Statistical Inference for Spatial Regionalization

Hussah Alrashid, Amr Magdy
Department of Computer Science and Engineering
University of California, Riverside
Riverside, USA
halra004@ucr.edu,amr@cs.ucr.edu

Sergio Rey
Center for Open Geographical Science
San Diego State University
San Diego, USA
srey@sdsu.edu

ABSTRACT

The process of regionalization involves clustering a set of spatial areas into spatially contiguous regions. Given the NP-hard nature of regionalization problems, all existing algorithms yield approximate solutions. To ascertain the quality of these approximations, it is crucial for domain experts to obtain statistically significant evidence on optimizing the objective function, in comparison to a random reference distribution derived from all potential sample solutions. In this paper, we propose a novel spatial regionalization problem, denoted as *SISR* (Statistical Inference for Spatial Regionalization), which generates random sample solutions with a predetermined region cardinality. The driving motivation behind *SISR* is to conduct statistical inference on any given regionalization scheme. To address *SISR*, we present a parallel technique named *PRRP* (P-Regionalization through Recursive Partitioning). *PRRP* operates over three phases: the *region growing* phase constructs initial regions with a predefined cardinality, while the *region merging* and *region splitting* phases ensure the spatial contiguity of unassigned areas, allowing for the growth of subsequent regions with predefined cardinalites. An extensive evaluation shows the effectiveness of *PRRP* using various real datasets.

CCS CONCEPTS

• **Information systems** → **Geographic information systems**.

KEYWORDS

Statistical Inference, Regionalization, Spatial Clustering

ACM Reference Format:
Hussah Alrashid, Amr Magdy and Sergio Rey. 2023. Statistical Inference for Spatial Regionalization. In *The 31st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '23), November 13–16, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589132.3625608

1 INTRODUCTION

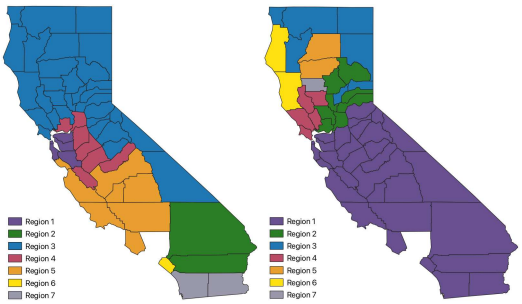
Spatial regionalization aggregates a set of spatial polygons (i.e., areas) into spatially contiguous groups of areas (i.e., regions) that satisfy one or more user-defined criteria [18]. It plays a vital role in various domains, addressing a wide range of problems such as land-use allocation [42], healthcare resources allocation [24], districting [10, 12, 29], community detection [34], and epidemic analysis [7, 14]. Since regionalization problems are NP-hard [17, 35], it is very challenging to efficiently explore the entire solution space and identify an optimal solution. For a tiny dataset of 16 areas, the mixed integer programming (MIP) formulation takes 10 hours to converge using the fast MIP solver Gurobi [25]. Even with more efficient exact algorithms, it takes more than four hours to generate a solution for tiny datasets of only 36, 49, and 64 areas [17]. Consequently, all existing techniques employ heuristics that generate approximate solutions for reasonably large datasets of thousands of areas to solve real-world critical problems. However, those approximate solutions do not provide any quality guarantees on the produced output. Therefore, domain experts need a way to perform statistical inference and assess the quality of output solutions. To conduct statistical inference, a random reference distribution of sample solutions is necessary for comparison against the assessed solution. These sample solutions within the reference distribution must be similar to the assessed solution in terms of the number of regions and each region's cardinality (i.e., the number of areas within each region) to provide meaningful statistical evidence.

Assume the Max-P regions [1, 2, 17, 25, 41] that produces the maximum number of regions satisfying a floor constraint in each region and minimizing the dissimilarity (heterogeneity) between the areas within the same region. If a geographer aims to divide California counties into regions with at least 900K households and are homogeneous in terms of the number of individuals aged 30 to 39 using an approximate Max-P regions technique in [1, 25]. The output consists of 7 regions with cardinalities: 31, 8, 7, 5, 2, 2, 1 (shown in Figure 1a). To verify if this approximate solution truly minimizes the regional dissimilarity rather than being random, we generate a reference distribution of 100 sample solutions with the same number of regions (i.e., 7) and regions' cardinalities (i.e., 31, 8, 7, 5, 2, 2, 1). Figure 1b shows one sample solution from the reference distribution. We then plot in Figure 2 the histogram of the objective function values (regions' heterogeneity) of the reference distribution compared to the assessed Max-P solution. The assessed solution's objective function is 10641×10^3, which is substantially lower than other sample solutions. This provides statistical evidence on minimizing the heterogeneity. Another statistical evidence is calculating the pseudo P-Value as follows: $\frac{(1+\sum_{i=1}^{100} solution_i <= 10641 \times 10^3)}{1+100} \approx 0.01$ [38], leading to rejecting the null hypothesis that the assessed solution is due to random chance at the 1% significance level. These pieces of statistical evidence can be also shown for the algorithms in [17, 41] to compare their quality to the ones in [1, 25] and evaluate different approximate solutions.

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 Saudi Arabian Cultural Mission SACM.

SIGSPATIAL '23, November 13–16, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0168-9/23/11.
https://doi.org/10.1145/3589132.3625608



(a) Max-P solution (b) Sample solution

Figure 1: An Example of Max-P Regions Output and a Corresponding Sample Solution

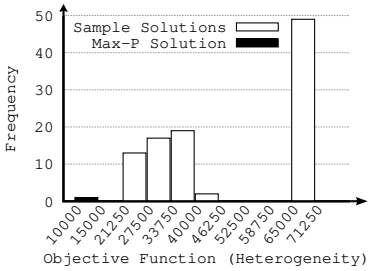
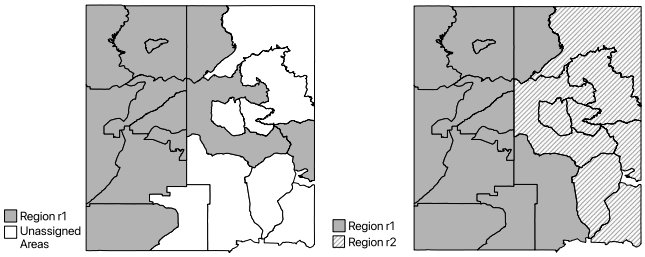


Figure 2: Histogram-based Statistical Significance Analysis for Max-P Solution Quality

The above statistical significance analysis is general and can be applied to different types of regionalization problems, such as p-regions [16, 35], compact regions [33], flexible user-constrained regions [21, 25], network-based regions [40, 43], and graph-based regions [5, 6]. However, performing this statistical analysis is impossible without generating a reference distribution of sample solutions. Existing computational techniques cannot produce such distribution as they face four main challenges. First, the high computational cost of generating multiple sample solutions poses a significant challenge. Existing regionalization techniques, such as [16, 17, 25, 35, 41], take 1.42-297 seconds, 139-184 seconds, and 256-500 seconds to generate a single solution for 400 areas, 3K areas, and 10K areas, respectively. This makes the generation of hundreds of sample solutions even more challenging. Second, the introduction of a cardinality constraint, which specifies the number of areas in each output region, is a new restrictive requirement that has not been addressed in existing regionalization problems. This constraint limits the solution's search space, making it impossible to adapt current approximate techniques to solve the problem. Existing techniques typically build regions in two phases: developing initial regions and then assigning the remaining areas to those regions. However, in our case, assigning any area to existing regions violates the cardinality constraint, rendering the region invalid.

Third, the addition of a cardinality constraint decreases the likelihood of finding feasible sample solutions. This is due to the fact that it restricts the spatial contiguity of the region being grown and future regions as illustrated in Figure 3. If the unassigned areas become spatially disconnected at any point during any region growth, it prevents the future regions (to be grown in the following iterations) from reaching their cardinality target. Empirical results demonstrate that existing techniques always fail to generate



(a) State-of-the-art techniques (b) *PRRP*

Figure 3: Growing regions with a dataset of 20 areas

a complete distribution due to the failure of generating 15-56% of regions since they do not enforce spatial contiguity. Lastly, the fourth challenge is ensuring randomness, as regions must be grown in a completely random manner for the sample solution to qualify as a random solution in a random distribution. This restriction limits the ability to control the shape of regions, e.g., compactness. This makes it more challenging to maintain the spatial contiguity of both the unassigned areas and output regions while building regions when area removal is necessary in some phases.

In this paper, we introduce a novel regionalization problem called *Statistical Inference for Spatial Regionalization* (*SISR*). *SISR* aims to generate a random sample distribution of size M, with each sample solution consisting of a set of p regions with predefined regions' cardinalities (i.e., the number of areas in each region). This random sample distribution serves as a reference distribution, drawn randomly from the set of all possible solutions sharing the same number of regions and regions' cardinalities. Consequently, it enables domain experts to perform statistical significance analysis for assessing the quality of approximate regionalization solutions for the first time in the extensive regionalization literature.

