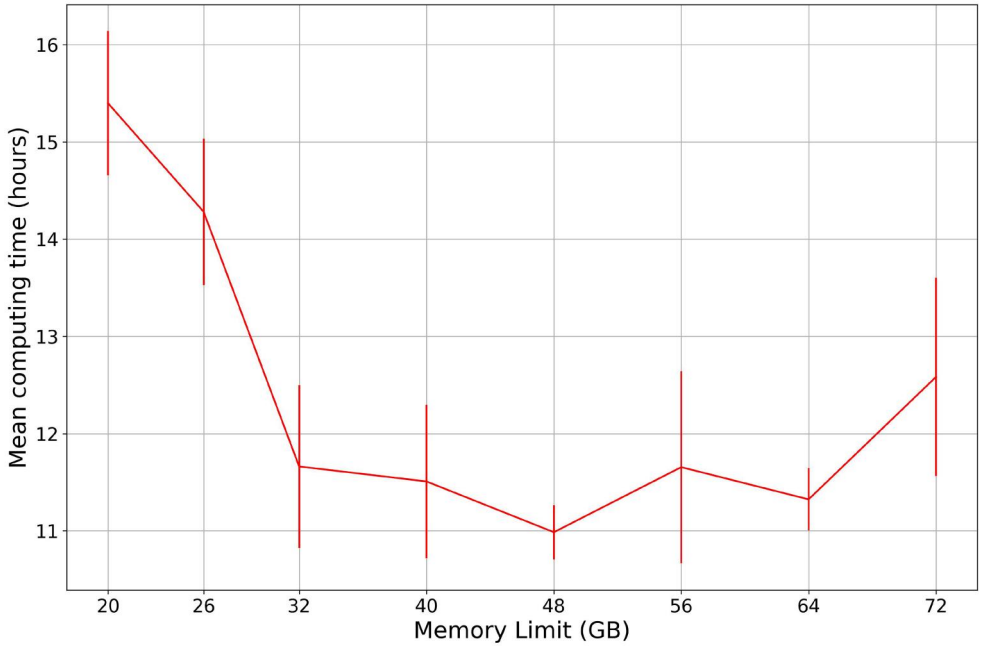

Figure 7. Mean total computing time for calculating travel-time with SPASTC on Expanse. Error bars represent standard deviation of runtime.

together tasks with high overlap for the largest potential to save time, thus there are fewer opportunities to save time as the memory limit increases. The increase in total computing time past 48 GB can be explained by the fact that the time to complete many of the operations being performed (merging graphs, checking for weakly and strongly connected components, shortest paths, etc.) scales with the size of the network. Using a smaller number of partitions also helps us to minimize the turnaround time, assuming a sufficient number of compute nodes to analyze all partitions simultaneously, if the results of the analysis are urgently needed. Using a limit of 20 GB, the

turnaround was only about 38 min compared to almost two hours at 72 GB. However, note that all these patterns are due in part to our chosen use case of calculating travel-time catchments and might not hold across other applications of the SPASTC algorithm.

4. Concluding discussion

This article describes the SPASTC. SPASTC is a novel algorithm for spatial partitioning while preserving spatial context. The algorithm accomplishes this by clustering objects in space (e.g. hospitals) and their spatial contexts (e.g. road networks) together, considering the costs (e.g. memory usage) of the spatial contexts. We applied SPASTC to compute travel-time catchments and spatial accessibility to hospital beds for the state of Illinois, the Midwest, and the conterminous United States. Lastly, we benchmarked the performance of SPASTC *vs.* analyzing each hospital seperately and found that SPASTC can perform more than fifty times faster even with the same memory limits.

SPASTC has made two interrelated contributions to advancing theoretical frontiers in geographic information science. The first contribution addresses the question of why spatial is special for evaluating computational intensity of geospatial analysis (Wang and Armstrong 2009). The SPASTC approach demonstrates that spatial partitioning based on computational constraints such as memory limits is critical to enabling optimal computation of large- and multi-scale geospatial analysis. The second contribution paves a way to represent shared spatial context by explicitly capturing spatial relationships encoded in computational design for geospatial analysis. This representation provides a computational foundation for addressing the Uncertain Geographic Context Problem (UGCoP) (Kwan 2012) particularly for the purpose of conducting data-intensive geospatial analysis.

