PAGE: Parallel Scalable Regionalization Framework

HUSSAH ALRASHID, YONGYI LIU, and AMR MAGDY, University of California, Riverside, USA

Regionalization techniques group spatial areas into a set of homogeneous regions to analyze and draw conclusions about spatial phenomena. A recent regionalization problem, called MP-regions, groups spatial areas to produce a maximum number of regions by enforcing a user-defined constraint at the regional level. The MP-regions problem is NP-hard. Existing approximate algorithms for MP-regions do not scale for large datasets due to their high computational cost and inherently centralized approaches to process data. This article introduces a parallel scalable regionalization framework (*PAGE*) to support MP-regions on large datasets. The proposed framework works in two stages. The first stage finds an initial solution through randomized search, and the second stage improves this solution through eficient heuristic search. To build an initial solution eficiently, we extend traditional spatial partitioning techniques to enable parallelized region building without violating the spatial constraints. Furthermore, we optimize the region building eficiency and quality by tuning the randomized area selection to trade off runtime with region homogeneity. The experimental evaluation shows the superiority of our framework to support an order of magnitude larger datasets eficiently compared to the state-of-the-art techniques while producing high-quality solutions.

CCS Concepts: • Information systems → Geographic information systems;

Additional Key Words and Phrases: Max-p regions, max-p regionalization, spatial clustering, regionalization, spatial regionalization

ACM Reference format:
Hussah Alrashid, Yongyi Liu, and Amr Magdy. 2023. PAGE: Parallel Scalable Regionalization Framework. *ACM Trans. Spatial Algorithms Syst.* 9, 3, Article 21 (September 2023), 26 pages.
https://doi.org/10.1145/3611011

1 INTRODUCTION

Regionalization is the process of clustering a set of spatial polygons into spatially contiguous regions that meet given user constraints [17]. Each spatial polygon represents a fundamental spatial area, e.g., city block, city, or county. An example of regionalization is clustering cities in California into regions so that each region has at least 30K COVID-19 infections. Regionalization has many real-world applications including epidemic analysis [5], weather temperature classification [18], and spatial crowdsourcing [12]. The traditional problem formulations [15, 17, 34] put a major hurdle on users to input the number of regions *p*. Such a hurdle introduced the challenging "spatial scale problem," as users fail to determine the appropriate spatial scale according to the underlying

This work is partially supported by the National Science Foundation, USA, under grants IIS-2237348, SES-1831615, and CNS-2031418, the Google-CAHSI research grant, and the Saudi Arabian Cultural Mission SACM.
Authors' address: H. Alrashid, Y. Liu, and A. Magdy, Department of Computer Science and Engineering, 351 Winston Chung Hall, University of California Riverside, CA 92521, USA; emails: {halra004, yliu786}@ucr.edu, amr@cs.ucr.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2374-0353/2023/09-ART21 $15.00
https://doi.org/10.1145/3611011

data. For example, in the United States, if the studied phenomenon is at a county level, $p \approx 3{,}000$, and if it is at a state level, $p = 50$, determining such p value by the users ahead of submitting the query is challenging and limits their analysis capability. A new formulation, called *max-p-regions* (or MP-regions for short), has recently addressed the spatial scale problem [14] and become popu-lar in different applications, including urban spatial structures [3], crime analysis [37], and spatial uncertainty [47]. Instead of forcing users to input p, the MP-regions automatically discover the maximum value of p based on a user-defined constraint on a certain attribute. The regions are grown based on the user-defined constraint, which naturally generates the maximum number of regions. Generating the maximum number of regions ensures that the dissimilarity of the regions is minimized since each region will have a smaller number of areas as the p value increases. This gives the user more tractable and flexible expressiveness. However, it suffers from severe scalability limitations and cannot support even moderate-sized datasets.

MP-regions are an NP-hard problem. So, finding an exact solution is prohibitively expensive and impractical in real applications. Besides, existing approximate algorithms support relatively small datasets. For ⍰3,000 US counties, the proposed techniques in [14, 51] consume from 2.5 minutes up to several hours, depending on the underlying constraints, for each single regionalization query. In fact, real applications already use datasets that are an order of magnitude larger than this dataset. Therefore, such ineficient time makes existing techniques incapable of supporting many real-world applications. For example, such limitation is crystal clear when MP-regions are applied to reduce the uncertainty of American Community Survey data [47], which help determine how more than $675 billion in US federal and state funds are distributed each year [10] and are heavily used by community organizations, governmental agencies, and social scientists in various disciplines [11]. It is explicitly stated in [47] that "*Using the algorithm (meaning MP-regions) requires a tradeoff that is not appropriate in all situations or for all audiences - one must be willing to reduce the number of geographic units of analysis.*" Similar examples repeat in studying neighborhoods [38, 51] and studying regional poverty [16]. This scalability limitation urges the need to build scalable regionalization algorithms to support large-sized data.

This article proposes *PAGE* (Parallel Scalable Regionalization Framework), a system-level framework that scales up MP-regions on large spatial datasets. *PAGE* is an extension of *SMP* [2] that provides higher-quality solutions through data partitioning and parallelization, which solves the tradeoff between the runtime and the solution's quality. In *SMP*, the quality of the solutions is considered in later stages. On the other hand, *PAGE* works on the solution's quality in earlier stages to provide high-quality initial solutions eficiently. It introduces a low-overhead spatial partitioning technique that enables the parallelization of the regionalization process. *PAGE* provides seamless and eficient parallelization for MP-regions exploiting the computation power of multi-core processors that are widely available in regular machines. So, users seamlessly use *PAGE* through the same application programming interfaces (APIs) of regional science libraries [40], while they can analyze significantly larger datasets compared to the capabilities of existing techniques [14, 46, 51]. Such seamless integration with existing spatial data science tools is identified by the periodic community report on database research [1] as a major way for advancing data management contributions to data science tools and widening their impact on layman users. We present our proposed framework on multi-core machines for easy integration with existing spatial data science tools. However, this framework is easily generalizable to other parallel frameworks on multiple machines.

Scaling up MP-regions faces several challenges. First, as MP-regions are an NP-hard problem, exploring all solutions and finding an optimal solution is prohibitively expensive. Second, using standard spatial partitioning to partition the input spatial areas into smaller subsets while

maintaining spatial contiguity is not straightforward. These standard techniques partition spatial objects based on their own boundaries regardless of neighboring objects. This leads to spatially disconnected partitions and prevents parallelized techniques from producing connected regions. Third, existing techniques for MP-regions use tightly coupled steps that are not straightforward to parallelize. This makes them inherently centralized and limits supporting large datasets.

To address these challenges, we propose two variations of *PAGE* (*PAG-P* and *PAG-L*) that provide eficient and high-quality solutions for MP-regions. We propose contiguous spatial partitioning through a two-phase spatial partitioning technique that enforces spatial contiguity while still ensuring low processing overhead. This enables building eficient and fully parallelized techniques that scale up with adding more computational resources to regular machines. Then, we employ a two-stage approximate search. First, we employ novel heuristics to find an initial solution efficiently. Second, we optimize existing heuristics to improve the initial solution and provide a final solution. This approximate search avoids exhaustive exploration of the search space, so it eficiently supports large datasets that are not currently supported.

The first *PAGE* variation (i.e., *PAG-P*) grows as many randomized regions as possible in its first phase in parallel partitions. However, randomization and partitioning cause enclave areas to remain unassigned, so the following phase assigns leftover areas to existing regions to build a complete initial solution. This technique produces a large number of enclave areas. To overcome this, the second *PAGE* variation (i.e., *PAG-L*) reduces the number of enclaves and maximizes the number of regions. It builds the initial regions with minimal inter-regional gaps by selecting new borders based on a criterion of spatial compactness. Our experiments on real datasets show significant improvements in regionalization scalability and quality by saving up to 97% of query time and achieving 3× solution quality compared to existing competitors. Our contributions are summarized as follows:

- We propose a light spatial partitioning that warrants eficient parallelization for regionalization techniques.
- We propose eficient approximate parallel techniques that significantly scale up the MP-regions problem.
- We perform extensive experiments on large real datasets to show the superiority of our techniques.

The rest of the article is organized as follows. Section 2 discusses the related work. Section 3 formulates the MP-regions problem. Sections 4 through 6 introduce our proposed techniques, and Section 7 analyzes their complexity. Finally, Section 8 provides experimental evaluation, and Section 9 concludes the article.

2 RELATED WORK

The regionalization literature studies various problems under spatial clustering [20, 22, 23, 33, 49] and spatial graph partitioning [6, 13, 53, 56]. The latter is related to regionalization as spatial polygons can be modeled as a node-attributed spatial neighborhood graph, and regionalization can be expressed as a graph partitioning problem. However, the different objectives and constraints on the output sub-graphs make existing spatial graph partitioning techniques inapplicable for several regionalization problems. The problems most related to our work are the *p-regions* [7, 8, 15, 27, 29–32, 54] and the *max-p-regions* [3, 14, 16, 19, 26, 37, 41, 42, 45–47, 51]. The *p-regions* problem is related in the sense that the MP-regions problem is a successor that overcomes its limitations as previously discussed. There is an existing literature on the *p-regions* problem that ranges from building compact regions [7, 8, 29–32], network-constrained regionalization [54], and functional regions delimitation [27, 28]. Despite this literature, the fundamental change in MP-regions

compared to *p-regions* makes it inapplicable to adopt existing *p-regions* techniques. Specifically, the *p-regions* problem requires users to input the number of regions p, which makes growing the initial regions fundamentally dependant on this input that does not exist in MP-regions.

The literature on MP-regions [14] is more related to our work. This literature has problem variations with a single constraint [14, 46, 51], multiple constraints [19, 26], and network-aware regionalization [45], in addition to a variety of applications that are built on top of them [3, 16, 37, 41, 42, 47]. Existing techniques build variations of the same framework, which is also used in our proposed techniques in spirit. This framework first builds a set of regions as an initial solution, then uses a heuristic search to improve the solution quality and find a final approximate solution. The differences among these different techniques are in the way of performing the two steps, including different heuristics to build the initial regions, different types of heuristic search, processing optimizations of different steps, and the underlying spatial space, e.g., Euclidean space versus network-constrained space. Distinguished from all existing work, our work is the first to build a parallelized technique that decouples different processing steps and addresses the challenge of spatially contiguous partitioning. Our techniques significantly beat all existing techniques in different performance measures, query response time, output set size, and regions' homogeneity, and are able to support an order of magnitude larger datasets.

3 PROBLEM DEFINITION

This section provides preliminary definitions and formal definition for the MP-regions problem.

3.1 Preliminary Definitions

Definition 1 (Area). An *area* a_i is defined with four attributes: $a_i = (i, b_i, s_i, d_i)$, where i is the area identifier, b_i is an arbitrary spatial polygon that defines the area's spatial boundaries, s_i is a spatially extensive attribute, and d_i is a dissimilarity attribute.

Definition 2 (Spatially Extensive Attribute). A *spatially extensive attribute* s_i of an area a_i is an attribute whose value is divided over the smaller k sub-areas of $a_i = a_{i0}, a_{i1}, \ldots, a_{ik}$ when the area is fragmented such that $\sum_{a_{ij} \in a_i} s_{ij} = s_i$. For example, the *population* value of a county is divided over its cities, so the *population* is a spatially extensive attribute since the sum of the population value of all the cities is equal to the county population. This is the opposite of *spatially intensive attributes*, such as *temperature*, that are not divided when the spatial area is fragmented (i.e., $\sum_{a_{ij} \in a_i} s_{ij} \neq s_i$).

Definition 3 (Region). A *region* $r_i = \{a_1, a_2, a_3, \ldots, a_n\}$ is a set of spatially contiguous areas. During the region growing phase, an area is *assigned* if it belongs to a region and *unassigned* if it does not belong to any region. After the region growing phase, all the unassigned areas are marked as *enclave areas* (i.e., the areas that are not part of any region).