To solve *SISR*, we propose a parallel technique named *PRRP*: *P-Regionalization through Recursive Partitioning* that addresses the challenges detailed above. *PRRP* grows each region using three phases. The first phase grows one initial region with a predefined cardinality. The second and third phases perform merge and split operations to maintain the spatial contiguity of the remaining unassigned areas. The three phases are repeated for each region to recursively partition the unassigned areas into two parts, one region and one set of spatially contiguous unassigned areas, so that all unassigned areas are spatially contiguous at all times. Maintaining the spatial contiguity of unassigned areas is a major step for *PRRP* to allow subsequent regions to grow with their predefined cardinality and prevent scattered unassigned areas that cannot be grouped together to form a single region. Figure 3 emphasizes the significance of maintaining the spatial contiguity of unassigned areas and the distinction of *PRRP* from existing state-of-the-art techniques for growing two regions with cardinalities of 12 and 8. In state-of-the-art techniques, a single region is grown with 12 areas, resulting in the unassigned areas being fragmented into two separate spatially contiguous components, thereby preventing the formation of the second region (Figure3a). In contrast, *PRRP* groups the remaining unassigned areas into a single spatially contiguous component after growing the first region, thus ensuring enough space for the subsequent growth of the second region (Figure 3b).

PRRP grows regions in a completely random way by randomizing area selection for additions and removals in the first and

third phases. Such randomization increases the probability of generating different sample solutions and satisfies the randomization requirements for statistical inference. To alleviate the high cost of generating multiple solutions, *PRRP* generates different solutions in parallel, exploiting the natural independence of solutions.

The experimental evaluation of *PRRP* on real datasets demonstrates its effectiveness in solving *SISR*. *PRRP* outperforms all alternatives in different performance measures and has a 100% success probability of generating random and distinct sample solutions. *PRRP* is more efficient and is four times faster than other techniques. Our contributions are summarized as follows:

- We introduce a novel spatial regionalization problem, *SISR*, that generates random reference distributions to perform statistical inference on regionalization output.
- We introduce a novel spatial partitioning technique that significantly increases the probability of success for generating random distributions of regionalization output.
- We introduce a parallel technique *PRRP* that uses our novel partitioning to solve *SISR* effectively and efficiently.
- We conduct extensive experiments using real datasets.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 formulates the *SISR* problem. Section 4 presents our proposed technique *PRRP*. Section 5 presents the performance evaluation of *PRRP* and Section 6 concludes the paper. Appendix A presents additional experiments and Appendix B presents a complexity analysis for *PRRP*.

2 RELATED WORK

To the best of our knowledge, no current techniques set cardinality constraints on the output regions. As a result, these techniques fail to address our problem. There are two relevant domains in the literature: (a) *spatial regionalization* with different variations of constraints, and (b) *graph partitioning* techniques that are applicable to various regionalization problems. Each is briefly discussed below.

Spatial Regionalization. Existing regionalization techniques [16, 18, 21, 25–28, 30, 33, 35, 40, 41] either require the number of regions p as an input or they automatically discover the appropriate number of regions. Techniques that require p as an input include the *p-regions* problem [16] and different variations that impose a constraint on the shape of the region (i.e., compact) [30, 33] or type of the region (i.e., functional) [26–28, 35]. Other variations solve the problem in a different spatial space, such as a network space [43]. On the other hand, *Max-P regions* [17] enforces a user-defined threshold at the regional level to automatically discover the number of regions p. Several variations of *Max-P regions* [21, 25, 40, 41] have been proposed to address problems in various domains, such as the map generalization in automated cartography, which aggregates areas to produce small-scale maps [22, 23, 31, 32, 36].

Our problem requires p as an input, therefore, it is more related to the *p-regions* problem and its variations. However, none of them can be adapted to apply a different cardinality constraint on different regions since their constraints are unified for all regions, which makes the proposed solutions inherently inapplicable. In fact, adapting these techniques to solve cardinality constraints fail solve any instance of our problem (as shown in our experiments), since they do not maintain the spatial contiguity of unassigned areas while building the regions as illustrated in Figure 3.

Graph Partitioning. Several graph partitioning techniques could be adapted to generate p regions in which areas are represented as graph nodes and output regions are represented as graph partitions (i.e., sub-graphs). SKATER [5] and SKATER-CON [6] generate k sub-graphs by removing edges from spanning trees. *K-way* graph partitioning [3, 8, 39] divides a graph into k sub-graphs while minimizing the number of edges between sub-graphs. The graph bisection technique bisects a graph into two sub-graphs of roughly the same size while minimizing the number of edges connecting different sub-graphs [15, 19, 20]. Graph bisection, when applied recursively, produces p sub-graphs with a minimum number of edges between sub-graphs. Attributed graph clustering [9, 13, 44] generates k homogeneous sub-graphs based on a set of attributes associated with each node. All of the aforementioned techniques focus on generating k sub-graphs while satisfying one or more constraints. However, non of these techniques enforce a cardinality constraint on the number of nodes in each sub-graph, making them inapplicable for the same reasons above.

Standing apart from all existing research, we are the first to introduce the generation of a reference distribution of random sample solutions for statistical inference to assess the quality of approximate regionalization techniques. To achieve this, our novel problem is the first to impose a cardinality constraint that is different on each output region to empower such statistical inference.

3 PROBLEM DEFINITION

This section formally defines the *SISR* problem. We first give preliminary definitions. Then, we formally define the problem.

Definition Set 1. (Area). An ***area*** a_i is represented by two attributes (i, g), where i is the area's unique identifier and g is the area's geometry represented as an arbitrary spatial polygon.

The ***area spatial neighbors*** SN_{a_i} of area a_i are the areas that share a boundary (i.e., line or curve) with area a_i. In Figure 4a, the spatial neighbors of a_{14} are: $SN_{a_{14}} = \{a_{12}, a_{13}, a_{15}\}$. An area a_i is a ***nested area*** if its spatial polygon is completely enclosed within another area's spatial polygon. An area a_i is a ***parent area*** if its spatial polygon fully encloses the spatial polygon of another area. The ***degree*** of a parent area is the number of its nested areas. Non-nested areas are pairwise disjoint while nested areas are enclosed within larger areas' boundaries. An area a_i is an ***articulation area*** of a set of areas U if its removal breaks the spatial contiguity of the remaining areas in U. In Figure 4a, area a_4 is a nested area, area a_3 is a parent area, and area a_{19} is an articulation area for region r_1 since its removal disconnects a_{20} from the rest of r_1's areas.

Definition Set 2. (Region). A ***region*** $r = \{a_1, a_2, a_3, ..., a_m\}$ is a set of spatially contiguous areas.

Spatially contiguous areas mean that $\forall a_i, a_j \in r, \exists$ as sequence of areas $\{a_k, ...a_l\}$ such that both a_i, a_k and a_l, a_j are spatial neighbors and every two consecutive areas in the sequence are spatial neighbors.

The ***cardinality constraint*** c_i on a region r_i is a user-defined constraint on the region's size and it represents the total number of areas that must belong to r_i (i.e., $|r_i|$). A region is ***valid*** if its size equals to its predefined cardinality, i.e., $|r_i| = c_i$.

The ***region spatial neighbors*** SN_{r_i} of any region r_i are the set of areas that do not belong to r_i but have at least one spatial neighbor that belongs to r_i. The ***boundary areas*** BA_{r_i} of a region r_i are the areas in r_i that have at least one spatial neighbor that does not belong to r_i. In Figure 4a, r_3 is a region containing the following areas: $\{a_1, a_2, a_3, a_4, a_5\}$, its cardinality is 5, its spatial neighbors $SN_{r_3} = \{a_6, a_8, a_{12}, a_{13}\}$, and its boundary areas $BA_{r_3} = \{a_2, a_3, a_5\}$. A ***seed area*** is the first area added to the region (i.e., the area that a region starts growing from). An area a_i is ***assigned*** if it belongs to a region and ***unassigned*** if does not belong to any region.

Definition Set 3. (Sample Solution). A ***sample solution*** R_i is the division of any given set of areas into spatially contiguous regions: $\{r_1, r_2, r_3, ..., r_p\}$. Figure 4 shows two sample solutions R_1 in Figure 4a and R_2 in Figure 4b. A sample solution is ***feasible*** if all its regions are valid regions that satisfy the cardinality constraints.

Problem Formulation: The *SISR* problem is formally defined as follows:

Input: (1) A set of n areas: $A = \{a_1, a_2, a_3, ..., a_n\}$, in which all the areas in A are spatially contiguous. (2) An integer p which represents the number of regions, $1 \leq p \leq n$. (3) An integer M which represents the sample size. (4) A list of cardinality constraints of p integers: $C = \{c_1, c_2, c_3, ..., c_p\}$, where $\sum_{i=1}^{p} c_i = n,\ \forall c_i \in C$.

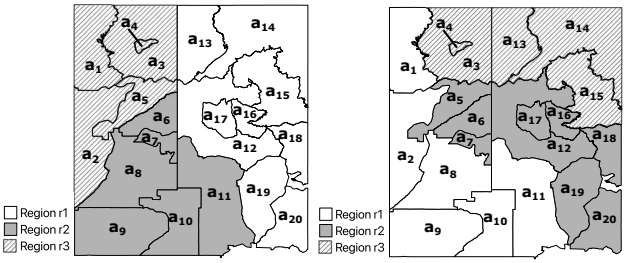
Output: A random reference distribution of M sample solutions: $S = \{R_1, R_2, ..., R_M\}$, where each sample solution R_i contains a set of p regions $R = \{r_1, r_2, ..., r_p\}$ and each region r_i is composed of a c_i spatially contiguous areas. The set of sample solutions in S must satisfy the constraints mentioned below.