Our research has gained three major aspects of insights for guiding the use of SPASTC. The first is that SPASTC improves the performance of geospatial computing workflows that have overlapping work, in our case data to load and process, but tasks that have no overlapping data or processing may not benefit from the approach. Second, if there are many tasks that are not computationally intensive, the time savings from SPASTC may not be enough to justify the time used for partitioning the tasks. Lastly, if we had enough compute nodes to run all of our jobs and could submit them all instantly, there is a trade-off between turnaround (the time between submitting the job and getting results) and total computing time. For our use case, the approach of computing without SPASTC took 37–51 times more total resources and it could in theory give us our result in about half the time, but note that even Expanse could not handle all the jobs at once[3], as we could not submit thousands of jobs instantly[4]. It may not be worth using 37–51 times more resources to get results twice as fast.

For computing travel time, the algorithm may have room for improvement and further utility. For our use case, we only considered driving, but additional modes of transportation like walking, biking and public transit could be considered. Our case study utilized OpenStreetMap data which has limitations and its accuracy varies spatially (Zhang and Malczewski 2017, Herfort *et al.* 2023). However, the SPASTC algorithm

is not specific to OSM data and could be used with higher quality governmental and commercial road network datasets where they are available. While OpenStreetMap and Dijkstra's algorithm are used throughout the literature for travel-time analysis (Saxon and Snow 2020, Kang *et al*. 2020), these methods are not perfect and further work should explore how the accuracy of this approach compares to proprietary datasets like Google Maps API. The buffer estimate used to calculate primary regions can also be fine-tuned and possibly even vary across space based on each area's speed limits and traffic congestion. In Algorithm 2, we estimate the memory usage of a composition of networks by summing the memory usage of the partitions of the network giving us an upper-bound on memory usage, but there may be room for improvement in this process. Further research could explore how best to determine memory limits, as this likely varies by the objective (minimizing computing *vs*. turnaround time), the hardware configuration, the data, and the operations performed on the data. Lastly, while we utilized SPASTC to derive driving-time catchments around hospitals, the algorithm could find use in other travel-time calculations like deriving travel-time matrices or analyzing mobility data.

While SPASTC was demonstrated in this research for scalable spatial accessibility computation, it could be applied to other use cases. Models and analyses that require partitioning objects in space which depend on a memory-intensive spatial context may benefit from SPASTC, such as spatially explicit agent-based models. While certain steps like estimating an object's spatial context may depend on each specific application, the framework of partitioning objects based on shared spatial context and then memory usage of the spatial context data is generalizable to work in a variety of geospatial analysis and modeling settings. The algorithm may also find use for partitioning data for distributed workflows, such as database sharding (Bagui and Nguyen 2015) or spatial resilient distributed datasets (SRDD) in GeoSpark (Yu *et al.* 2015).

Notes

1. https://hifld-geoplatform.opendata.arcgis.com/datasets/hospitals/explore
2. https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html
3. Expanse currently allows up to 4096 queued and running jobs per user: https://www.sdsc.edu/support/user_guides/expanse.html#running
4. SLURM supports *up to* 500 per second: https://slurm.schedmd.com/high_throughput.html

Acknowledgments

This work used the Expanse high-performance computer at the San Diego Supercomputer Center through allocation CIS230031 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF grants #2138259, #2138286, #2138307, #2137603 and #2138296.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Authors' contributions

The contributions of Alexander Michels to this article include conceptualization, methodology, software, visualization and writing. Jinwoo Park's contributions include conceptualization, validation and writing. Jeon-Young Kang's contributions include conceptualization, data curation and writing. Shaowen Wang's contributions include conceptualization, methodology, writing, supervision, project administration and funding acquisition.

Funding

This work was supported by the Institute for Geospatial Understanding through an Integrative Discovery Environment (I-GUIDE) funded by the National Science Foundation (NSF) under grant number 2118329 and the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Notes on contributors

Alexander Michels is a Ph.D. student advised by Dr. Shaowen Wang at the University of Illinois Urbana-Champaign. His research focuses on cyberGIS and spatial accessibility. Twitter: @alexcmichels, Mastodon: @alexmichels@mastodon.online

Jinwoo Park is an Assistant Professor in the Department of Geography and Geographic Information Science at the University of North Dakota. By harnessing the power of data-rich environments and advanced cyberinfrastructure, his research delves into intricate relationships between urban environments and the availability of essential resources, shedding light on how the dynamic nature of accessibility impacts various aspects of society.

Jeon-Young Kang is an Assistant Professor in the Department of Geography at Kyung Hee University, South Korea. His research interests lie broadly in GIScience, Geospatial Data Analytics, and Geospatial Artificial Intelligence, and his work to date has been centered on geospatial simulation and modeling.