Definition 4 (Heterogeneity or Dissimilarity). The *heterogeneity or dissimilarity* of two areas a_i and a_j determines the degree of dissimilarity between them and is defined as the absolute difference between the dissimilarity attribute d of the two areas as follows: $|d_i - d_j|$.

3.2 Problem Formulation

The MP-regions problem is defined as follows:

Input: (1) A set of n areas: $A = \{a_0, a_1, a_2, \ldots, a_n\}$. All the areas in A are spatially contiguous and form a single spatially connected component. (2) A threshold T. (3) An objective function H.

Output: A set of regions $R = \{r_1, r_2, \ldots, r_p\}$ of size p, where each region r_i is a non-empty set of spatially continuous areas satisfying the below constraints and objectives.

Constraints:

- $1 \leq p \leq n$
- $|r_i| \geq 1, \quad \forall r_i \in R$
- $r_i \cap r_j = \emptyset, \quad \forall r_i, r_j \in R \wedge i \neq j$
- $\bigcup_{i=1}^{p} r_i = A, \quad \forall r_i \in R$
- $$\sum_{a_{ij} \in r_i} s_{ij} \geq T, \quad \forall r_i \in R \tag{1}$$

Objectives:

- Maximizing p
- Minimizing the total heterogeneity $H(R)$ in dissimilarity attributes belonging to the same region

  In this article, $H(R)$ is defined as follows:

$$H(R) = \sum_{r_i \in R} \sum_{a_{ij}, a_{ik} \in r_i} |d_{ij} - d_{ik}|. \tag{2}$$

The first two constraints impose that the output has at least one region ($p \geq 1$) and each region has at least one area ($|r_i| \geq 1$). The third and fourth constraints ensure that each area is assigned to exactly one region, so output regions are both disjoint and covering all input areas. The last constraint ensures that each region has a total value of the spatially extensive attribute s equal to at least the input threshold T, e.g., total population of each region $\geq T$.

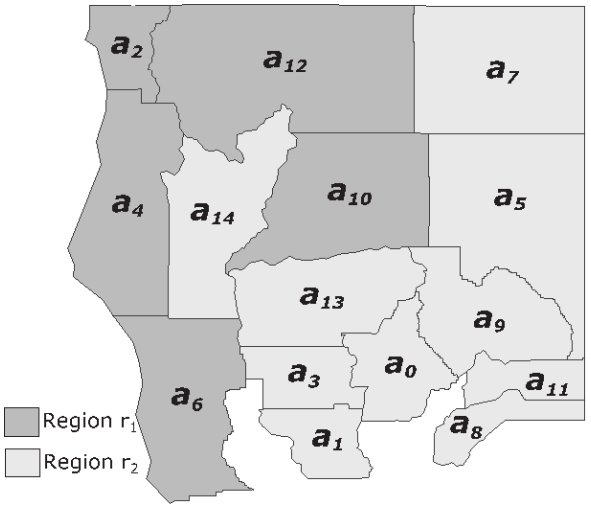
The objectives are twofold. The first objective is maximizing the number of output regions p. This is the main objective of MP-regions problems, and it is prioritized over the second objective. This objective allows to eliminate the number of regions as a user input, which addresses a major limitation in the previous regionalization problems. The second objective favors output regions to be as homogeneous as possible, measured as a function of a dissimilarity attribute. This attribute is not necessarily a spatial attribute. For example, a social scientist might need to produce regions that are homogeneous in average income level.

Figure 1 presents an example of a dataset with 15 areas where the spatially extensive attribute s is the number of houses in each area and the dissimilarity attribute d is the average house price. When $T = 140$ houses, the solution consists of two regions $r_1 = \{a_2, a_4, a_6, a_{10}, a_{12}\}$ and $r_2 = \{a_0, a_1, a_3, a_5, a_7, a_8, a_9, a_{11}, a_{13}, a_{14}\}$, where r_1 has 156 houses and r_2 has 147 houses.

4 PARALLEL SCALABLE REGIONALIZATION FRAMEWORK (PAGE) OVERVIEW

This section presents *PAGE*, which builds on a prevalent two-stage framework to address NP-hard problems, finding an initial solution and then improving it. *PAGE* consists of three phases: *region growing phase*, *region gluing phase*, and *optimization phase*. The region growing phase eficiently builds a set of initial regions with a maximum size over multiple parallel iterations. The region gluing phase finalizes building the initial regions by assigning any remaining areas that do not belong to any region. Finally, the optimization phase improves the quality of the solution.

PAGE has two variations that grow initial regions differently. The first variation employs a *partition growing phase* to build regions eficiently through data partitioning (*PAG-P*). It is designed to boost runtime scalability by exploiting the power of parallelization. The second variation em-ploys a *layered growing phase* to build regions through constrained randomization (*PAG-L*), and it is designed to optimize solution quality while maintaining a scalable runtime.



Area	s	d
a_0	12	159
a_1	18	180
a_2	31	180
a_3	32	300
a_4	52	140
a_5	13	440
a_6	20	249
a_7	10	200
a_8	11	990
a_9	20	290
a_{10}	31	150
a_{11}	9	220
a_{12}	22	720
a_{13}	9	470
a_{14}	13	230

Fig. 1. MP-regions example of a dataset with 15 areas.

As a preprocessing step to prepare the input data for regionalization, *PAGE* constructs a neighborhood graph that represents the spatial neighborhood relations among the input spatial areas. This graph is used in different phases to identify neighboring areas. Each graph node represents an area, and each graph edge connects two areas that are spatial neighbors. The neighborhood graph uses the rook spatial neighborhood relation, which entails that two spatial areas are neighbors only when they share a spatial border with more than one point, e.g., line or curve. To construct the graph, we iterate over the areas one by one and check if they intersect with any other area in the dataset to build its adjacency list.

5 PARALLEL SCALABLE REGIONALIZATION THROUGH DATA PARTITIONING (PAG-P)

PAG-P employs a *partition growing phase* that builds regions eficiently through data partitioning. Therefore, its region growing phase goes through two sub-phases: (1) a *partitioning phase* that splits data into spatially contiguous partitions and (2) a *growing phase* that builds the maximum number of initial regions for different partitions in parallel. Afterward, the initial regions are forwarded to the *region gluing phase* to finalize an initial solution, and to the *optimization phase* to improve it. Each phase is outlined below.

5.1 Partitioning Phase

This section presents the partitioning phase of *PAG-P*. This phase partitions both spatial areas and the neighborhood graph. It first divides the input areas into multiple partitions in order to be processed in parallel. Then, it divides the neighborhood graph across the resulting partitions. Partitioning data could be achieved by traditional spatial partitioning techniques that partition the spatial space and place an area into a partition if its spatial boundaries overlap with the partition's spatial boundaries [25, 35, 36, 50, 52, 55]. Other techniques include quadtrees [43], which recursively partition the space into cells that has a maximum size of four; R-trees [24], which is a height-balanced tree that partitions the objects based on their minimum bounding rectangle (MBR); and R*-trees [4] and R+-trees [44], which are variants of R-trees to enhance its performance. However, quadtree and R-tree techniques are not applicable in our case for several reasons. First, they do not consider the spatial contiguity of the objects, which is a critical requirement of

MP-regions. In particular, it is common that a border area a_b is the area that glues multiple spatially contiguous clusters of areas in a certain partition DP_j. To build contiguous regions in DP_j, a_b must be assigned to DP_j specifically, and not any other partition. Otherwise, many areas in DP_j will be marked as enclaves while unnecessary, which diminishes the solution quality and makes parallel solutions incompetent. Second, it is dificult to incorporate a load balancing technique with quadtree techniques to make parallel processing more eficient. Grid partitioning provides a uniform partitioning of the space, which enables generating balanced partitions by assigning the areas that intersect with more than one partition to the partition with the lowest number of areas in our case.

To address these challenges, the partitioning process works in two stages. It first partitions the space into a spatial grid of (*rows* × *cols*) partitions, where *rows* and *cols* are parameters that are set based on the available computation cores to fully utilize the cores. For example, for a quad-core machine, *rows* and *cols* are set to two, so (*rows* × *cols*) = 4. Then, all areas that are completely enclosed within a partition DP_j are placed in DP_j. Afterward, we ensure spatial contiguity and place any remaining areas based on their placed neighbors. This placing order speeds up the partitioning process as the neighborhood list is only checked when placing the border areas.

ALGORITHM 1: Data Partitioning (*A, rows, cols*)

```
1  Input: Area set A, Int rows, Int cols.
2  Output: Set of partitions.
3  Initialization:
4  partitions = set of (rows x cols) empty partitions;
5  A^p = {}  // set of areas placed into partitions;
6  A^u = A  // set of unplaced areas;
7  Begin:
8  for each a ∈ A do
9      for each DP ∈ partitions do
10         if a lies completely inside DP then
11             DP = DP ∪ a; A^p = A^p ∪ a; A^u = A^u - a;
12             break;
13 for each DP ∈ partitions do
14     if DP has more than one connected component then
15         DP = the component with the largest number of areas;
16         A^p = A^p - remaining components;
17         A^u = A^u ∪ remaining components;
18 while A^u ≠ ∅ do
19     for each area a ∈ A^u do
20         DP^n = smallest partition that has one of a's neighboring areas;
21         DP^n = DP^n ∪ a;  A^p = A^p ∪ a;  A^u = A^u - a;
22 return partitions;
```

Algorithm 1 presents the pseudo-code for the data partitioning. It first computes the MBR for the input areas A, divides it into (*rows* × *cols*) spatial grid cells, and initializes sets of placed and unplaced areas, A^p and A^u, respectively (Lines 4–6). The grid cells represent spatial boundaries for a set *partitions* of empty partitions. Then, it performs three steps. In the first step, Lines 8–12, we assign any area $a \in A$ that intersects with a single partition $DP \in partitions$ to DP. After this first step, areas in each partition can form multiple connected components that are not spatially contiguous. To maintain the spatial contiguity, in the second step, Lines 13–17, areas in the smallest components in terms of size are put back in the unplaced set of areas A^u. Marking those areas

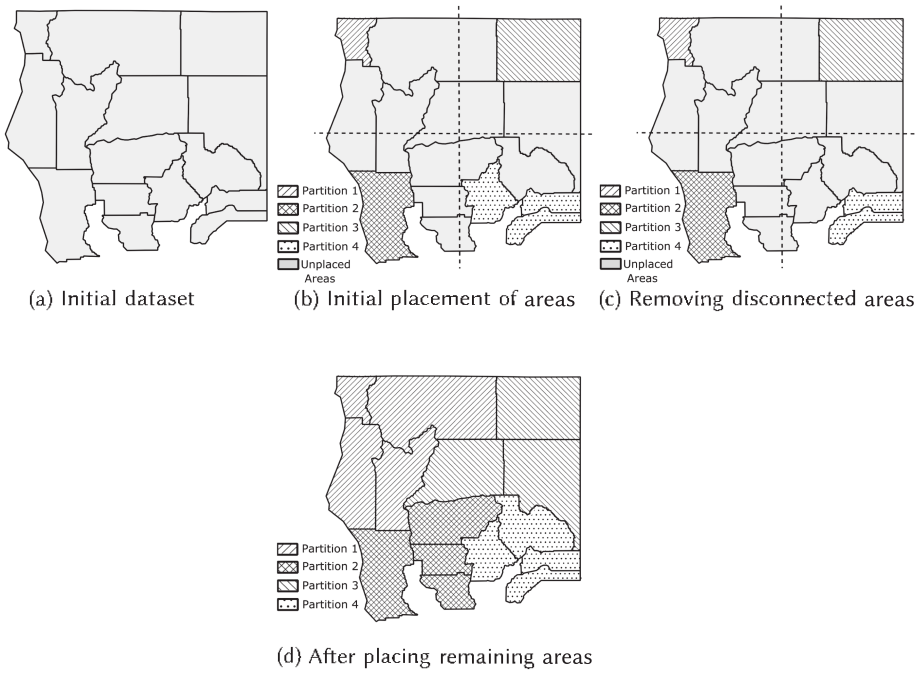


Fig. 2. Example of the *PAG-P* data partitioning phase.

as unplaced is less expensive than choosing the border areas that connect different connected components, which requires expensive spatial contiguity checks. So, it is faster to assign them again one by one based on assigned neighbors as detailed below.