Constraints:

- $|r_i| \geq 1, \qquad \forall r_i \in R$
- $r_i \cap r_j = \emptyset, \qquad \forall r_i, r_j \in R \wedge\ i \neq j$
- $\bigcup_{i=1}^{p} r_i = A, \qquad \forall r_i \in R$
- $\forall\ c_i \in C\ , \exists\ r_i \in R$ such that $|r_i| = c_i$
- $R_i \neq R_j \qquad \forall R_i, R_j \in S \wedge\ i \neq j$

All the input areas $\in A$ form a single spatially contiguous component. This increases the randomness chances of the sample solutions as having separate disjoint sets of areas will reduce the space to start and grow regions in any random area. The first three constraints ensure that each area is assigned to exactly one region and each region has at least one area. The last two constraints imply providing M distinct sample solutions that satisfy all the cardinality constraints in C. In other words, for every integer c_i in C, there must be a region r_i in R whose size (i.e., number of areas) equals c_i. This is a unique feature of *SISR* that is not addressed in any previous spatial regionalization problem.

Figure 4 presents an example of *SISR* using a dataset with 20 areas. When $M = 2$, $p = 3$, and $C = \{9, 6, 5\}$. The output is two sample solutions $S = \{R_1, R_2\}$ and each sample solution contains three regions r_1, r_2, and r_3 with cardinalites 9, 6, and 5.



(a) Sample solution R_1 **(b) Sample solution R_2**

Figure 4: An example of *SISR* with a dataset of 20 areas

4 *PRRP*: P-REGIONALIZATION THROUGH RECURSIVE PARTITIONING

This section introduces our proposed technique P-Regionalization through Recursive Partitioning (*PRRP*) to solve the *SISR* problem.

A standard way to build regions in existing regionalization techniques [16, 17, 25, 35] is to randomly select a seed area and then add neighboring areas based on the objective value until a region is formed. However, this implies that regions are grown with no regard to the spatial contiguity of the unassigned areas that do not yet belong to any region. Consequently, in most cases, there is a high possibility that unassigned areas are scattered over two or more spatially contiguous components. This is not an issue for existing regionalization problems as they do not impose strict cardinality constraints on output regions. However, for *SISR*, such scattered areas cannot be grouped in a single spatially contiguous region, preventing one or more of the future regions from reaching their cardinality target as illustrated in Figure 3. This leads to generating invalid regions with a low success probability (practically, 0% success) of finding a feasible solution in which all the regions satisfy their cardinalities.

***PRRP* main ideas**. To build valid *SISR* regions effectively, we propose the general framework of *PRRP* that is based on the recursive partitioning of n input areas A into exactly two spatially contiguous components. The first component represents one region r_i with a predefined cardinality c_i and the second component represents the remaining unassigned $(n - c_i)$ areas. When another region r_j grows, *PRRP* still maintains the spatial contiguity of the remaining unassigned areas. So, *PRRP* gives new regions ample space to grow by grouping all the unassigned areas into a single spatially contiguous component at all times as illustrated in Figure 3. This increases the success probability of finding feasible solutions from 0% for state-of-the-art techniques to 100% for *PRRP*, as shown in our experiments.

One of *PRRP* novel heuristics is growing regions in a descending order of cardinality values. This provides more flexibility for regions with larger cardinality to grow as there will be a large number of unassigned areas, reducing the needed merge and split costs in the following iterations. Moreover, larger regions have a larger set of neighboring areas, which ensures that the following region's seed is chosen randomly from a sizable set and improves the randomness chances of a sample solution.

***PRRP* framework.** The recursive partitioning in *PRRP* is achieved through three phases: *the region growing phase*, *the region merging phase*, and *the region splitting phase*. The region growing phase builds one region r_i that satisfies one of the given cardinalities $c_i \in C$. Then, the region merging phase enforces the spatial contiguity constraint on the unassigned areas by merging any disconnected components with r_i. Enforcing spatial contiguity in a separate merging phase is much more efficient than enforcing it

in the growing phase, as detailed in Section 4.1. However, it might violate the cardinality constraint of r_i. To fix this potential violation, the region splitting phase adjusts the size of r_i by removing any excess areas that might result from the region merging phase to meet the cardinality constraint again. The three phases are repeated for each region to recursively partition the remaining unassigned areas until all the regions are formed. The three phases are carefully designed to maintain the spatial contiguity of unassigned areas efficiently and find a feasible solution with high probability, as detailed below. If finding a solution fails, due to randomness, we repeat the process for a maximum number of iterations MS to increase the probability of finding a feasible solution.

***PRRP* randomness**. As *PRRP* is mainly designed to produce a random reference distribution for statistical inference, it inherently incorporates mechanisms that ensure the randomness and independence of the produced M sample solutions. First, *PRRP* constructs the M solutions in parallel, completely independent from each other. This ensures statistical independence and reduces the overall runtime. Second, for each solution, the seed area is picked randomly for each region, and the regions are grown in a completely random way by adding the areas randomly in the region growing phase. Third, the areas are added/removed to/from regions in both the region merging and region splitting phases randomly. Fourth, after producing all solutions randomly and independently, *PRRP* checks if there are duplicate solutions. The high level of randomness introduced in the different phases of *PRRP* always produces distinct solutions based on all conducted experiments.

***PRRP* Preprocessing.** *PRRP* includes two preprocessing steps. **(1) Constructing a neighborhood graph** G that captures the spatial neighborhood relationships among the input areas A. Each area $a_i \in A$ is represented by a node in G and each spatial neighbor a_j of a_i is connected to a_i by an edge in G. The neighborhood relationships between areas in G are defined by the rook neighborhood relation. The rook neighborhood relation considers two areas to be neighbors only when a common edge or curve exists between them. G is constructed by iterating over each input area and checking if it intersects with any other area by more than one point (i.e., edge or curve). **(2) Constructing a degree list** D to store only parent areas with a degree value > 0. The parent area's degree is the number of its nested areas (see Definition Set 1). To construct the degree list D, the spatial boundaries of all areas are checked against each other to check if one boundary is completely enclosed within another area. The degree list is used to check for the feasibility of moving areas and ensure the spatial contiguity of regions.

4.1 Region Growing Phase

This phase initially grows the regions to satisfy the cardinality constraint. While *PRRP* is based on maintaining spatial contiguity of unassigned areas, enforcing this while growing the region requires checking if every area is an articulation area before it is added to the region. This is prohibitively inefficient and involves traversing the entire set of unassigned areas frequently. Therefore, this phase allows growing regions without enforcing spatial contiguity, delegating this to the following phases.

Growing regions randomly in the spatial space causes the unassigned areas to break into two or more spatially contiguous components. To reduce the probability of breaking the spatial contiguity of the unassigned areas and to alleviate the computational cost of the following phases, *PRRP* employs a *gapless random seed selection* strategy when growing regions. Instead of choosing seed areas from the set of unassigned areas, only the first seed area for region r_1 is picked from the unassigned areas, and the seed area of any subsequent region ($r_i, i > 1$) is picked randomly from the spatial neighbors of r_j, $j < i$, $j = i-1, i-2, i-3, ..., 1$ in sequence. If the list of spatial neighbors $SN_{r_{i-1}}$ is empty, then the seed is picked from the neighbors of r_{i-2}, and so on. This gapless seed selection strategy takes into account the relative spatial distribution of regions and ensures that regions are grown next to each other in one part of the spatial space. The seed selection process is still completely random, but the drawing sample is limited to the neighboring areas. As regions grow in arbitrary directions, the set of neighboring areas could be any subset of areas, so the process is still completely random and not biased. If a region fails to satisfy the cardinality constraint, the algorithm restarts the building process with a seed that is picked from all unassigned areas.

As discussed before, the regions are grown in descending order of cardinalities $c_i \in C$. The region growing phase starts by initializing a set of unassigned areas $A^u = A$ and a set of assigned areas $A^a = \phi$. Then, the largest region r_1 grows targeting c_1 areas. A seed area s is randomly selected from A^u, added to region r_1, and marked as assigned by adding it to A^a. A list of unassigned spatial neighbors of r_1, SN_{r_1}, is initialized with all unassigned spatial neighbors of the seed area s, i.e., $SN_{r_1} = SN_s$. SN_{r_1} is maintained while growing r_1 to ensure the disjointness of regions and the spatial contiguity in the following iterations of growing $r_i, i > 1$. To grow region r_1, an area a_i is selected at random from SN_{r_1} and is added to the region. The list SN_{r_1} is updated accordingly by removing a_i and adding the unassigned neighbors from SN_{a_i}. We keep adding areas to r_1 from SN_{r_1} until r_1 cardinality reaches the target cardinality c_1.