Shaowen Wang is a Professor of the Department of Geography and Geographic Information Science; and an Affiliate Professor of the Department of Computer Science, Department of Urban and Regional Planning, and School of Information Sciences at the University of Illinois Urbana-Champaign (UIUC). He has served as the founding director of the CyberGIS Center for Advanced Digital and Spatial Studies at UIUC since 2013. His research focuses on advancing cyberGIS and geospatial data science for scalable solutions to complex geospatial problems and sustainability challenges. Twitter: @swuiuc

ORCID

A. C. Michels http://orcid.org/0000-0002-7357-5206
S. Wang http://orcid.org/0000-0001-5848-590X

Data and codes availability statement

The data and codes that support the findings of this study are available with the identifier(s) at this link https://doi.org/10.6084/m9.figshare.23519190.

References

Aji, A., *et al.*, 2013. Demonstration of Hadoop-GIS: a spatial data warehousing system over MapReduce. *Proceedings of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems*, November. New York, NY: Association for Computing Machinery, 528–31. SIGSPATIAL'13; Available from: https://doi.org/10.1145/2525314.2525320.

Al Jawarneh, I.M., *et al.*, 2020. Locality-preserving spatial partitioning for geo big data analytics in main memory frameworks. *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 7–11 December 2020 Taipei, Taiwan. IEEE, 1–6.

Bagui, S. and Nguyen, L.T., 2015. Database sharding: to provide fault tolerance and scalability of big data on the cloud. *International Journal of Cloud Applications and Computing*, 5 (2), 36–52.

Barrington-Leigh, C. and Millard-Ball, A., 2017. The world's user-generated road map is more than 80% complete. *PLoS One*, 12 (8), e0180698.

Boeing, G., 2017. Osmnx: new methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65, 126–139. Available from: https://www.sciencedirect.com/science/article/pii/S0198971516303970.

Buluç, A., *et al.*, 2016. *Recent advances in graph partitioning*. Berlin, Germany: Springer.

Dai, D., 2011. Racial/ethnic and socioeconomic disparities in urban green space accessibility: where to intervene? *Landscape and Urban Planning*, 102 (4), 234–244. Available from: https://www.sciencedirect.com/science/article/pii/S0169204611001952.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1), 269–271. Available from: https://doi.org/10.1007/BF01386390.

Dony, C.C., Delmelle, E.M., and Delmelle, E.C., 2015. Re-conceptualizing accessibility to parks in multi-modal cities: a variable-width floating catchment area (vfca) method. *Landscape and Urban Planning*, 143, 90–99. Available from: https://www.sciencedirect.com/science/article/pii/S016920461500136X.

Eldawy, A., Alarabi, L., and Mokbel, M.F., 2015. Spatial partitioning techniques in SpatialHadoop. *Proceedings of the VLDB Endowment*, 8 (12), 1602–1605. Available from: https://doi.org/10.14778/2824032.2824057.

Galler, B.A. and Fisher, M.J., 1964. An improved equivalence algorithm. *Communications of the ACM*, 7 (5), 301–303. Available from: https://doi.org/10.1145/364099.364331.

Guagliardo, M.F., 2004. Spatial accessibility of primary care: concepts, methods and challenges. *International Journal of Health Geographics*, 3 (1), 3. Available from: https://doi.org/10.1186/1476-072X-3-3.

Hagberg, A.A., Schult, D.A., and Swart, P.J., 2008. Exploring network structure, dynamics, and function using networkx. *In:* G. Varoquaux, T. Vaught, and J. Millman, eds. *Proceedings of the 7th python in science conference*. Pasadena, CA: OSTI, 11–15.

Herfort, B., *et al.*, 2023. A spatio-temporal analysis investigating completeness and inequalities of global urban building data in OpenStreetMap. *Nature Communications*, 14 (1), 3985.

Ji, Y. and Geroliminis, N., 2012. On the spatial partitioning of urban transportation networks. *Transportation Research Part B: Methodological*, 46 (10), 1639–1656. Available from: https://www.sciencedirect.com/science/article/pii/S0191261512001099.

Joseph, L. and Kuby, M., 2011. Gravity modeling and its impacts on location analysis. *Foundations of location analysis*. Berlin, Germany: Springer, 423–443.

Kang, J.Y., *et al.*, 2020. Rapidly measuring spatial accessibility of covid-19 healthcare resources: a case study of Illinois, USA. *International Journal of Health Geographics*, 19 (1), 36.