The third step of Algorithm 1, Lines 18–21, assigns the remaining unplaced areas in A^u so that spatial contiguity in each partition is maintained at all times. For an unplaced area a, if a has a neighboring area that is placed in one of the partitions DP, then a is also added to DP. If a has neighbors in multiple partitions, then a is added to the partition with the least number of areas to balance partitions' sizes and improve the distributed processing performance.

Example. Figure 2 shows an example of our dataset partitioning process. The input is the initial dataset shown in Figure 2(a), which consists of 15 areas. The dataset is split into four partitions (2 rows × 2 columns). Figure 2(b) shows the four data partitions' boundaries (by the two dashed straight lines) after the initial placement of areas. At this step, only the areas that are completely enclosed within a partition are assigned to that partition. Figure 2(c) shows the partitions after removing the disconnected areas. One area is removed from Partition 4 because it is disconnected from all other placed areas in the same partition. So, it is marked as unplaced. All other unplaced areas intersect with two or more partitions. Iteratively, unplaced areas are placed into partitions of spatial neighbor areas. The final partitions after placing all the areas are shown in Figure 2(d).

The final step in this phase is splitting the neighborhood graph across partitions. Each data partition DP has its own subgraph of the neighborhood graph, which only contains its areas. This prevents an area from being assigned to a region in multiple partitions.

5.2 Region Growing Phase

Given the data partitions that are produced by the partitioning phase, the region growing phase builds regions for each partition in parallel to produce different initial solutions, from which it keeps the one with the largest number of regions p. In particular, a specific number of solutions

(referred to as *MI*) are constructed for all data partitions. To alleviate the high computational cost and support large datasets, our region growing phase supports two-level parallelism. It produces independent solutions in parallel, and for each solution, data partitions are processed in parallel as well. Such two-level parallelization operates on disjoint partitions, which gives two performance advantages that make it eficient for large datasets: (1) it is free of synchronization mechanisms, e.g., locking, and (2) it eliminates any checks to ensure that each area is assigned to only one region.

To grow regions for any partition, initially all areas within the partition are not assigned to any region and are placed in a set A^u. Then, to build a region r, a seed area is picked at random from A^u and placed in r. While the total value of the spatially extensive attribute s of all areas in r is less than T, we keep adding new areas to r. New areas are selected from the spatial neighboring areas of r that are not assigned to other regions. If multiple neighboring areas are available, we select the one that minimizes the heterogeneity of r, computed as in Equation (2). Once r meets the user-defined threshold T, we add it to the list of regions and start growing a new region in a similar way. If r fails to meet the threshold T when it has no more unassigned neighboring areas to add, all areas in r are marked as enclaves and are handled in the following phase.

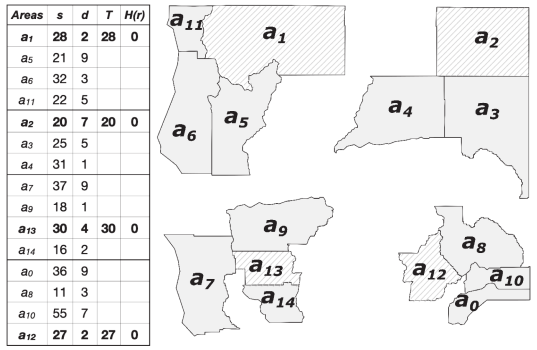
To speed up the process of searching for r's neighboring areas, we maintain a list of r's neighbors. For every new area added to r, we retrieve its neighbors from the spatial neighborhood graph and add it to the list. Only the neighbors that are unassigned and do not already exist in the list are added; therefore, maintaining the list becomes expensive as the list grows in size. In addition to parallelizing the region growing phase, *PAG-P* introduces two changes that significantly reduce the time for maintaining the list of neighboring areas for any region. First, when splitting the dataset into multiple partitions, there will be a smaller number of areas in each partition, which makes maintaining the list of neighboring areas for regions less expensive. Second, *PAG-P* uses a set data structure instead of a list to maintain the neighboring areas. Sets are faster when performing lookup, insertion, and deletion operations.

Since there is no overlap between the data partitions, the process of growing the regions is independent for each data partition. Therefore, the merge is achieved by appending all the regions and enclaves for all the partitions together. After growing regions for each partition, all the regions and enclaves of all partitions are grouped together to form a single set of regions and a single set of enclaves, which represents the initial solution. For example, if there are two partitions and the first one generates two regions and 10 enclaves and the second one generates three regions and 5 enclaves, then the regions are grouped together to form five regions and the enclaves are also grouped together to form 15 enclaves. As this is repeated *MI* times, the algorithm produces *MI* initial solutions. These solutions are compared according to the number of regions p, and all solutions with the maximum value of p are kept for the following phases.

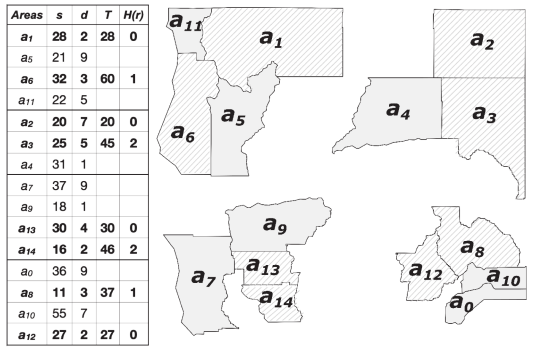
Example. Figure 3 gives an example for the region growing phase. The phase takes four data partitions as an input as shown in Figure 3(a). The table in Figure 3(a) shows the areas in each data partition, their spatially extensive attribute value s, their dissimilarity attribute value d, the region threshold T, and the region dissimilarity H. The regions start growing by randomly selecting seed areas a_1, a_{13}, a_2, and a_{12}. The table shows the change in T and H values after adding the seed area. At this point, region r_1 in Partition 1 has a single area a_1, so its H value is zero and its threshold value equals 28, which is the s value of a_1. Figure 3(b) shows adding the next area that minimizes the H value to each region. For r_1, the dissimilarity of the region equals 3 when adding a_{11}, 7 when adding a_5, and 1 when adding a_6; therefore, a_6 is added.

5.3 Region Gluing Phase

The initial solution produced by the region growing phase might include enclave areas that are not assigned to any region and introduce spatial gaps between the grown regions. The region gluing



(a) Random seed selection for growing a region in each partition



(b) Adding the area that minimizes the region's dissimilarity

Fig. 3. Example of growing regions in *PAG-P* in parallel.

phase closes these gaps by assigning the enclave areas to the existing regions. For each solution, we iterate over the list of enclave areas. For each enclave area, we list their neighbor areas from the spatial neighborhood graph, and in turn we list their neighbor regions. If it has no neighbor regions, we skip it in this step, and it is picked up again when at least one of its neighbors is assigned to a region. If it has one neighbor region, it is assigned to that region. If it has multiple neighbor regions, it is assigned to the region that gives the lowest heterogeneity value increase. We repeat this process until assigning all enclaves for all produced solutions. Then, only the solution with the minimum heterogeneity value is kept for further processing in the optimization phase.

5.4 Optimization Phase

The optimization phase optimizes the heterogeneity value *H(R)* of the initial solution. We use a heuristic search technique based on a modified simulated annealing (MSA) algorithm [51]. The technique moves areas from a region to another neighbor region so that the *H(R)* value is improved. The original simulated annealing algorithm [14] generates a set of movable areas by identifying all the areas that could be moved from one region (donor region) to the neighboring regions (recipient regions). The algorithm then selects an area to move randomly without violating the input constraints. The move is accepted if it improves the *H(R)* value of the current solution. Otherwise, the move is accepted with a probability calculated using the Boltzmann equation: $e^{-\Delta\, H/TM}$, where $-\Delta\, H$ represents the heterogeneity variation on both donor region and recipient region, and *TM* represents the temperature. The temperature *TM* is decreased at each iteration at a fixed cooling rate *PH* until *TM* reaches a predefined value. The MSA algorithm [51] reduces the overhead of

recomputing valid moves and introduces a tabu list, similar to tabu search [21], to prevent search cycles. Specifically, instead of recomputing the list of movable areas at each iteration, the algorithm reuses the same list until it becomes empty. After an area is moved from a donor region to a recipient region, all areas that come from either the donor or the recipient regions are removed to avoid expensive spatial contiguity checks.

An area is considered movable if it satisfies two conditions: (1) the donor region's threshold remains above T after removing the area, and (2) moving the area does not break the spatial contiguity of the donor region. Movable areas are identified from boundary areas of a region r that are direct neighbors to other regions and satisfy these conditions. Boundary areas are added to a list L_m. Filtering L_m to exclude areas that violate the threshold T condition is straightforward in a single pass over the region's areas. However, enforcing the second condition is a computational bottleneck due to the cost of detecting articulation areas. To detect if an area a is an articulation area naively, we remove a's node from the neighborhood graph along with its associated edges, and check whether the graph still maintains spatial contiguity. If the region is no longer spatially contiguous, then a is an articulation area. Such check is expensive; therefore, we employ Tarjan's algorithm [39] to find all the articulation areas in a single graph traversal. After identifying the boundary areas list L_m, we perform the Tarjan operation and then subtract the articulation areas from L_m. Hence, L_m contains only the feasible moves from region r to its neighbor.

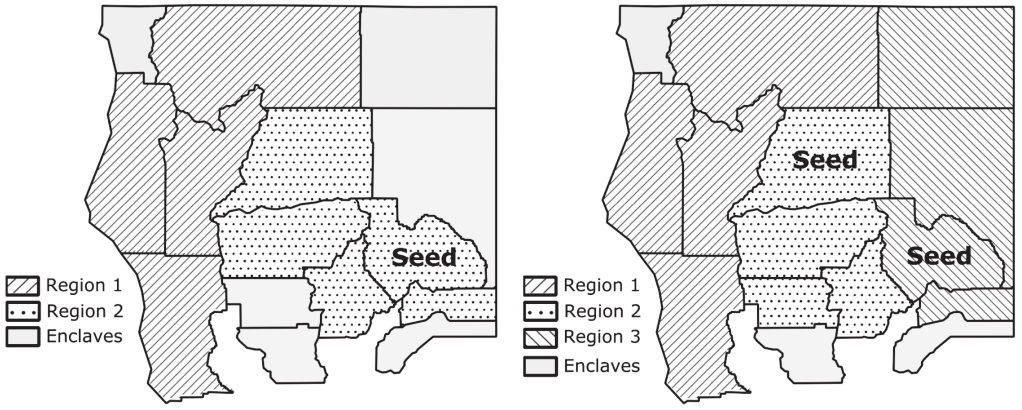
6 PARALLEL SCALABLE REGIONALIZATION THROUGH LAYERED GROWTH (PAG-L)

The second variation of *PAGE* is Parallel Scalable Regionalization through Layered Growth (*PAG-L*). It employs a *layered growing phase* to increase the number of regions p in the region growing phase while maintaining an eficient runtime similar to *PAG-P*.