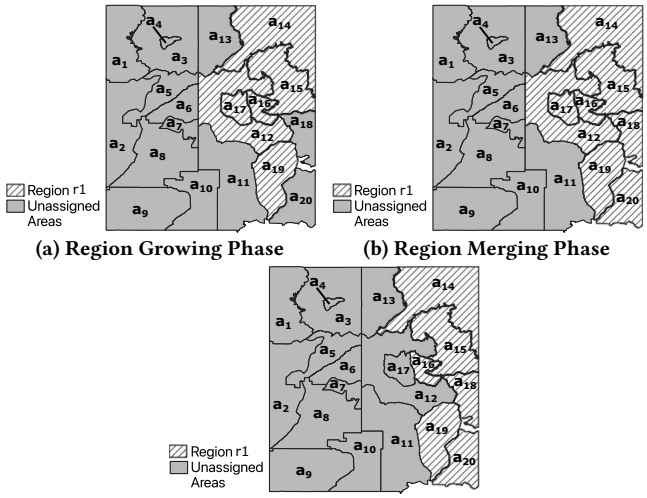
After r_1 becomes a valid region, a queue for the seed areas *seeds* is initialized with all the unassigned spatial neighbors of r_1. Seed areas in the following iterations are drawn randomly from *seeds*. *seeds* is updated in the following region growing iterations for $r_i, i > 1$ by removing the areas that have been assigned to r_i and adding the unassigned spatial neighbors from SN_{r_i}.

If a region fails to reach its cardinality value for any reason, then the region growing phase is restarted as long as the number of restarts does not exceed a parameter MR, representing the maximum allowed number of growing a region. In this case, the seed area is not picked from *seeds*. Instead, it is picked randomly from A^u to increase the probability of growing a valid region successfully.

Example. Figure 5 presents an example of the three phases of building a region in *PRRP*. Figure 5a shows the result of growing a region with cardinality = 6 after the region growing phase. This region splits the unassigned areas A^u into three spatially contiguous components: $\{a_{18}\}$, $\{a_{20}\}$, and $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{17}\}$.

4.2 Region Merging Phase

This phase is the first step in ensuring the spatial contiguity of the remaining unassigned areas A^u after growing a region r_i. When r_i is grown, the remaining unassigned areas A^u could break into two or more spatially contiguous components. It is important that the areas in A^u stay spatially contiguous at all times to allow the remaining



(a) Region Growing Phase (b) Region Merging Phase

(c) Region Splitting Phase

Figure 5: The phases of constructing a region in *PRRP*

regions to grow correctly and become valid. When the areas in A^u become disconnected, spatially-scattered areas cannot form a spatially contiguous region, and subsequent regions cannot reach their predefined cardinality targets and fail to produce any correct solution. For instance, in Figure 3a after growing region r_1, A^u contains 8 areas and the last region to be formed have a cardinality value of 8. However, growing r_1 has disconnected A^u into two spatially contiguous components each with 4 areas preventing the next region from growing with cardinality 8.

To prevent this from happening, the spatially contiguous components for the unassigned areas A^u are calculated after the region growing phase. If the number of the spatially contiguous components is one, then both the region merging and region splitting phases are skipped, and the algorithm moves to the next region growing phase. If there is more than one spatially contiguous component, then only the largest component is kept as unassigned and the rest of them are removed from A^u and added to region r_i and the set of the assigned areas A^a. In this case, more areas are added to r_i, and it exceeds its cardinality constraint. Therefore, r_i must go through the region splitting phase to remove the excess areas to bring it back to its predefined cardinality value.

Example. Figure 5b depicts the region merging phase merging the small components, in terms of the number of areas, $\{a_{18}\}$ and $\{a_{20}\}$ with the region to produce a region r_1 with a total of 8 areas.

4.3 Region Splitting Phase

This phase is required when the region r_i is merged in the region merging phase to reduce its size and bring it back to its cardinality target c_i. To this end, it removes areas from the region r_i and adds them back to the list of unassigned areas A^u.

Removing areas from r_i focuses on removing a subset of boundary areas of r_i. A list of boundary areas BA_{r_i} is computed as the areas in r_i that have spatial neighbors belonging to A^u. A naive way to process BA_{r_i} is computing and excluding articulation areas, removing one of the boundary areas at a time, and recomputing boundary and articulation areas for each boundary area to be removed. Such a naive way to process BA_{r_i} has several limitations. First, recomputing boundary and articulations areas with every area removal is prohibitively expensive since it requires traversing all the areas in the region. Second, removing only boundary areas that are not articulation areas is restrictive and could prevent the removal of excess areas when all the boundary areas are also articulation areas. To overcome these limitations, we employ a randomized technique that permits the removal of any area from BA_{r_i} then checks if the spatial contiguity constraint still holds for r_i and makes the necessary adjustments. Removing a boundary area randomly has several advantages: (1) it provides more flexibility while removing areas when all the areas in BA_{r_i} are articulation areas, (2) it increases the randomness of the sample solution by allowing the removal of articulation areas, (3) it alleviates the cost of recomputing the boundary areas' list; instead the same list is used until it becomes empty. This does not affect the randomness of the solution as the areas are still chosen randomly from the list.

Another issue with removing areas is that a parent area a_t is always an articulation area. Since one or more areas are enclosed within $a_t \in r_i$, some or all of its nested areas could also belong to r_i. So, removing a_t breaks the spatial contiguity of r_i. This prevents a_t from ever being removed from r_i as it is always identified as an articulation area. To solve this, we construct and use the degree list D during preprocessing to identify parent and nested areas. So when a boundary area that is also a parent area is removed, all the of its nested areas are also removed.

Splitting the excess areas from the region is performed over two steps: the ***region shrinking*** and the ***region expansion***. The region shrinking step removes areas randomly from the region and the region expansion step adds areas to the region if the region size falls under the cardinality target after the region shrinking step.

Region shrinking. Algorithm 1 shows the pseudo-code of the region shrinking step. The algorithm starts by initializing a counter k that is equal to the number of areas to be removed from region r_i where $k = |r_i| - c_i$ (Line 5). Then a list of boundary areas BA_{r_i} for region r_i is computed (Line 8). An area a_i is randomly selected from BA_{r_i} and checked against the degree list D. If D contains a_i, then a_i and all of its nested areas that belong to r_i are removed from r_i and added to A^u. Then, the counter k is updated by subtracting the total number of removed areas and BA_{r_i} is updated by removing all the removed areas (Lines 9-15). The spatial contiguity of region r_i is checked after the removal of area a_i by calculating the spatially contiguous components of r_i. If there is more than one component, then only the large component is kept and the other components are removed from r_i and added to A^u. The number of areas in the components removed from r_i is subtracted from k (Lines 16-20). To speed up the removal of areas, the list of boundary areas BA_{r_i} is only computed when it becomes empty. The process of removing areas continues until the counter $k <= 0$. If k is zero, then region r_i becomes valid and the algorithm moves to building the next region. Otherwise, if $k < 0$ and the region is not empty after the shrinking step, then the region is adjusted in the region expansion step by removing areas from A^u and adding them to the region to meet the cardinality constraint. If $k < 0$ and the region becomes empty after the region shrinking step, then the region splitting phase terminates and the region is re-grown in a new iteration. The region could become empty when, for instance, the region cardinality is one and it currently contains two areas: one parent area and its nested area. In this case, only one area should be removed. However, since

Algorithm 1: Region Shrinking Step

Input : *Area list A^u, Region r, Integer c, Neighborhood list G, Degree list D.*
Output: *Region r.*
Initialization:
BA_r = {} // *list of boundary areas for r*;
k = |r| - c;
while $k > 0$ **do**
  if |BA_r| == 0 **then**
    BA_r = *boundary areas for r*;
  a = *random area* ∈ BA_r;
  if $a \in D$ **then**
    NA = *nested areas for a that belong to r*;
  r = (r - a) - NA;
  A^u = ($A^u \cup a$) ∪ NA;
  BA_r = (BA_r - a) - NA;
  k = k - (|NA| + 1);
  cc = *connected components for r*;
  if |cc| > 1 **then**
    r = r - *smallest connected components*;
    A^u = A^u ∪ *smallest connected components*;
    k = k - |*smallest connected components*|;
return r;

the shrinking step removes the parent area and its nested areas to maintain the spatial contiguity of the areas, then the two areas are removed and the region becomes empty.

Region expansion. The process of removing areas from A^u is very similar to the process of removing areas from the region. However, the calculation of the articulation areas every time an area is removed is necessary here since the removal of an exact number of areas from A^u is required to maintain the region's cardinality and to prevent the region from exceeding the predefined cardinality target again. A counter w which represents the number of areas to be added to the region is initialized as follows: $w = c_i - |r_i|$. After that, a list of boundary areas for the unassigned areas A^u is calculated BA_{A^u}. Then, the articulation areas of A^u are computed without considering the nested areas of the parent areas in BA_{A^u}. This ensures that parent areas are only identified as articulation areas when they actually connect other areas to the region beside their nested areas. The articulation areas of A^u are computed using Trajan's algorithm [37] and subtracted from BA_{A^u} to maintain the spatial contiguity of A^u. Next, we pick an area a_i from BA_{A^u}, if D contains a_i, then a_i and a_i's unassigned nested areas are removed from A^u and added to region r_i. The counter w is updated by subtracting the number of areas that are added to r_i. This process continues until the value of w reaches zero.

If the value of w falls under zero after the region splitting phase, then the cardinality of r_i is checked against the values in C. If $|r_i| \in C$ then the region is kept. Otherwise: r_i is removed, A^u is reset to its previous state before growing r_i, and a new building iteration begins. The value of w can fall under zero if, for example, $w = 1$ and all the areas in BA_{A^u} are parent areas containing at least one unassigned nested area. In this case, all the unassigned nested areas would have to be removed with the area and w will fall under zero.