Kaur, J., *et al.*, 2017. Systematic literature review of data quality within OpenStreetMap. *2017 International conference on next generation computing and information systems (ICNGCIS)*, December 2017Jammu, India. IEEE, 177–182.

Kwan, M.P., 2012. The uncertain geographic context problem. *Annals of the Association of American Geographers*, 102 (5), 958–968.

Lopez, C., *et al.*, 2017. Spatiotemporal partitioning of transportation network using travel time data. *Transportation Research Record: Journal of the Transportation Research Board*, 2623 (1), 98–107. Available from:

Luo, W., 2004. Using a GIS-based floating catchment method to assess areas with shortage of physicians. *Health & Place*, 10 (1), 1–11. Available from: https://www.sciencedirect.com/science/article/pii/S1353829202000679.

Luo, W. and Qi, Y., 2009. An enhanced two-step floating catchment area (e2sfca) method for measuring spatial accessibility to primary care physicians. *Health & Place*, 15 (4), 1100–1107. Available from: https://www.sciencedirect.com/science/article/pii/S1353829209000574.

Luo, W. and Wang, F., 2003. Measures of spatial accessibility to health care in a gis environment: synthesis and a case study in the Chicago region. *Environment and Planning B, Planning & Design*, 30 (6), 865–884. Available from: https://doi.org/10.1068/b29120.

Luo, W. and Whippo, T., 2012. Variable catchment sizes for the two-step floating catchment area (2SFCA) method. *Health & Place*, 18 (4), 789–795. Available from: https://www.sciencedirect.com/science/article/pii/S1353829212000640.

Park, J., *et al.*, 2021. Leveraging temporal changes of spatial accessibility measurements for better policy implications: a case study of electric vehicle (EV) charging stations in Seoul, South Korea. *International Journal of Geographical Information Science*, 36 (6), 1185–1204.

Park, J., *et al.*, 2023. Daily changes in spatial accessibility to ICU beds and their relationship with the case-fatality ratio of COVID-19 in the state of Texas, USA. *Applied Geography (Sevenoaks, England)*, 154, 102929.

Parker, J. and Epstein, J.M., 2011. A distributed platform for global-scale agent-based models of disease transmission. *ACM Transactions on Modeling and Computer Simulation: a Publication of the Association for Computing Machinery*, 22 (1), 2–25. Available from: https://doi.org/10.1145/2043635.2043637.

Saxon, J. and Snow, D., 2020. A rational agent model for the spatial accessibility of primary health care. *Annals of the American Association of Geographers*, 110 (1), 205–222. Available from: https://doi.org/10.1080/24694452.2019.1629870.

Sehra, S.S., Singh, J., and Rai, H.S., 2014. A systematic study of OpenStreetMap data quality assessment. *2014 11th International conference on information technology: new generations,* December 2017 Las Vegas, NV, USA. IEEE, 377–381.

Shook, E., Wang, S., and Tang, W., 2013. A communication-aware framework for parallel spatially explicit agent-based models. *International Journal of Geographical Information Science*, 27 (11), 2160–2181.; Available from: https://doi.org/10.1080/13658816.2013.771740.

Smoyer-Tomic, K.E., Hewko, J.N., and Hodgson, M.J., 2004. Spatial accessibility and equity of playgrounds in Edmonton, Canada. *Canadian Geographies / Géographies Canadiennes*, 48 (3), 287–302. Available from: https://doi.org/10.1111/j.0008-3658.2004.00061.x.

Wang, S., and Armstrong, M.P., 2009. A theoretical approach to the use of cyberinfrastructure in geographical analysis. *International Journal of Geographical Information Science*, 23 (2), 169–193. Available from: https://doi.org/10.1080/13658810801918509.

Yan, Y., *et al.*, 2020. Volunteered geographic information research in the first decade: A narrative review of selected journal articles in GIScience. *International Journal of Geographical Information Science*, 34 (9), 1765–1791.

Yoo, A.B., Jette, M.A., and Grondona, M., 2003. SLURM: simple Linux utility for resource management. *In:* D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds. *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg, Germany. Springer, 44–60. Lecture Notes in Computer Science.

Yu, J., Wu, J., and Sarwat, M., 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, November. New York, NY: Association for Computing Machinery, 1–4. SIGSPATIAL '15; Available from: https://doi.org/10.1145/2820783.2820860.