As noticeable in all existing techniques that address the MP-regions problem [14, 19, 46, 51], the p value is determined in the region growing phase or its equivalent. The main observation of our *PAG-L* technique is that all existing techniques generate a large number of enclave areas while growing initial regions. These enclaves are scattered all over the space, so they leave gaps between regions but cannot form regions themselves. *PAG-L* increases the number of total regions p by reducing the number of enclaves and employing a layered-growth region growing. Figure 4 gives an example for state-of-the-art region growing versus layered-growth growing. In state-of-the-art region growing, Figure 4(a), a seed area is picked randomly for Region 1, and then the areas are added to the region until it reaches the threshold. Next, a seed area is also picked at random for Region 2, which creates a gap between the two regions and leads to having six enclave areas that are scattered in the space. The layered-growth instead groups these enclaves in one part of the space, enabling to grow three regions instead of two, and reduces the number of enclaves as depicted in Figure 4(b). This is due to the fact that the layered-growth only picks the seed area for the first region, Region 1, at random and then grows the region until it reaches the threshold. Then it picks the seed area for Region 2 from the unassigned neighboring areas for the region, which minimizes the gap between the regions and reduces the number of enclave areas.

PAG-L still consists of the same three phases of *PAGE*. Compared to *PAG-P*, *PAG-L* employs a different *region growing phase* and uses the same logic of the *region gluing phase* and *optimization phase*. The rest of this section details *PAG-L*'s region growing phase.

PAG-L increases the number of initial regions through optimizing the area selection criteria for this objective. Existing techniques, including *PAG-P*, select a random seed area every time they grow a new region. Then, the region grows based on the spatial neighbors of this seed area. The criterion of selecting a total random seed for every region does not give any guarantee about the relative spatial distribution of regions. So, on average, it scatters regions all around the space and



(a) State-of-the-art regions (b) Layered-growth regions

Fig. 4. Layered-growth impact on number of regions.

leaves a large number of enclaves in between. On the other hand, the growing strategy in *PAG-P* that adds the area that minimizes the heterogeneity of the region does not give a guarantee on the shape of the region. In several cases, it results in an arbitrary region shape that forms a gap between the unassigned areas and other growing regions, which increases the number of enclaves.

To reduce the number of enclaves that are caused by random areas, we change the criterion of choosing the seed areas and the criterion of growing the regions from *total randomness* to *constrained randomness*. At the beginning of the phase, the seed for the first region r_1 is still chosen at random. Then all the neighbor areas of r_1 are stored in a queue q_{r_1}. While growing r_1, we add the first unassigned area from q_{r_1} to the region and update q_{r_1} accordingly. This ensures that the unassigned neighbor areas of r_1 are processed and added to the region layer by layer, which leads to a compact region shape that reduces the probability of forming gaps between r_1 and other regions to be grown later. If there exists no unassigned neighbor areas but the total extensive attribute of r_1 is still below the threshold, then all the areas are considered as enclaves. Once the region reaches the threshold, it is added to the list of regions and a new region starts growing.

After the first region, for all subsequent regions, the seed area is chosen from the direct neighbor areas of existing regions. For region r_{i+1}, the seed area is determined to be the first unassigned area from the last grown region r_i's queue q_{r_i}. If all the areas in q_{r_i} are assigned, then the algorithm turns to $q_{r_{i-1}}$, $q_{r_{i-2}}$, up to q_{r_1}.

The seed selection is still randomized, as an essential requirement for a randomized algorithm that approximates an NP-hard problem. Yet, it guarantees better spatial distribution for the regions compared to total randomness, which reduces the number of enclave areas significantly and allows to build significantly more regions and maximizes p. We still run the region growing phase *MI* times and keep the result that has the maximum p. Afterward, a region gluing phase, which is described in Section 5.3, is employed to assign the remaining enclaves to the existing regions. The produced initial solutions are fed to the optimization phase described in Section 5.4 to produce the final solution.

7 COMPLEXITY ANALYSIS

This section analyzes the time and space complexity of the proposed algorithms *PAG-P* and *PAG-L*.

7.1 Time Complexity

We use $s(r)$ and $s(DP)$ to denote the number of areas in a region r and a partition DP, respectively.

For *PAG-P*, the partitioning phase takes $O(n)$ to locate areas into partitions. Then, finding the largest connected component in each partition requires traversing the partition's spatial neighborhood graph. This graph is a planar graph; therefore, for $r(DP_i)$ nodes in partition DP_i, the average degree of a vertex is strictly less than 6 and the number of edges is $3O(r(DP_i)) - 6 = O(r(DP_i))$ [48]. Thus, graph traversal takes a total of $O(n)$ for all partitions. Finally, assigning the rest of the areas to partitions takes time $O(n)$. So, the overall complexity of data partitioning is $O(n)$.

In the growing phase, as the i^{th} iteration grows region r_i, finding the unassigned neighboring area that has the minimum heterogeneity increase takes $O(s(r_i)^2)$ since we iterate over $O(s(r_i))$ neighbor areas and compute the heterogeneity increase for each area in $O(s(r_i))$. For a total of $O(n)$ iterations, the overall time complexity of $\sum_{i=1}^{O(n)} O(s(r_i)^2) = O(n^3)$. This step is executed in parallel on C cores, and the growing phase is executed MI times, which takes $\frac{MI}{C} O(n^3)$. The $O(n^3)$ complexity of the region growing phase has a very small constant due to the fact that $s(r) << n$.

In the region gluing phase, assigning an enclave area requires iterating all the neighboring regions to compute the heterogeneity increase, which takes $O(n)$. Suppose there are e enclaves, where typically $e << n$, and it takes a maximum of $O(e)$ to find an enclave area that has at least one neighboring region. Consequently, processing one enclave area takes time $O(e + n)$ in the worst case, and the overall time complexity of the phase is $e \times O(e + n) = O(e^2 + en) = O(n^2)$.

The optimization phase consists of two steps: identifying movable areas and shufling them among neighboring regions. Tarjan's algorithm is used to locate articulation areas. The complexity of Tarjan's algorithm on a planar graph of a region r is $O(s(r))$ as it traverses all nodes and edges in the worst case. So, applying Tarjan's algorithm for all regions takes $\sum_r O(s(r)) = O(n)$. Filtering out the areas that violate the threshold T constraint takes $O(n)$. Consequently, the overall complexity of the first step is $O(n)$. For the second step, while shufling each movable area a, computing the heterogeneity increase of a to all neighboring regions takes $O(n)$. In the worst case, we perform a single move before we need to repeat the first step again. These two steps repeat as long as the heterogeneity improves. So, the optimization phase takes in total $\alpha(O(n)+O(n)) = O(\alpha \times n)$, where α is the actual number of moves conducted. α is not theoretically bounded. However, empirically, α is two to three orders of magnitude of the *NI* parameter.

Consequently, the overall complexity of *PAG-P* is $\frac{MI}{C} O(n^3) + O(n^2) + \alpha O(n)$, where MI is the number of iterations of the region growing phase, C is the number of computing cores, and α is the number of the actual moves in the optimization phase.

PAG-L differs from *PAG-P* only in the region growing phase. In the region growing phase of *PAG-L*, for each area a that is added to region r_i at the i^{th} iteration, computing the heterogeneity increase takes $O(s(r_i))$. Consequently, for adding a maximum of n areas, the time complexity is $\sum_{i=1}^{n} O(s(r_i)) = O(n^2)$. The region gluing and the optimization phases of *PAG-L* are identical to *PAG-P*. Thus, the overall time complexity of *PAG-L* is $MI \times O(n^2) + \alpha O(n)$.

7.2 Space Complexity

The space complexity of both *PAG-P* and *PAG-L* is $O(N)$. This is because each area a is associated with a spatially extensive attribute and a dissimilarity attribute and storing them takes space $O(N)$. We also need to store the neighboring areas for each area. Since the spatial neighborhood graph is a planar graph, the average number of neighbors for each node, i.e., area, in the graph is strictly less than 6 [48], which gives $6 \cdot N = O(N)$ space complexity of storing neighbors for each area. Consequently, the space complexity for both algorithms is $O(N)$.

Table 1. Parameters and Values

Parameter	Values
DS	≈10, 20, 30, 35, 40, 50, 60, 70 ($\times 10^3$)
MI	20, 30, 40, 50, 60
T	50, 150, 250, 375, 500 ($\times 10^6$)
NI	10, 30, 50, 70, 90
NP	4, 9, 16, 25, 36
C	1, 2, 4

8 EXPERIMENTAL EVALUATION

This section provides an experimental evaluation for our proposed techniques. Section 8.1 introduces the experimental setup, Section 8.2 evaluates the runtime scalability, and Section 8.3 evaluates the solution quality.

8.1 Experimental Setup

We evaluate our proposed techniques *PAG-P* and *PAG-L* against three alternatives: (1) state-of-the-art technique for MP-regions problem [51], denoted as *MP*; (2) an optimized version of *MP*, denoted as *MP**, that uses a set instead of a list to maintain the regions' unassigned neighbors while growing the regions; and (3) a randomized version of *PAG-P*, denoted as *PAG-P**, that chooses areas randomly when growing the regions. We use three performance measures: *runtime* as a measure for scalability, and *number of regions p* and *heterogeneity* as measures for solution quality. All the experiments are based on Java 14 implementation and run on Ubuntu 16.04 with a quad-core 3.5GHz processor and 128GB of memory. We study the impact of the following parameters on the performance. The parameter values are shown in Table 1; the default values are emphasized in boldface.

- *DS*: dataset size, i.e., the number of areas in the dataset
- *MI*: maximum iterations for the region growing phase
- *T*: threshold value of the user-defined constraint
- *NI*: the number of iterations allowed in the optimization phase without improving the heterogeneity
- *NP*: the number of partitions used to divide the input dataset
- *C*: the number of computing cores

Regarding the tabu list length (*LI*), the temperature of computing Boltzmann's probability (*TM*), and its cooling rate (*PH*), we set them to 100, 1, and 0.9, respectively, extending the values from [14].

Evaluation datasets. We evaluate all techniques on the TIGER/Line shape files for (1) the census tracts of the US states and (2) the county subdivisions [9]. In the census tracts dataset, the areas represent the census tracts for each state. In the county subdivisions dataset, the areas represent the division of counties in the United States. In our experiment, we define the spatially extensive attribute over the *ALAND* attribute that represents the water area. The dissimilarity attribute is defined over the *AWATER* attribute that represents the land area. The county subdivisions dataset includes 35K areas and is denoted as *D*35. For the census tracts dataset, the shape files of different neighboring states are merged together to form seven datasets of increasing sizes as follows:

- *D*10: 10K areas = CA, NV, and AZ states
- *D*20: 20K areas = *D*10, OR, WA, ID, UT, MT, WY, CO, NM, OK, KS, NE, SD, and ND states

- *D*30: 30K areas = *D*20, TX, LA, AR, MO, and IA states
- *D*40: 40K areas = *D*30, MN, MS, AL, TN, KY, IL, and WI states
- *D*50: 50K areas = *D*40, GA, IN, MI, OH, and WV states
- *D*60: 60K areas = *D*50, FL, SC, NC, VA, and MD states
- *D*70: 70K areas = *D*60, PA, NY, NJ, and DE states

8.2 Runtime Scalability

This section details the impact of different parameters on the runtime scalability of different approaches.

Impact of the dataset size (*DS*). Figure 5(a) shows that our techniques *PAG-P* and *PAG-L* have linear runtime scalability with increasing dataset size ranging from 2 to 44 seconds and 0.9 to 54 seconds, respectively. On the other hand, *MP* encounters an exponentially increasing runtime ranging from 22 to 1,297 seconds. *MP** and *PAG-P** are faster than *MP* with a runtime ranging from 2 to 53 seconds and 5 to 168 seconds, respectively, but they are still slower than *PAG-P* and *PAG-L*. Our techniques are 23 to 29 times faster than *MP* for the largest dataset. The speedup ratio increases with dataset sizes. The changes introduced in both *PAG-P* and *PAG-L*, either in the region growing phase or the optimization phase, are the dominating factors in speeding up the processing.