Example. In Figure 5c, the region splitting phase removes 2 areas from the region (a_{12} and a_{17}) to satisfy the cardinality.

At the end, our algorithm checks if there are duplicate sample solutions by checking if the sample solutions have the same regions, i.e., the same cardinality and set of areas. Therefore, the sample solutions in the reference distribution are guaranteed to be distinct.

5 EXPERIMENTAL EVALUATION

This section provides an extensive performance evaluation for our proposed technique *PRRP*. Section 5.1 presents the experimental setup and Section 5.2 evaluates the performance of *PRRP*.

5.1 Experimental Setup

Our evaluation is based on Java 14 implementation and evaluated on a machine running Ubuntu 16.04 with a quad-core 3.5GHz processor and 128GB of memory. We compare *PRRP* with the state-of-the-art *p-regions* technique, *PRUC* [35], in addition to two baseline variations that evaluate *PRRP* design decisions: (1) A variation of *PRUC* [35], denoted as ***PRUC-RP***. The seed selection in *PRUC* identifies p scattered seeds. Then, it grows regions from those seeds by adding the area that has more neighbors in the region. As *SISR* is a novel problem, it is the first to impose a different cardinality on each region. Therefore, the cardinalites are set as threshold values for each region in *PRUC-RP* and the extensive attribute of each area is set to 1. (2) A modified version of *PRRP*, denoted as ***PRRP-G***, that adopts only the region growing phase of *PRRP* to show the importance of continuously maintaining the spatial contiguity in the other two phases. (3) A sequential version of *PRRP*, denoted as ***PRRP-S***, to show the effect of parallelization on the runtime.

Evaluation Datasets. We evaluate the performance of all techniques using 3 datasets: (1) the census tracts of the US states (denoted as *CT*) [11], (2) the US county subdivisions (denoted as *CS*) [11], and (3) the health, income diversity of the US counties (denoted as *HI*) [4]. The *CT* and *CS* datasets have three subsets of sizes ≈2K, 5K, and 10K denoted as *2KCT*, *5KCT*, and *10KCT* for the *CT* and *2KCS*, *5KCS*, and *10KCS* for the *CS*. The *2KCT* represents the census tracts for NY city, the *5KCT* represents the census tracts for NY state, and the *10KCT* represents the census tracts for CA, AZ, and NV states. The *2KCS* represents the county subdivisions for CA, NV, AZ, UT, ID, OR, WA, MT, WY, CO, and NM states. The *5KCS* is the county subdivisions of all the states in the *2KCS* in addition to TX, OK, and KS states. The *10KCS* is the county subdivisions for all the states in the *5KCS* plus NE, SD, ND, LA, and AR states. The *HI* dataset has ≈3K areas denoted as *3KHI* and represents the health, income, diversity of the US counties. The data available for US counties and states form datasets that are large enough to appropriately evaluate and stress our algorithm.

Evaluation Measures. We evaluate the performance of *PRRP* and all alternatives against the following measures: ***(1) Execution time***, to measure scalability and runtime efficiency. ***(2) Success probability***, to measure the probability of successfully producing feasible sample solutions. It is calculated as $\frac{|S|}{M}$, where $|S|$ is the number of output feasible sample solutions. It takes a range from zero to 1 where zero indicates that the algorithm failed to generate any feasible sample solution and 1 indicates that the algorithm successfully produced M feasible sample solutions, so the larger,

Table 1: Evaluation parameters

Parameter	Values
DS	≈ 2, 3, **5**, 10 ($\times 10^3$)
p (% of DS)	1%, **2%**, 3%
M	10, **50**, 100
MR	10, **30**, 50
Q	1, 2, **4**

the better. ***(3) Effectiveness***, to measure the effectiveness of producing a feasible sample solution in early iterations. It is measured as $\frac{|S|}{\sum_{\forall R_i \in S} ms_i}$, where $|S|$ is the number of output feasible sample solutions and ms_i is the number of executed *MS* iterations until a feasible sample solution is produced. It takes a range from zero to 1 where zero indicates that the algorithm failed to generate any feasible sample solution after going through all the iterations and 1 indicates that the algorithm successfully produced all the sample solutions in a single iteration. Values between 0 and 1 indicate that generating each solution in average takes multiple iterations, so the larger effectiveness, the better. ***(4) Completeness***, this measure distinguishes solutions from each other, even failed ones, depending on the number of valid regions built before the solution terminates either successfully or unsuccessfully. This is measured through counting how many valid regions were successfully generated out of the p required regions. So, failed solutions that generate only one valid region are considered much worse than failed solutions that generate $(p-1)$ regions. It is measured as $\frac{\sum_{\forall R_i \in S} \frac{|R_i|}{p}}{M}$ where $|R_i|$ is the number of valid regions. It takes a range from zero to 1 where zero indicates that the algorithm failed to generate any valid region for any sample solution and 1 indicates that all the regions are valid for all solutions, so the larger, the better.

Parameters. We study the effect of the parameters listed below on the performance. Table 1 shows the values for all parameters where the default values are emphasized in boldface.

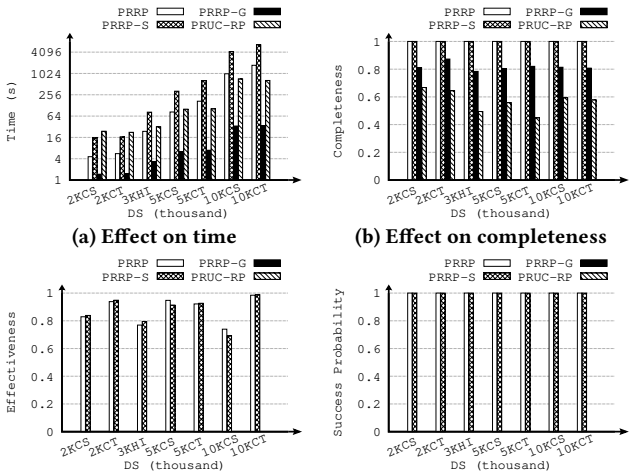
- *DS*: the dataset size (i.e., number of areas in the dataset).
- *p*: the number of regions in each sample solution.
- *M*: the sample size.
- *MR*: the maximum iterations to build a valid region.
- *Q*: the number of computing cores.

The maximum number of iterations for finding a feasible sample solution (*MS*) is empirically set to 10 since there is a high probability of finding a feasible solutions in early iterations. The cardinality list (C) is generated randomly, which emulates random cardinalities generated from the majority of regionalization techniques.

5.2 Performance Evaluation

This section studies the performance of *PRRP* and all other alternatives under different parameter values.

Impact of the dataset size (DS). Increasing the *DS* increases the runtime for all alternatives as shown in Figure 6a. *PRRP* is 4 times faster than *PRRP-S* for the largest dataset due to parallelization with 4 cores. *PRRP* has a smaller runtime ranging from 4 to 1737, 4 to 1001, and 23 seconds than *PRRP-S* which ranges from 16 to 6589, 15 to 4218, 82 seconds for the *CT*, *CS*, *HI* datasets, respectively. The runtime breakdown shows 0.2-3% of the runtime is for the region growing phase, 5-13% of the runtime is for the region merging phase,



(a) Effect on time (b) Effect on completeness

(c) Effect on effectiveness (d) Effect on success probability

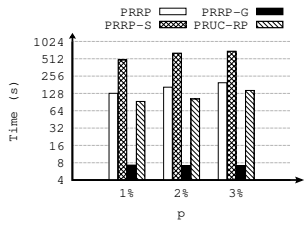
Figure 6: Impact of the Dataset Size under *CT, CS, HI* datasets

and the region splitting phase is 74-93% of the runtime. The region splitting phase dominates the runtime as it involves traversing the areas to identify articulation areas. The success probability and completeness of *PRRP* and *PRRP-S* is always 1 for all dataset sizes (Figure 6b, Figure 6d). Similarly, *PRRP* and *PRRP-S* have a high effectiveness as shown in Figure 6c.

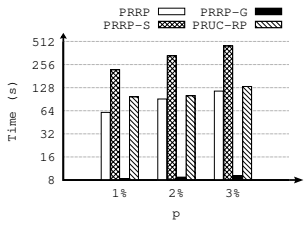
On the contrary to *PRRP* and *PRRP-S*, both *PRRP-G* and *PRUC-RP* have a zero success probability and effectiveness for all dataset sizes, depicted in Figure 6c and Figure 6d. Both alternatives fail to produce any feasible sample solutions for all datasets. Although they are both faster than *PRRP* and *PRRP-S* for all datasets, they terminate early without producing feasible sample solutions. This is due to the fact that both *PRRP-G* and *PRUC-RP* do not maintain the spatial contiguity of unassigned areas causing areas to get disconnected which prevents the regions from reaching their cardinality targets. Regarding completeness, *PRRP-G* have 21%-82% higher completeness value compared to *PRUC-RP*. This proves that the seed selection and region growing strategies of *PRRP* helps in producing valid regions. The seed selection strategy in *PRUC-RP* selects all the seed areas in advance and then grows regions sequentially from those seeds. Using this technique, later seed areas get completely blocked by assigned areas when growing regions which prevents those seeds from growing their own regions.