Zenk, S.N., *et al.*, 2005. Neighborhood racial composition, neighborhood poverty, and the spatial accessibility of supermarkets in metropolitan Detroit. *American Journal of Public Health*, 95 (4), 660–667. Available from: https://doi.org/10.2105/AJPH.2004.042150.

Zhang, H., and Malczewski, J., 2017. Accuracy evaluation of the Canadian OpenStreetMap road networks. *International Journal of Geospatial and Environmental Research*, 5 (2), 1–14. Available from: https://ir.lib.uwo.ca/geographypub/347.

Zhang, X., Lu, H., and Holt, J.B., 2011. Modeling spatial accessibility to parks: a national study. *International Journal of Health Geographics*, 10 (1), 31. Available from: https://doi.org/10.1186/1476-072X-10-31.

Zhou, Y., Zhu, Q., and Zhang, Y., 2007. GIS spatial data partitioning method for distributed data processing. *MIPPR 2007: remote sensing and GIS data processing and applications; and innovative multispectral technology and applications*. Vol. 6790, November. SPIE, 78–84. Available from: https://doi.org/10.1117/12.739790.full.

Appendix A. Plots of clustering results

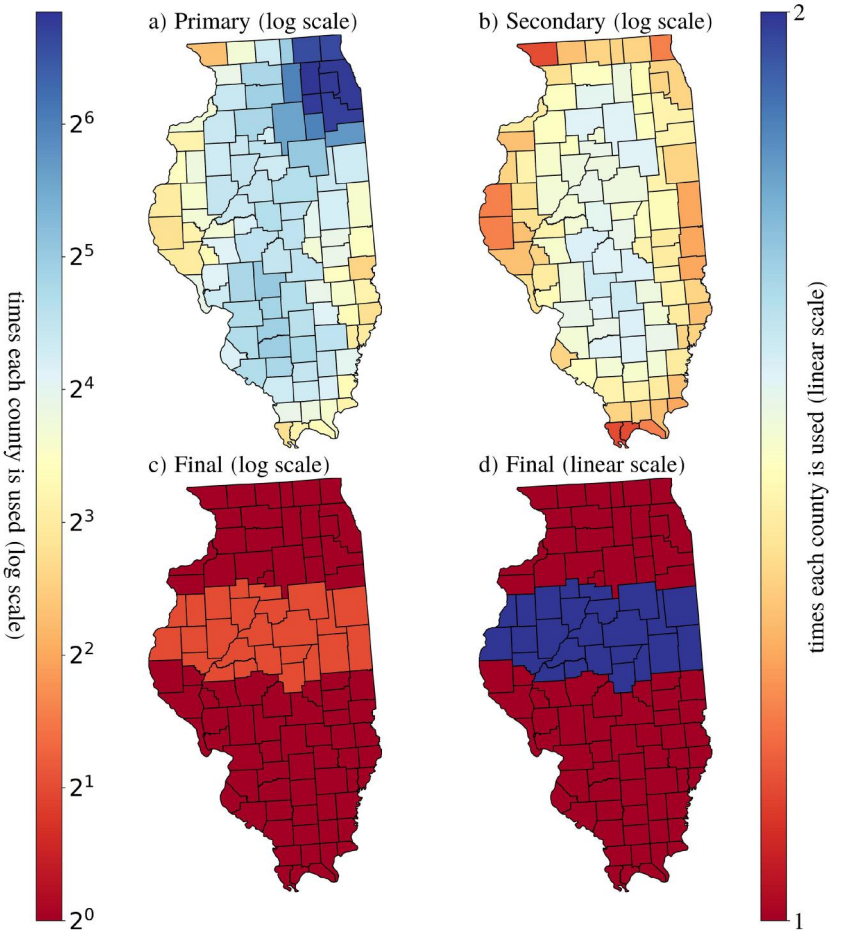

Figure A1. A diagram demonstrating the number of clusters each county is in for each stage of the SPASTC algorithm for Illinois with 8 GB memory limit. The Primary regions (a), Secondary regions (b) and Final regions (c) are given in log base 2 scale (left colorbar) while the Final regions in (d) are given in linear scale (right colorbar).