Impact of partitioning and parallelization (*NP*, *C*). Figure 5(b) shows the impact of different *NP* values on the total runtime, and Figure 5(c) shows the impact of different *C* values on the region growing time for *PAG-P* and *PAG-P**. Running *PAG-P* and *PAG-P** with one core is equivalent to running them sequentially. Even in this case, they are still at least 20 times faster than *MP*, which takes 1,297 seconds to build a solution for the default dataset. This speedup in both *PAG-P* and *PAG-P** comes from partitioning and processing smaller subsets of data.

Increasing the *NP* value (Figure 5(b)) decreases the runtime as partitions handle a smaller subset of the dataset in parallel. However, the runtime improvement is not linear with the added partitions due to the limited number of available computing cores. Comparing the total runtime in Figure 5(b) with the region growing time in Figure 5(c), *PAG-P* grows regions slower than *PAG-P** since it aims to improve the heterogeneity when growing the regions, while *PAG-P** picks areas randomly. However, *PAG-P** is slower than *PAG-P* as *PAG-P** consumes more time in the optimization phase.

Impact of the user-defined constraint threshold (*T*). Figure 5(d) shows that increasing the *T* value slightly increases the runtime of all alternatives. Increasing *T* increases the region size, and this increases its number of neighboring and boundary areas. Increasing the number of neighboring areas makes maintaining the list of neighboring areas more expensive while growing the regions. In addition, increasing the number of boundary areas increases the shufling feasibility checks during the optimization phase. It has the most noticeable impact on *MP* as it increases its runtime from 1,096 seconds to 1,303 seconds. Meanwhile, it has much less impact on the performance of both *PAG-P* and *PAG-L*, which consistently finish in 51 seconds and 71 seconds, respectively. *PAG-P** and *MP** are less impacted than *MP*. In *PAG-P* and *PAG-P**, the cost of maintaining the neighboring areas is alleviated by partitioning and the use of a set data structure. In *PAG-L*, the speedup comes from the way it grows regions. As for *MP**, the alleviated cost comes from us-ing a set data structure. On another hand, shufling feasibility checks are more eficient in *PAG-P*, *PAG-L*, and *PAG-P** since they employ Tarjan's algorithm. Overall, *PAG-P* and *PAG-L* runtime is stable regardless of the *T* value.

Impact of the maximum iterations (*MI*). The effect of changing the maximum number of iterations for all alternatives is shown in Figure 5(e). The maximum number of iterations in the region growing phase increases the execution time of *MP* and *MP** since iterations are handled sequentially. On the other hand, increasing the number of iterations in *PAG-P* and *PAG-P** does not significantly impact the runtime since they grow regions for each iteration in parallel. Moreover,

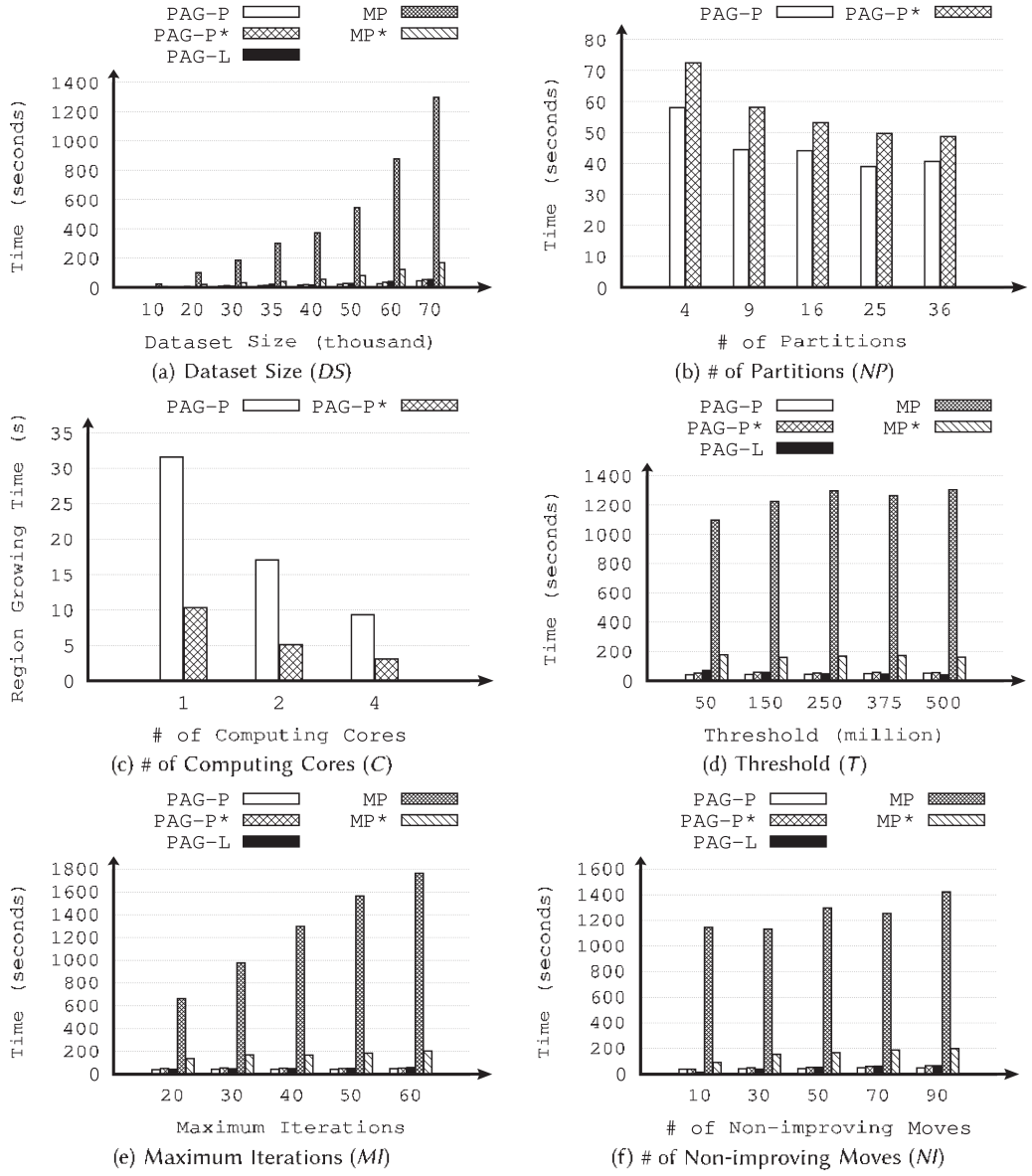


Fig. 5. Runtime scalability varying different parameters.

when the data is partitioned, the process of growing regions becomes faster since we have a smaller number of areas in each partition. For *PAG-L*, despite performing sequential iterations, its eficient runtime comes from the way it selects its next area to grow a region, which is O(1), piggybacking its cost on maintaining the list of neighbor areas.

Impact of the number of non-improving moves (*NI*). Increasing the value of *NI* increases the time for the optimization phase as shown in Figure 5(f). The *NI* parameter affects the number of moves that actually improve the solution's heterogeneity in the optimization phase. The runtime increases when the optimization phase keeps moving to a better solution and the number of

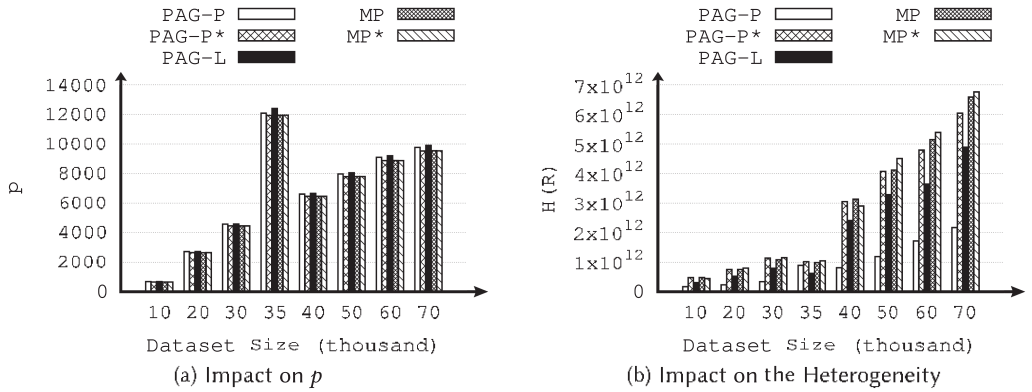


Fig. 6. Solution quality varying *DS* values.

iterations is reset to zero. Therefore, increasing the value of *NI* increases the possibility of finding an improving move, which in turn increases the runtime for all alternatives.

8.3 Solution Quality

This section evaluates the solution quality of the different alternatives in terms of the number of regions p and regions' heterogeneity.

Impact of the dataset size (*DS*). Figure 6 shows that increasing *DS* increases both p (Figure 6(a)) and the heterogeneity (Figure 6(b)) for all alternatives. Since larger datasets have more areas, using the same default threshold value leads to growing more regions, which adds to the heterogeneity score. *PAG-P* generates 27 to 243 and *PAG-L* generates 26 to 376 regions more than *MP*, *MP**, and *PAG-P**. *PAG-P* has three times lower heterogeneity than its competitors, while *PAG-L* has higher heterogeneity than *PAG-P* but significantly lower than the other competitors. The better p value of *PAG-P* and *PAG-L* comes from growing the regions with small inter-regional gaps, which leads to a smaller number of scattered enclaves and increases the number of regions. Dividing the same dataset into more regions leads to reducing the heterogeneity due to having a smaller number of area pairs that contribute to computing the heterogeneity. *PAG-P* produces solutions with less heterogeneity compared to *PAG-L* because it optimizes the heterogeneity at the region growing phase.

Impact of the number of partitions (*NP*). Figure 7 shows that changing the number of partitions has a slight impact on the solution quality. This shows the ability of our techniques to provide higher scalability while maintaining high solution quality compared to state-of-the-art techniques. Figure 7(a) and 7(b) shows that the number of regions and heterogeneity for *PAG-P* is almost the same and only decreases slightly as we add more partitions.

Impact of the user-defined constraint threshold (T). For the same dataset size, increasing T decreases the number of regions p as more areas are needed to construct each region as shown in Figure 8(a). In all cases, *MP*, *MP**, and *PAG-P** still have lower p than *PAG-P* and *PAG-L* for all T values. Larger T values means having regions with a larger number of areas, which increases the number of area pairs that contribute to heterogeneity, so the overall heterogeneity increases as shown in Figure 8(b).

Impact of the number of non-improving moves (*NI*). *NI* is a parameter for the optimization phase, so it has no impact on p as shown in Figure 9(a). For heterogeneity, Figure 9(b) shows the percentage of improvement in heterogeneity after the optimization phase. The heterogeneity improvement is computed as $\frac{H(R)-H(\bar{R})}{H(R)}\%$, where $H(R)$ and $H(\bar{R})$ are the heterogeneity before and after the optimization phase, respectively. Overall, increasing *NI* improves the heterogeneity for all

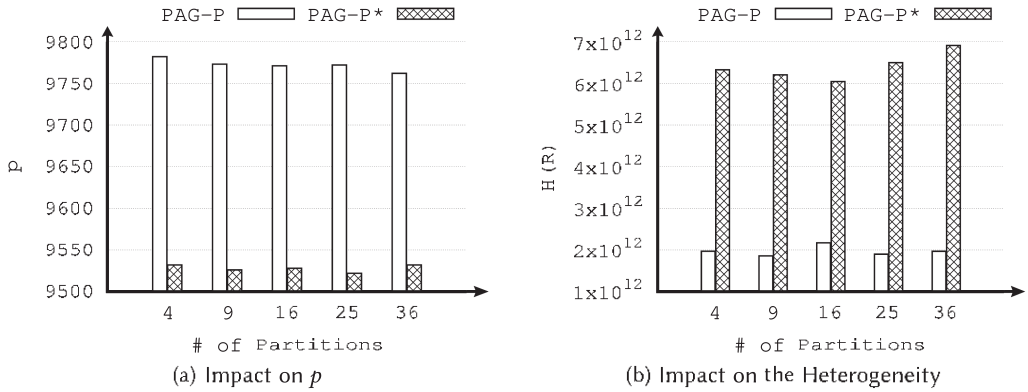


Fig. 7. Solution quality varying *NP* values.