The performance difference of *PRRP* and *PRRP-S* under the *CT* and *CS* datasets varies. The time for the *CT* dataset for both *PRRP* and *PRRP-S* is 1-2 times slower than that of the *CS* dataset because the *CT* dataset has more nested areas and they have to be removed from the region to compute the boundary areas in the region splitting phase which increases the overall time. On the other hand, the effectiveness of *PRRP* and *PRRP-S* using the *10KCS* dataset is 33-43% less than the effectiveness using the *10KCT* dataset (0.69 and 0.73 vs. 0.98 and 0.98). This is due to the fact that the areas in the *10KCS* dataset are larger and span more space which increases the number of articulation areas and makes regions more prone to get spatially disconnected in the region growing and splitting phases. As a result, more iterations are performed to produce feasible sample solutions.

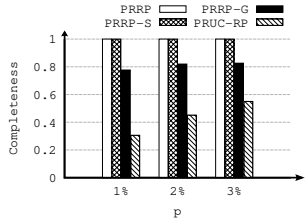
Impact of the number of regions (p). The number of regions p affects the runtime of *PRRP*, *PRRP-S*, and *PRUC-RP* for both *CT* and



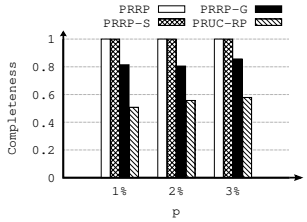
(a) Time under the *CT* dataset



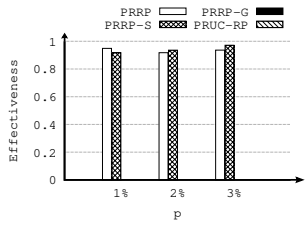
(b) Time under the *CS* dataset



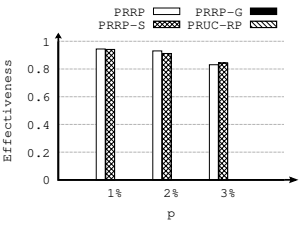
(c) Completeness under the *CT* dataset



(d) Completeness under the *CS* dataset

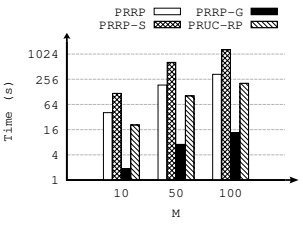


(e) Effectiveness under the *CT* dataset

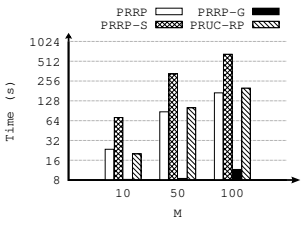


(f) Effectiveness under the *CS* dataset

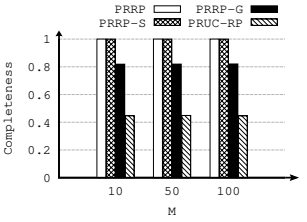
Figure 7: Impact of the number of regions (p)



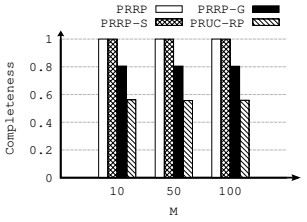
(a) Time under the *CT* dataset



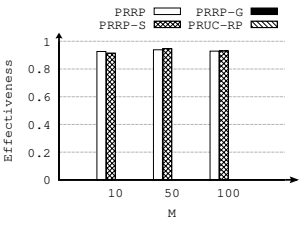
(b) Time under the *CS* dataset



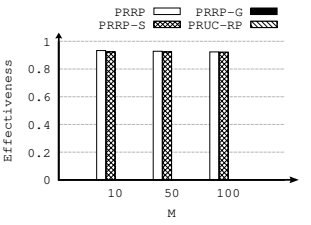
(c) Completeness under the *CT* dataset



(d) Completeness under the *CS* dataset



(e) Effectiveness under the *CT* dataset



(f) Effectiveness under the *CS* dataset

Figure 8: Impact of the sample size (M)

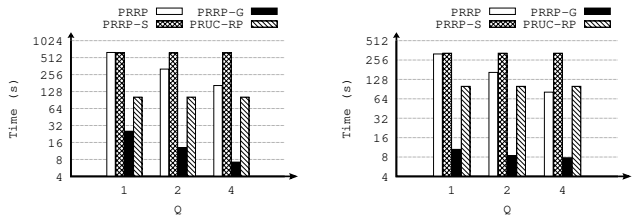
CS as shown in Figure 7a and Figure 7b. The runtime for those alternatives slightly increases as the p value increases (from 128 to 196 seconds for *PRRP* with a 53% increase, from 490 to 693 seconds for *PRRP-S* with a 41% increase, and from 92 to 144 seconds for *PRUC-RP* with a 56% increase). In *PRRP* and *PRRP-S*, increasing the p value increases the merge and split operations which are the most expensive phases of the algorithm. In *PRUC-RP*, the area that is most connected to the region is chosen, therefore, the time increases as there are more regions. In *PRRP-G*, the areas are randomly picked which does not have a significant effect on the runtime.

In several cases, *PRRP-G* and *PRUC-RP* are faster than *PRRP* and *PRRP-S*, however, they fail to produce feasible sample solutions as shown in Figure 11a in Appendix A, Figure 7e, Figure 11b in Appendix A, and Figure 7f. On the contrary, *PRRP* and *PRRP-S* always have a very high probability of success, completeness, and effectiveness for different p values because they continuously maintain the spatial contiguity of unassigned areas.

Relatively, the time for *PRRP* and *PRRP-S* on the *CT* dataset is slower than the time on the *CS* dataset (35%-55% increase). As mentioned before, this is due to nature of the datasets as the *CT* dataset has more nested areas which increases the merge and split operations. Also, the effectiveness of *PRRP* (0.97) and *PRRP-S* (0.97) is 14% higher for the *10KCT* dataset than the effectiveness of the *10KCS* dataset (0.83 for *PRRP* and 0.84 for *PRRP-S*). As mentioned before, the areas in the *10KCS* dataset are larger which increases the number of articulation areas causing the areas to get spatially disconnected which requires more attempts to build the region.

Impact of the sample size (M). As the value of M doubles, more sample solutions need to be generated which increases the time of all alternatives linearly as shown in Figure 8a and Figure 8b. *PRRP* and *PRRP-S* generate valid regions for both the *CT* and *CS* datasets which in turn produce feasible sample solutions with a success probability and completeness of 1 for all values of M (Figure 12a in Appendix A, Figure 8c, Figure 12b in Appendix A, and Figure 8d). As for the effectiveness, Figure 8e shows that the sample size M does not have a noticeable impact on *PRRP* (0.92-0.95 for the *CT* dataset and 0.92-0.93 for the *CS* dataset) and *PRRP-S* (0.91-0.94 for the *CT* dataset and 0.92 for the *CS* dataset), which is very high. *PRRP-G* and *PRUC-RP* both fail to generate feasible sample solutions under different M values with a zero success probability and effectiveness. Similar conclusions are drawn for completeness as discussed in the previous experiments. *PRRP* has a smaller run time than *PRRP-S* (3-4 times faster) because it produces multiple sample solutions in parallel but slower than *PRRP-G* and *PRUC-RP* as they terminate early without producing any feasible sample solution.

Impact of the number of computing cores (Q). The number of computing cores Q only affects the runtime of *PRRP* and *PRRP-G* as they are the only parallelized techniques. The runtime for *PRRP-S* and *PRUC-RP* in Figure 9 for all Q values is the same and with the default values for all parameters. Figure 9 depicts the runtime in a log scale and shows a linear scalability for the runtime of *PRRP* and *PRRP-G* when increasing Q. *PRRP* takes 641 seconds with one core, 291 seconds with two cores, and 179 seconds with four cores and *PRRP-G* takes 25 seconds with one core, 12 seconds with two cores, and 7 seconds with four cores for the *CT* dataset as shown



(a) *CT* dataset **(b) *CS* dataset**

Figure 9: Impact of the number of computing cores (Q)

in Figure 9a. Figure 9b shows similar scalability for the *CS* dataset. This shows that the parallel technique in *PRRP* makes full use of the computing resources which leads to perfect (linear) speedup.

Impact of the maximum number of iterations (MR). Figure 10 in Appendix A shows that increasing the *MR* value has a slight impact on the runtime of *PRRP*, *PRRP-S*, and *PRRP-G*. *PRRP* and *PRRP-S* still have a high success probability, completeness, and effectiveness. More details are provided in Appendix A.

Impact of the seed selection technique. In addition to our gapless random seed selection technique, we employ two techniques for *PRRP* and its variations *PRRP-S* and *PRRP-G*: (1) a random technique that picks the seed area randomly from the unassigned areas, and (2) a modified gapless random technique that picks the seed from the spatial neighbors of all regions. Figure 13 in Appendix A shows that this random seed selection technique is 1.3 times slower than our technique with less effectiveness. On the other hand, Figure 14 in Appendix A shows that the modified gapless random technique has no apparent effect as the results are comparable to those produced by our technique. More details are provided in Appendix A.