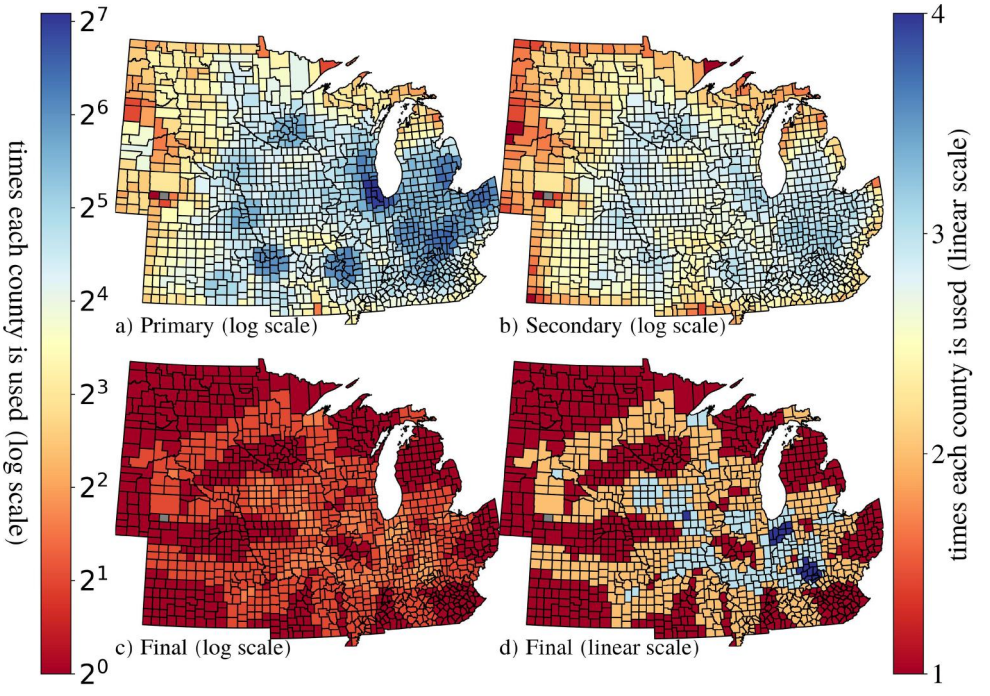

Figure A2. A diagram demonstrating the number of clusters each county is in for each stage of the SPASTC algorithm for the Midwest with 12 GB memory limit. The Primary regions (a), Secondary regions (b) and Final regions (c) are given in log base 2 scale (left colorbar) while the Final regions in the (d) are given in linear scale (right colorbar).

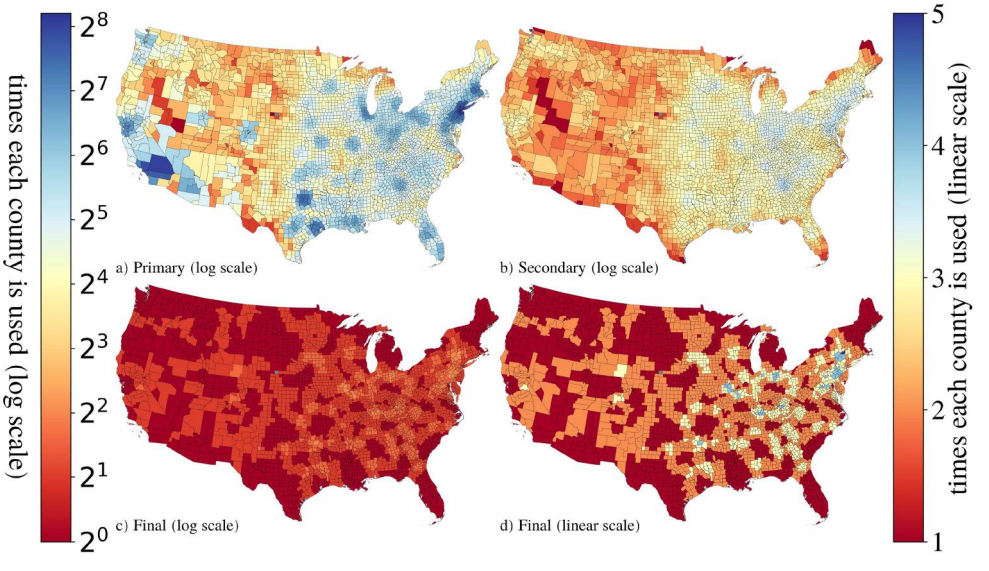

Figure A3. A diagram demonstrating the number of clusters each county is in for each stage of the SPASTC algorithm for the Conterminous United States with 16 GB memory limit. The Primary regions (a), Secondary regions (b) and Final regions (c) are given in log base 2 scale (left colorbar) while the Final regions in the (d) is given in linear scale (right colorbar).