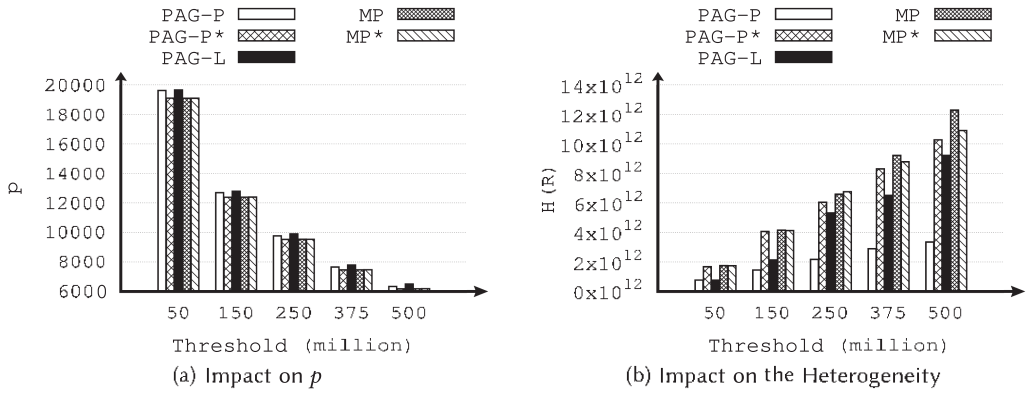


Fig. 8. Solution quality varying *T* values.

alternatives. As *NI* increases, the number of conducted moves in the optimization phase increases, which in turn increases the probability of finding a better solution. A notable observation is that *PAG-L* achieves the best heterogeneity improvement because it grows compact-shaped regions. Compact regions are more likely to maintain spatial contiguity when a boundary area is removed, resulting in more valid moves. Another observation is that *PAG-P* has the lowest heterogeneity improvement even though it has the best heterogeneity absolute value. This is due to the high-quality initial solution provided by *PAG-P*, so the optimization phase has small room for improvement.

Impact of the maximum iterations (*MI*). Figure 10 shows the p value and heterogeneity for different *MI* values. Increasing the *MI* value produces more solutions for all alternatives and therefore increases the possibility of discovering a larger value of p. In general, having a larger p value leads to having fewer pairs of areas to contribute to heterogeneity in each region, which improves heterogeneity. Figure 10(a) shows a slight increase in p in some cases. Figure 10(b) shows no apparent impact for *MI* on heterogeneity. In all cases, *PAG-P* and *PAG-L* still have the best solution quality compared to all competitors.

8.4 Evaluating Different Design Decisions

This section evaluates different design decisions regarding the placement of border areas in the partitioning phase, the seed selection in the region growing phase, adding the areas to build regions

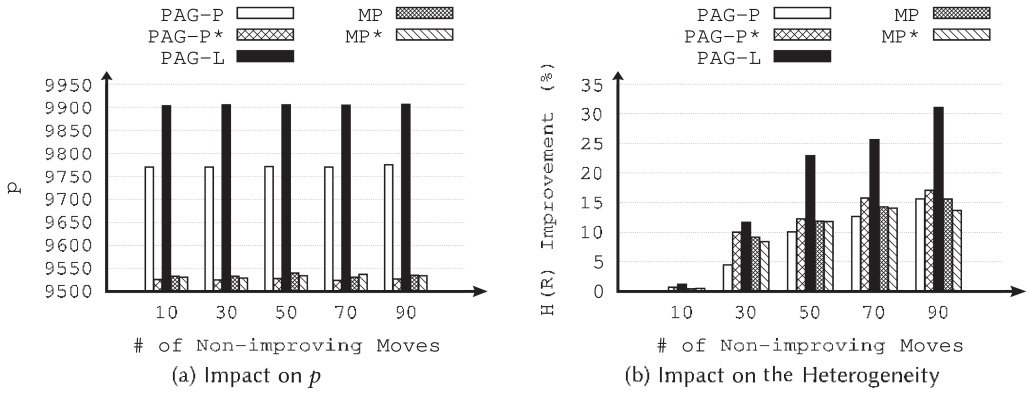


Fig. 9. Solution quality varying *NI* values.

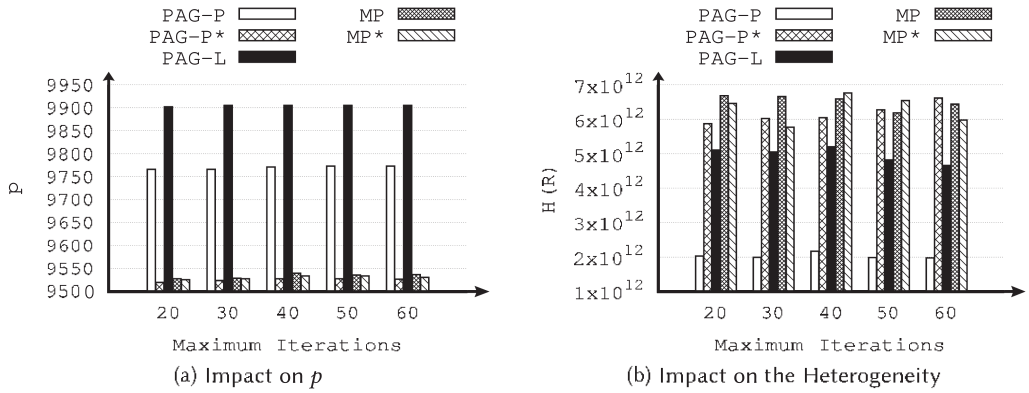


Fig. 10. Solution quality varying *MI* values.

in the region growing phase, the number of solutions kept after the region growing phase, the effect of swapping areas between regions in the optimization phase, and the effectiveness on different datasets and attributes.

Swapping the areas in the optimization phase improves the heterogeneity as it gives more room for improvement by allowing the algorithm to escape from the local minimum. The rest of the design decisions provide alternative ways, while others limit the quality or the randomization of the algorithm.

8.4.1 Swapping Areas in the Optimization Phase. *PAG-P+* here represents a variation of *PAG-P* that does not discard a movable area when its removal violates the threshold constraint of the region. Instead, it randomly picks a boundary area from the neighboring regions to swap it with and checks if the threshold and spatial contiguity constraints still hold for both regions. Figure 11(a) shows that *PAG-P+* has a better heterogeneity increase especially when the number of non-improving moves increases; however, this slows the runtime slightly as shown in Figure 11(b). *PAG-P+* provides more improvement as it widens the search space, which enables the local optimization to escape from the local minimum.

8.4.2 Placing Border Areas Based on the Heterogeneity. *PAG-P+* here represents a variation of *PAG-P* where the border areas are placed into the partition with the minimum heterogeneity

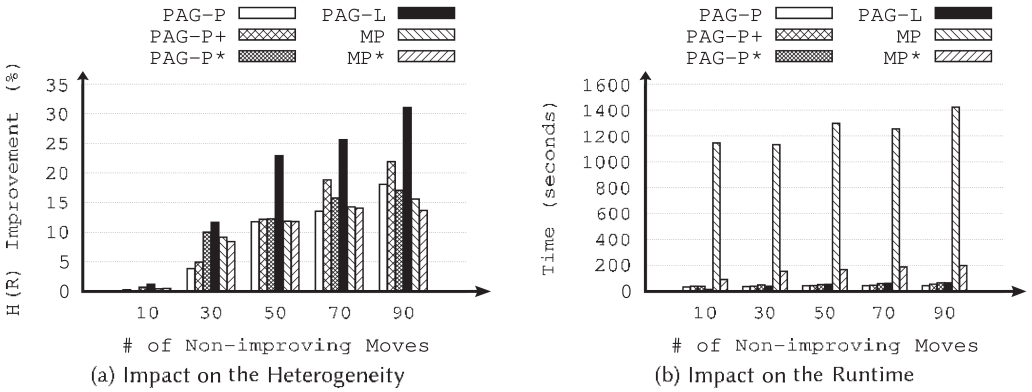


Fig. 11. Effect of swapping the areas in the optimization phase.

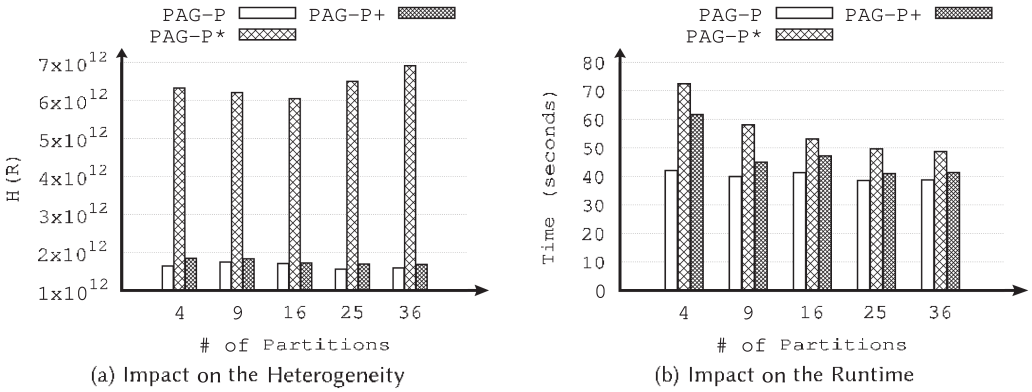


Fig. 12. Effect of placing border areas based on heterogeneity.

increase in case there are multiple neighboring partitions in the partitioning phase. Figure 12(a) shows that *PAG-P+* does not increase the heterogeneity of the solution compared to *PAG-P* with different number of partitions. This is due to the fact that the border areas are only a small part of the dataset and therefore do not contribute significantly to the heterogeneity. *PAG-P+* is slightly slower than *PAG-P* as shown in Figure 12(b) since it calculates the heterogeneity increase for every border area in order to place it into a partition.

8.4.3 Placing Border Areas Based on the Overlap. *PAG-P+* here represents a variation of *PAG-P* where border areas are placed into the partition that has the highest overlap with the area in case there are multiple neighboring partitions in the partitioning phase. Figure 13 shows that *PAG-P+* does not improve the heterogeneity or runtime. It has a slower running time since it needs to calculate the overlap between the area and the partition. It also has a higher heterogeneity since the partitions will be the same with each run, which contradicts the randomness of MP-regions and prevents exploring better solutions. Placing a border area based on the overlap will lead to including it in the same partition every time, regardless of the order of the placement. On the other hand, in *PAG-P*, placing the border area with the aim of balancing the partitions results in including it in different partitions based on the order of placing that area in different runs.