6 CONCLUSIONS

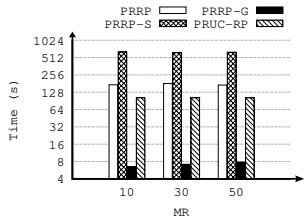
This paper introduces *SISR*, a regionalization problem that partitions a set of spatial areas into p regions, each with a predefined cardinality constraint. *SISR* enables spatial statisticians to generate reference distributions to perform statistical inference and assess the quality of regionalization solutions. We propose a parallel technique *PRRP* to solve *SISR*. *PRRP* works by recursively partitioning the unassigned areas into two spatially contiguous components over three phases: (1) region growing phase that constructs a single region with a predefined cardinality, (2) region merging phase that merges any disconnected areas with the grown region, (3) region splitting phase that adjusts the region size after the region merging phase by removing excess areas. Experimental results using real datasets show the high effectiveness of *PRRP* and the invalidity of the state-of-the-art techniques to produce any valid solutions.

REFERENCES

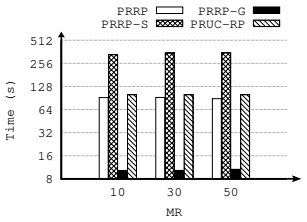
[1] H. Alrashid, Y. Liu, and A. Magdy. SMP: Scalable Max-P Regionalization. In *SIGSPATIAL*, pages 1–4, 2022.
[2] H. Alrashid, Y. Liu, and A. Magdy. PAGE: Parallel Scalable Regionalization Framework. *TSAS*, pages 1–27, 2023.
[3] K. Andreev and H. Racke. Balanced Graph Partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
[4] l. Anselin. GeoDa, 2003. https://geodacenter.github.io/.
[5] R. Assunção and et. al. Efficient Regionalization Techniques for Socio-economic Geographical Units Using Minimum Spanning Trees. *IJGIS*, 20:797–811, 2006.
[6] O. Aydin, M. V. Janikas, R. Assunção, and T.-H. Lee. SKATER-CON: Unsupervised Regionalization via Stochastic Tree Partitioning Within a Consensus Framework Using Random Spanning Trees. In *SIGSPATIAL*, pages 33–42, 2018.
[7] R. Benedetti and et. al. The Identification of Spatially Constrained Homogeneous Clusters of Covid-19 Transmission in Italy. *RSPP*, 12:1169–1187, 2020.
[8] U. Benlic and J.-K. Hao. An Effective Multilevel Tabu Search Approach for Balanced Graph Partitioning. *Operations Research*, 38(7):1066–1075, 2011.
[9] D. Bereznyi, A. Qutbuddin, Y. Her, and K. Yang. Node-attributed spatial graph partitioning. In *SIGSPATIAL*, pages 58–67, 2020.
[10] S. Biswas and et. al. REGAL: A Regionalization Framework for School Boundaries. In *SIGSPATIAL*, pages 544–547, 2019.
[11] U. C. Bureau. TIGER/Line Shapefile, 2016, Series Information for the Current Census Tract State-based Shapefile, 2019. https://catalog.data.gov/dataset.
[12] E. Chambers and et. al. Aggregating Community Maps. In *SIGSPATIAL*, pages 1–12, 2022.
[13] D. Combe, C. Largeron, E. Egyed-Zsigmond, and M. Géry. Combining Relations and Text in Scientific Network Clustering. In *ASONAM*, pages 1248–1253, 2012.
[14] A. Das, S. Ghosh, K. Das, T. Basu, I. Dutta, and M. Das. Living Environment Matters: Unravelling the Spatial Clustering of COVID-19 Hotspots in Kolkata Megacity, India. *Sustainable Cities and Society*, 65:102577, 2021.
[15] D. Delling, D. Fleischman, A. V. Goldberg, I. Razenshteyn, and R. F. Werneck. An Exact Combinatorial Algorithm for Minimum Graph Bisection. *Mathematical Programming*, 153(2):417–458, 2015.
[16] J. C. Duque, R. L. Church, and R. S. Middleton. The P-Regions Problem. *Geographical Analysis*, 43:104–126, 2011.
[17] J. C. Duque and et. al. The Max-P-Regions Problem. *JRS*, 52:397–419, 2012.
[18] J. C. Duque, R. Ramos, and J. Suriñach. Supervised Regionalization Methods: A Survey. *IRSR*, 30:195–220, 2007.
[19] U. Feige and R. Krauthgamer. A Polylogarithmic Approximation of the Minimum Bisection. *SICOM*, 31(4):1090–1118, 2002.
[20] A. Felner. Finding Optimal Solutions to the Graph Partitioning Problem with Heuristic Search. *AMAI*, 45(3):293–322, 2005.
[21] D. C. Folch and S. E. Spielman. Identifying Regions Based On Flexible User-defined Constraints. *IJGIS*, 28:164–184, 2014.
[22] Gedicke and et. al. Aggregating Land-use Polygons Considering Line Features as Separating Map Elements. *CGIS*, 48(2):124–139, 2021.
[23] J.-H. Haunert and A. Wolff. Area Aggregation in Map Generalisation by Mixed-integer Programming. *IJGIS*, 24(12):1871–1897, 2010.
[24] J. Hurley. Regionalization and the Allocation of Healthcare Resources to Meet Population Health Needs. *Healthcare Papers*, 5:34–39, 2004.
[25] Y. Kang and A. Magdy. EMP: Max-P Regionalization with Enriched Constraints. In *ICDE*, 2022.
[26] H. Kim, Y. Chun, and K. Kim. Delimitation of Functional Regions Using a P-Regions Problem Approach. *IRSR*, 38:235–263, 2015.
[27] K. Kim and et. al. Spatial Optimization for Regionalization Problems with Spatial Interaction: a Heuristic Approach. *IJGIS*, 30(3):451–473, 2016.
[28] K. Kim and et. al. p-Functional Clusters Location Problem for Detecting Spatial Clusters with Covering Approach. *Geographical Analysis*, 49:101–121, 2017.
[29] Y. Kong, Y. Zhu, and Y. Wang. A Center-based Modeling Approach to Solve the Districting Problem. *IJGIS*, 33(2):368–384, 2019.
[30] J. Laura, W. Li, S. J. Rey, and L. Anselin. Parallelization of a Regionalization Heuristic in Distributed Computing Platforms–a Case Study of Parallel-P-Compact-Regions Problem. *IJGIS*, 29:536–555, 2015.
[31] C. Li, Y. Yin, X. Liu, and P. Wu. An Automated Processing Method for Agglomeration Areas. *ISPRS*, 7(6):204, 2018.
[32] C. Li, Y. Yin, P. Wu, and W. Wu. An Area Merging Method in Map Generalization Considering Typical Characteristics of Structured Geographic Objects. *CGIS*, 48(3):210–224, 2021.
[33] W. Li, R. L. Church, and M. F. Goodchild. The p-Compact-Regions Problem. *Geographical Analysis*, 46:250–273, 2014.
[34] Y. Liang and et. al. Region2Vec: Community Detection on Spatial Networks Using Graph Embedding with Node Attributes and Spatial Interactions. In *SIGSPATIAL*, pages 1–4, 2022.
[35] Y. Liu, A. R. Mahmood, A. Magdy, and S. Rey. PRUC: P-Regions with User-Defined Constraint. In *VLDB*, pages 491–503, 2022.
[36] J. Oehrlein and J.-H. Haunert. A Cutting-Plane Method for Contiguity-Constrained Spatial Aggregation. *JOSIS*, (15):89–120, 2017.
[37] T. R. Depth-first Search and Linear Graph Algorithms. *SICOM*, 1:114–121, 1971.
[38] S. Rey. Random Regions, 2021. https://github.com/sjsrey/spopt/blob /randomregion/notebooks/randomregion.ipynb.
[39] K. Schloegel, G. Karypis, and V. Kumar. Parallel Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *European Conference on Parallel Processing*, pages 296–310, 2000.
[40] B. She, J. C. Duque, and X. Ye. The network-max-p-regions model. *IJGIS*, 31:962–981, 2017.
[41] R. Wei, S. Rey, and E. Knaap. Efficient Regionalization for Spatially Explicit Neighborhood Delineation. *IJGIS*, 35:1–17, 2020.
[42] J. Yao, X. Zhang, and A. T. Murray. Spatial Optimization for Land-use Allocation: Accounting for Sustainability Concerns. *IRSR*, 41(6):579–600, 2018.
[43] X. Ye, B. She, and S. Benya. Exploring Regionalization in the Network Urban Space. *JGSA*, 2:4, 2018.
[44] Y. Zhou, H. Cheng, and J. X. Yu. Graph Clustering Based on Structural/Attribute Similarities. *PVLDB*, 2:718–729, 2009.