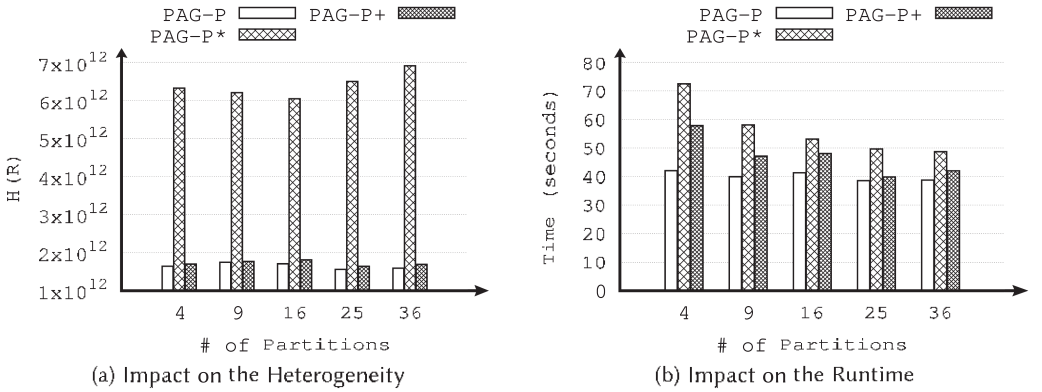


Fig. 13. Effect of placing border areas based on overlap.

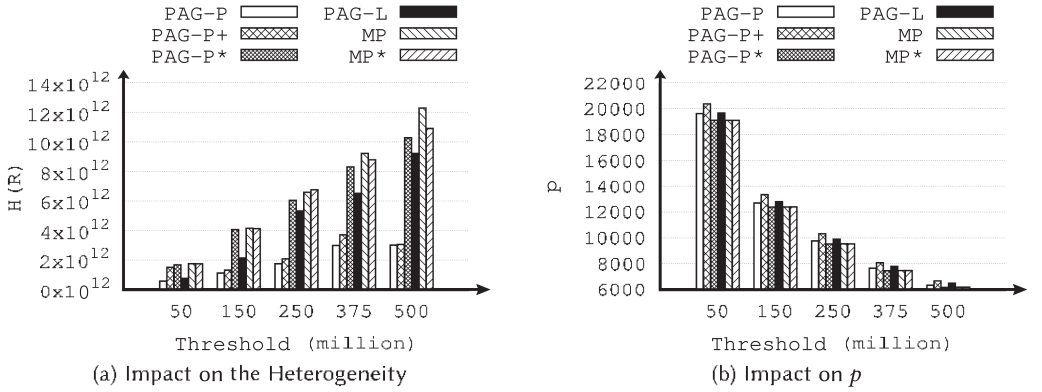


Fig. 14. Effect of selecting seeds based on the *s* value.

8.4.4 Choosing Seeds Based on the s Value. *PAG-P+* here represents a variation of *PAG-P* where the seed area with the highest *s* value is picked when growing the regions in the region growing phase. Having a seed area with a high *s* value could increase the number of regions since it will require a smaller number of areas to reach the threshold value. However, this contradicts the randomness of MP-regions and limits the search space, which prevents exploring better solutions since the solution will be the same each time. The area with the highest *s* value will always be chosen as the seed for the first region. Then, the same neighboring areas that minimize the heterogeneity of the region the most will be added to the region until it reaches the threshold. This is the same for the rest of regions, which yields the same solution. Figure 14(b) shows that *PAG-P+* generates more regions *p* than *PAG-P* (since the first regions will reach the threshold with less area) with worse heterogeneity (Figure 14(a)) for different threshold values. The heterogeneity is worse even though it has a larger number of regions than *PAG-P* since the technique is biased and not random, which leads to generating very skewed regions, i.e., regions with a very small number of areas and regions with a very large number of areas. Having regions with a very large number of areas increases the overall heterogeneity since more areas pairs will contribute to the heterogeneity.

8.4.5 Choosing Seeds Based on their Size. *PAG-P+* here represents a variation of *PAG-P* where the seed area with the largest size is picked when growing the regions in the region growing phase.

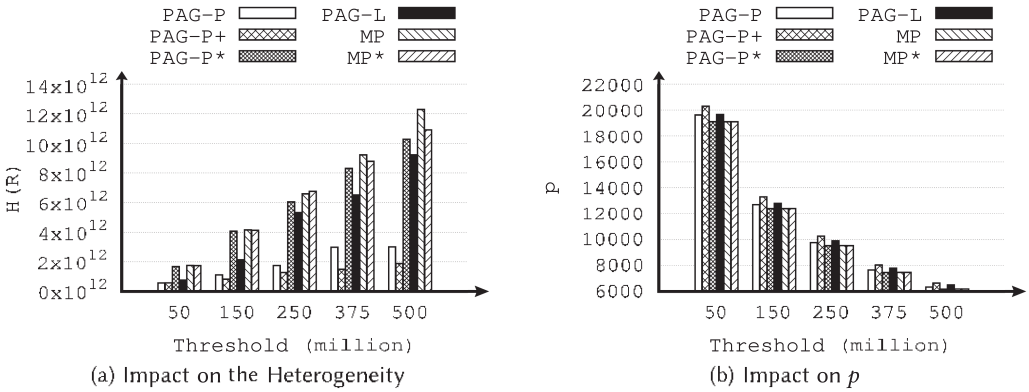


Fig. 15. Effect of selecting seeds based on their size.

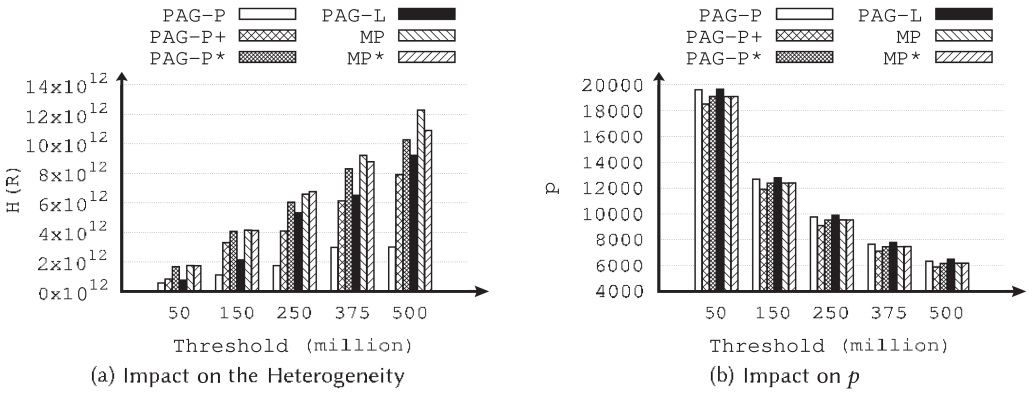
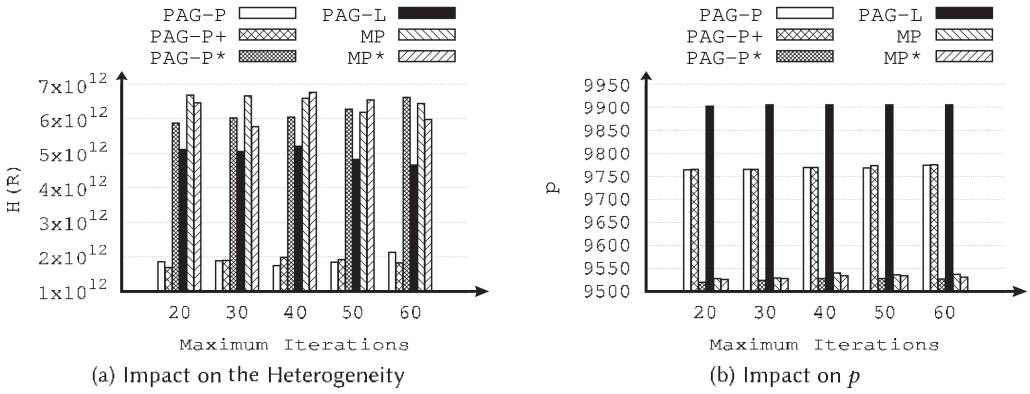


Fig. 16. Effect of growing the regions using the s value.

This is exactly the same as the previous experiment when choosing seeds based on their s value. The same solution will be generated each time, which diminishes the randomness of MP-regions and prevents exploring other solutions. Figure 15 shows that *PAG-P+* generates solutions with greater p value and better heterogeneity than *PAG-P* for different threshold values; however, this approach is invalid as it will always generate the same solution.

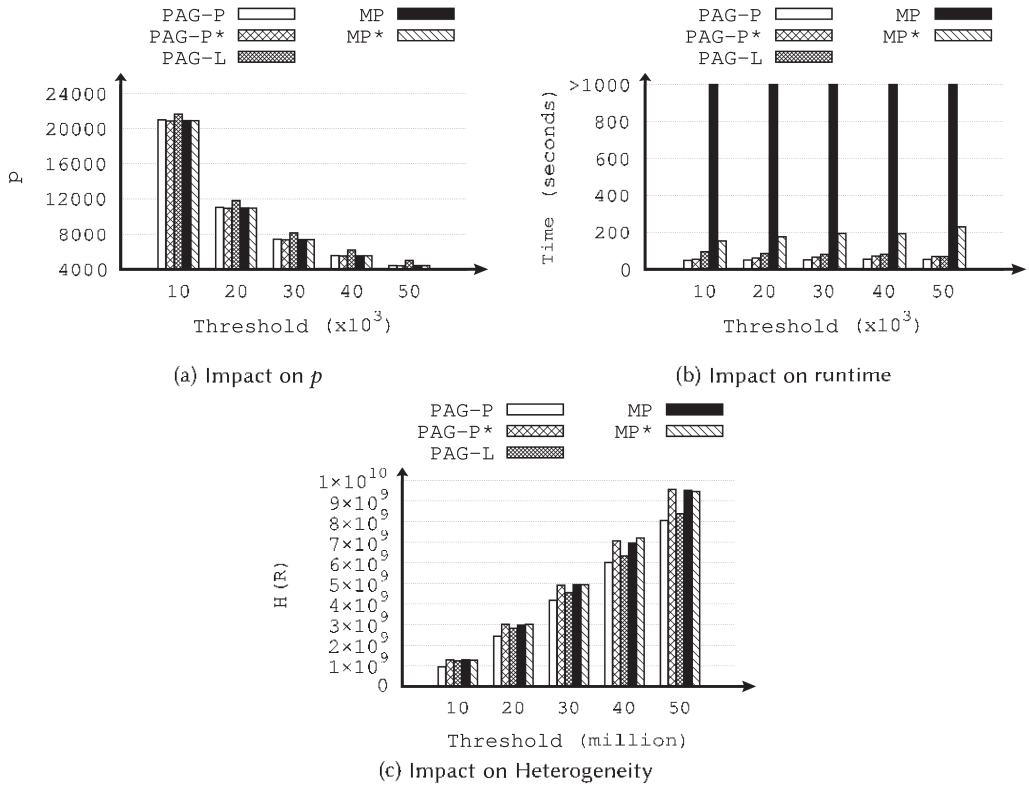
8.4.6 Growing the Regions Based on the s Value. *PAG-P+* here represents a variation of *PAG-P* that grows the regions based on the s value. It adds the area with the largest s value when growing the regions to reach the threshold value faster and increase the number of regions. *PAG-P+* generates less p value and worse heterogeneity since it does not optimize the heterogeneity as shown in Figure 16.

8.4.7 Keeping the Top k Solutions. *PAG-P+* here represents a variation of *PAG-P* that keeps the top k solutions in terms of heterogeneity after the region growing phase. The k value is set to 10 in our experiment. There is no difference between the solution quality of *PAG-P+* and *PAG-P* as shown in Figure 17. In fact, the solution quality of *PAG-P* might be better in some cases since it keeps all the solutions with the maximum number of regions. Those solutions then go through the region gluing phase and their heterogeneity might change during that phase. Finally, the solution with the minimum heterogeneity is passed to the following phase.



(a) Impact on the Heterogeneity (b) Impact on p

Fig. 17. Effect of keeping the top k solutions after the region growing phase.



(a) Impact on p (b) Impact on runtime

(c) Impact on Heterogeneity

Fig. 18. Results of running the modules with the ACS dataset.

8.4.8 Testing the Modules on Different Datasets and Different Attributes. To demonstrate generalizability, we tested our model on the American Community Survey (ACS) dataset, which has 68K areas, with the total population of the tract as the spatially extensive attribute and the average income of the tract as the dissimilarity attribute. Figure 18 shows the results on the new dataset. In general, *PAG-P* and *PAG-L* still achieve greater p and less heterogeneity compared to the baseline algorithms *MP* and *MP** under different threshold values. *PAG-L* achieves the best p value in all

cases due to the reduction of inter-regional gaps while growing the regions. *PAG-P* achieves the best heterogeneity in all cases due to having heterogeneity in consideration while growing the regions. All proposed algorithms achieve faster runtime compared to the baseline algorithms due to the optimization introduced in different phases. The good performance under the new dataset demonstrates the good generalizability of our modules.

9 CONCLUSIONS

This article addresses the scalability issue of MP-regions. MP-regions are a regionalization problem that clusters spatial areas into homogeneous regions. MP-regions are an NP-hard problem. Many of the existing works experience performance degradation when solving MP-regions on large datasets. Unfortunately, this limits its potential wide impact to support large spatial datasets. We propose the *PAGE* module, which introduces two novel variations, *PAG-P* and *PAG-L*, to pro-vide eficient and scalable approximate solutions for MP-
regions. Both techniques employ multiple phases to find an initial solution and then optimize it using heuristic search to provide a final so-lution. *PAGE* uses a high degree of parallelization and exploits eficient data structures to process large data eficiently. Our experimental evaluation has shown the superiority of our techniques against the state-of-the-art techniques in both scalability and solution quality.

REFERENCES

[1] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca Popa, Re Ramakrishnan, Raghu Christopher, Michael Stonebraker, and Dan Suciu. 2020. The Seattle report on database research. *SIGMOD Record* 48 (2020), 44–53.

[2] Hussah Alrashid, Yongyi Liu, and Amr Magdy. 2022. SMP: Scalable max-p regionalization. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, 00–00.

[3] Daniel Arribas-Bel and Charles R. Schmidt. 2013. Self-organizing maps and the US urban spatial structure. *EPB* 40 (2013), 362–371.

[4] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An eficient and robust access method for points and rectangles. In *SIGMOD Record*. 322–331.

[5] Roberto Benedetti, Federica Piersimoni, Giacomo Pignataro, and Francesco Vidoli. 2020. The identification of spatially constrained homogeneous clusters of Covid-19 transmission in Italy. *RSPP* 12 (2020), 1169–1187.

[6] Daniel Bereznyi, Ahmad Qutbuddin, YoungGu Her, and KwangSoo Yang. 2020. Node-attributed spatial graph partitioning. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, 58–67.

[7] Subhodip Biswas, Fanglan Chen, Zhiqian Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2020. Incorporating domain knowledge into memetic algorithms for solving spatial optimization problems. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, 25–35.

[8] Subhodip Biswas, Fanglan Chen, Zhiqian Chen, Andreea Sistrunk, Nathan Self, Chang-Tien Lu, and Naren Ramakrishnan. 2019. REGAL: A regionalization framework for school boundaries. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, 544–547.

[9] US Census Bureau. 2019. TIGER/Line Shapefile, 2016, Series Information for the Current Census Tract State-based Shapefile. https://catalog.data.gov/dataset/tiger-line-shapefile-2016-series-information-for-the-current-census-tract-state-based-shapefile

[10] US Census Bureau. 2021. About the American Community Survey. https://www.census.gov/programs-surveys/acs/about.html

[11] US Census Bureau. 2021. ACS Data Stories–Stats in Action! https://www.census.gov/programs-surveys/acs/about/acs-data-stories.html

[12] Zhao Chen, Peng Cheng, Lei Chen, Xuemin Lin, and Cyrus Shahabi. 2020. Fair task assignment in spatial crowdsourcing. *VLDB* 13, 12 (2020), 2479–2492.

[13] David Combe, Christine Largeron, Elöd Egyed-Zsigmond, and Mathias Géry. 2012. Combining relations and text in scientific network clustering. In *ASONAM*. IEEE Computer Society, Washington, DC, 1248–1253.

[14] Juan C. Duque, Luc Anselin, and Sergio J. Rey. 2012. The max-p-regions problem. *JRS* 52 (2012), 397–419.

[15] Juan C. Duque, Richard L. Church, and Richard S. Middleton. 2011. The p-regions problem. *Geographical Analysis* 43 (2011), 104–126.
[16] Juan C. Duque, Jorge E. Patino, Luis A. Ruiz, and Josep E. Pardo-Pascual. 2015. Measuring intra-urban poverty using land cover and texture metrics derived from remote sensing data. *LUP* 135 (2015), 11–21.
[17] Juan Carlos Duque, Raúl Ramos, and Jordi Suriñach. 2007. Supervised regionalization methods: A survey. *IRSR* 30 (2007), 195–220.
[18] Ahmed El Kenawy, Juan I. López-Moreno, and Sergio M. Vicente-Serrano. 2013. Summer temperature extremes in northeastern Spain: Spatial regionalization and links to atmospheric circulation (1960–2006). *TAC* 113 (2013), 387–405.
[19] David C. Folch and Seth E. Spielman. 2014. Identifying regions based on flexible user-defined constraints. *IJGIS* 28 (2014), 164–184.
[20] Thanaa M. Ghanem, Rahul Shah, Mohamed F. Mokbel, Walid G. Aref, and Jeffrey Scott Vitter. 2004. Bulk operations for space-partitioning trees. In *ICDE*. IEEE Computer Society, Washington, DC, 29–40.
[21] Fred Glover. 1989. Tabu search-Part I. *ORSA Journal on Computing* 1 (1989), 190–206.
[22] Diansheng Guo. 2008. Regionalization with dynamically constrained agglomerative clustering and partitioning (REDCAP). *IJGIS* 22 (2008), 801–823.
[23] Diansheng Guo and Hu Wang. 2011. Automatic region building for spatial analysis. *Transactions in GIS* 15 (2011), 29–45.
[24] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Record*. 47–57.
[25] Hua Jiang, Junfeng Kang, Zhenhong Du, Feng Zhang, Xiangzhi Huang, Renyi Liu, and Xuanting Zhang. 2018. Vector spatial big data storage and optimized query based on the multi-level Hilbert grid index in HBase. *Information* 9, 5 (2018), 116.
[26] Yunfan Kang and Amr Magdy. 2022. EMP: Max-P regionalization with enriched constraints. In *ICDE*. IEEE, 1914–1926.
[27] Hyun Kim, Yongwan Chun, and Kamyoung Kim. 2015. Delimitation of functional regions using a p-regions problem approach. *IRSR* 38 (2015), 235–263.
[28] Kamyoung Kim, Yongwan Chun, and Hyun Kim. 2017. p-Functional clusters location problem for detecting spatial clusters with covering approach. *Geographical Analysis* 49 (2017), 101–121.
[29] Jason Laura, Wenwen Li, Sergio J. Rey, and Luc Anselin. 2015. Parallelization of a regionalization heuristic in distributed computing platforms–a case study of parallel-p-compact-regions problem. *IJGIS* 29 (2015), 536–555.
[30] Wenwen Li, Richard L. Church, and Michael F. Goodchild. 2014. An extendable heuristic framework to solve the p-compact-regions problem for urban economic modeling. *CEUS* 43 (2014), 1–13.
[31] Wenwen Li, Richard L. Church, and Michael F. Goodchild. 2014. The p-compact-regions problem. *Geographical Analysis* 46 (2014), 250–273.
[32] Yongyi Liu, Ahmed R. Mahmood, Amr Magdy, and Sergio Rey. 2022. PRUC: P-regions with user-defined constraint. In *VLDB*. 491–503.
[33] Maurizio Maravalle and Bruno Simeone. 1995. A spanning tree heuristic for regional clustering. *Communications in Statistics - Theory and Methods* 24 (1995), 625–639.
[34] Stan Openshaw. 1977. Optimal zoning systems for spatial interaction models. *EPA* 9 (1977), 169–184.
[35] Kwangjin Park. 2014. Location-based grid-index for spatial query processing. *Expert Systems with Applications* 41, 4 (2014), 1294–1300.
[36] Kwangjin Park and Patrick Valduriez. 2013. A hierarchical grid index (HGI), spatial queries in wireless data broadcasting. *Distributed and Parallel Databases* 31 (2013), 413–446.
[37] Jorge E. Patino, Juan C. Duque, Josep E. Pardo-Pascual, and Luis A. Ruiz. 2014. Using remote sensing to assess the relationship between crime and the urban layout. *Applied Geography* 55 (2014), 48–60.
[38] Ate Poorthuis. 2018. How to draw a neighborhood? The potential of big data, regionalization, and community detection for understanding the heterogeneous nature of urban neighborhoods. *Geographical Analysis* 50, 2 (2018), 182–203.
[39] Tarjan R. 1971. Depth-first search and linear graph algorithms. *SICOM* 1 (1971), 114–121.
[40] Sergio J. Rey and Luc Anselin. 2010. PySAL: A Python library of spatial analytical methods. In *SAGE*. Springer, Heidelberg, 175–193.
[41] Sergio J. Rey, Luc Anselin, David C. Folch, Daniel Arribas-Bel, Myrna L. Sastré Gutiérrez, and Lindsey Interlante. 2011. Measuring spatial dynamics in metropolitan areas. *EDQ* 25 (2011), 54–64.
[42] Sergio J. Rey and Myrna L. Sastré-Gutiérrez. 2010. Interregional inequality dynamics in mexico. *SEA* 5 (2010), 277–298.
[43] Hanan Samet. 1988. An overview of quadtrees, octrees, and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and CAD* (1988), 51–68.
[44] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. 1987. *The R+-Tree: A Dynamic Index for Multi-dimensional Objects*. Technical Report.
[45] Bing She, Juan C. Duque, and Xinyue Ye. 2017. The network-max-p-regions model. *IJGIS* 31 (2017), 962–981.

[46] V. Sindhu. 2018. *Exploring Parallel Eficiency and Synergy for Max-P Region Problem Using Python*. Master's thesis. Georgia State University.
[47] Seth E. Spielman and David C. Folch. 2015. Reducing uncertainty in the american community survey through data-driven regionalization. *PloS ONE* 10 (2015), e0115626.
[48] Richard J. Trudeau. 2013. *Introduction to Graph Theory*. Courier Corporation.
[49] Dimitrios Tsitsigkos, Konstantinos Lampropoulos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2021. A two-layer partitioning for non-point spatial data. In *ICDE*. IEEE Computer Society, Washington, DC, 1787–1798.
[50] Shaohua Wang, Ershun Zhong, Hao Lu, Hui Guo, and Liang Long. 2015. An effective algorithm for lines and polygons overlay analysis using uniform spatial grid indexing. In *ICSDM*. IEEE, 175–179.
[51] Ran Wei, Sergio Rey, and Elijah Knaap. 2020. Eficient regionalization for spatially explicit neighborhood delineation. *IJGIS* 35 (2020), 1–17.
[52] Xiaofeng Xu, Li Xiong, and Vaidy Sunderam. 2016. D-grid: An in-memory dual space grid index for moving object databases. In *MDM*, Vol. 1. IEEE, 252–261.
[53] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *SIGMOD*. Association for Computing Machinery, New York, NY, 505–516.
[54] Xinyue Ye, Bing She, and Samuel Benya. 2018. Exploring regionalization in the network urban space. *JGSA* 2 (2018), 4.
[55] Baihua Zheng, Jianliang Xu, Wang-Chien Lee, and Dik Lun Lee. 2006. Grid-partition index: A hybrid method for nearest-neighbor queries in wireless location-based services. *VLDB* 15 (2006), 21–39.
[56] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *VLDB* 2 (2009), 718–729.

Received 19 October 2022; revised 23 May 2023; accepted 20 July 2023