APPENDIX

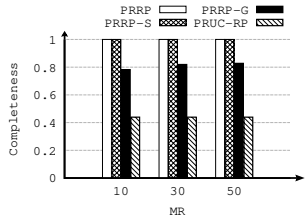
A ADDITIONAL EXPERIMENTS



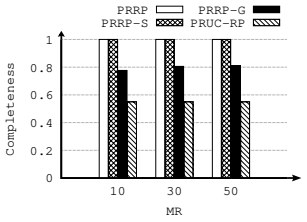
(a) Effect on time under the *CT* dataset



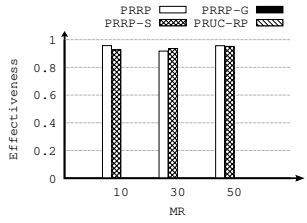
(b) Effect on time under the *CS* dataset



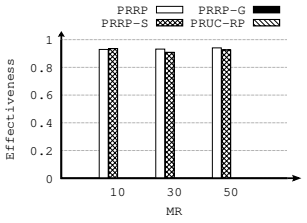
(c) Effect on completeness under the *CT* dataset



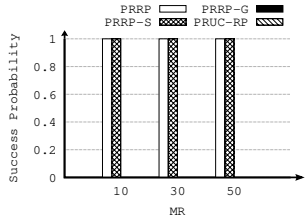
(d) Effect on completeness under the *CS* dataset



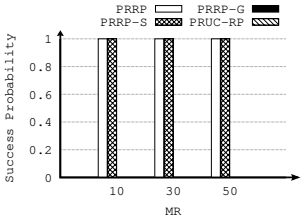
(e) Effect on effectiveness under the *CT* dataset



(f) Effect on effectiveness under the *CS* dataset



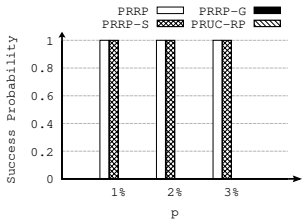
(g) Effect on success probability under the *CT* dataset



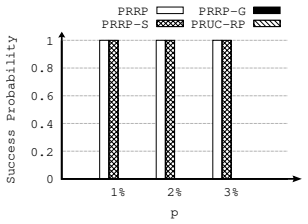
(h) Effect on success probability under the *CS* dataset

Figure 10: Impact of the maximum number of iterations for building a region (*MR*)

Impact of the maximum number of iterations (MR). The maximum number of iterations for building a region *MR* is a parameter for *PRRP* and it variations *PRRP-S* and *PRRP-G* as the seed selection and region building techniques are randomized. In *PRUC-RP*, the seeds are fixed and the most connected area is added to the region, therefore, attempting to grow the region again will always produce the same result. Figure 10a and Figure 10b show that increasing the *MR* value has a slight impact on the runtime of *PRRP*, *PRRP-S*, and *PRRP-G*. *PRRP* and *PRRP-S* still have a high success probability and completeness as they produce feasible sample solutions with valid regions for all *MR* values (shown in Figure 10c, Figure 10d, Figure 10g, and Figure 10h). Similarly, Figure 10e and Figure 10f show that the effectiveness of *PRRP* and *PRRP-S* is almost constant (between 0.90 and 0.95 for both the *CT* and *CS* datasets) under different *MR* values. Even with multiple attempts for growing a region, *PRRP-G* still have a zero success probability and effectiveness (Figure 10e, Figure 10f, Figure 10g, and Figure 10h) as it fails to maintain the spatial contiguity of unassigned areas.

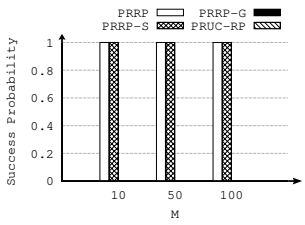


(a) Effect on success probability under the *CT* dataset

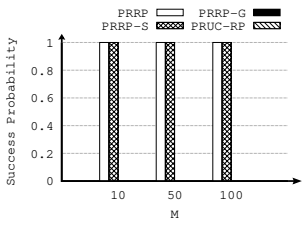


(b) Effect on success probability under the *CS* dataset

Figure 11: Impact of the number of regions (*p*)



(a) Effect on success probability under the *CT* dataset

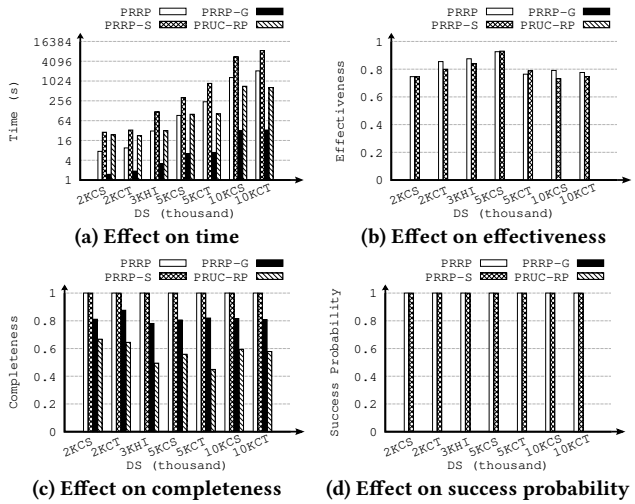


(b) Effect on success probability under the *CS* dataset

Figure 12: Impact of the sample size (*M*)

The success probability under different parameter values. For all values of all performance measures *p* (Figure 11), and *M* (Figure 12), *PRRP* and *PRRP-S* always produce feasible sample solutions and achieve the highest probability of success (i.e., 1). On the other hand, *PRRP-G* and *PRUC-RP* both fail to generate feasible sample solutions with a zero success probability. This is due to the fact that *PRRP* and *PRRP-S* maintain spatial contiguity of input areas which allows to generate valid regions.

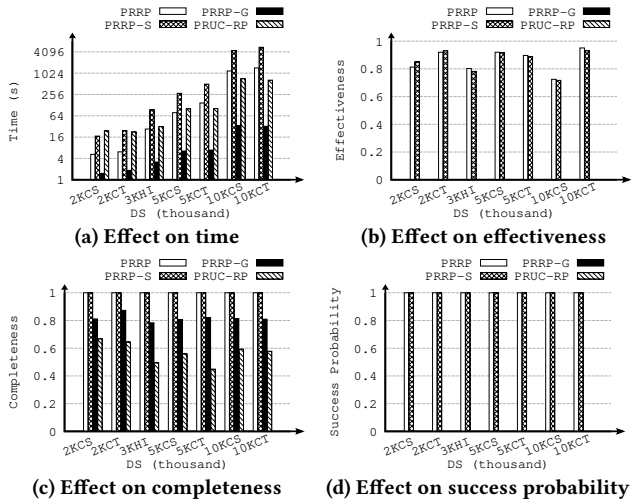
Impact of the seed selection technique. In addition to our gapless random seed selection technique, we employ two techniques for *PRRP* and its variations *PRRP-S* and *PRRP-G*: (1) a random technique that picks the seed area randomly from the unassigned areas, and (2) a modified gapless random technique that picks the seed from the spatial neighbors of all regions. Figure 13 shows that the random seed selection technique is 1.3 times slower than our technique with less effectiveness. This is because the random seed selection technique builds regions without minimizing the gaps between them which increases the probability of breaking the spatial contiguity of the unassigned areas into two or more connected components with a larger size duo to the large space between regions. This leads to having to merge/remove more areas in the region merging and region splitting phases which increases the runtime of the algorithm. In addition, having a larger number of areas to remove in the region splitting phase decreases the probability of building valid regions, which in turn, decreases the effectiveness.



(a) Effect on time (b) Effect on effectiveness

(c) Effect on completeness (d) Effect on success probability

Figure 13: Random seed selection



(a) Effect on time (b) Effect on effectiveness

(c) Effect on completeness (d) Effect on success probability

Figure 14: Modified gapless random seed selection

On the other hand, Figure 14 shows that the modified gapless random technique has no apparent effect as the results are comparable to those produced by our gapless random technique.

B COMPLEXITY ANALYSIS

This appendix provides a complexity analysis for *PRRP*. We denote the the size of the unassigned areas A^u as $s(A^u)$ and the size of the spatial neighbors SN_a of an area a as $s(SN_a)$.

Preprocessing. The preprocessing step first reads n areas which takes $O(n)$. Then it builds the neighborhood graph and the degree list by checking the boundary of each area against all the areas which takes $O(n^2)$ time. So the overall complexity of the preprocessing is $O(n) + O(n^2) = O(n^2)$.

Region growing phase. The region growing phase randomly adds an area to a region in $O(1)$ time. Adding n areas takes $O(1) \times n = O(n)$. After growing the region, we compute the number of connected components for the remaining unassigned areas A^u by performing a single graph traversal over the areas in A^u. In the worst case, $s(A^u) = n - 1$, therefore, it takes $O(n)$ to compute the connected components for A^u. So, the overall time complexity of the region growing phase is $O(n) + O(n) = O(n)$.

Region merging phase. The region merging phase merges the smallest connected components with the region. In the worst case, the total size of the smallest connected components cannot exceed n, so its worst case complexity is $O(n)$.

Region splitting phase. This phase involves two steps. **First, the shrinking step** identifies the region's boundary areas, removes them, and then computes the number of connected components for the region. Identifying the boundary areas BA_r for any region r involves evaluating the spatial neighbors SN_a of every area a in r. The set of areas A in our problem is represented as a planar graph of the spatial neighborhood relation among areas in A. In the worst case, the degree of a vertex is $O(n-1)$ and it takes $O((n-1) \times n) = O(n^2)$ to identify the region's boundary areas. There are a total of k areas to be removed in the shrinking step. In the worst case, we need to identify the region's boundary areas each time an area is removed from the region. Since $k < n$, i.e., $k = O(n)$, the time complexity for removing k boundary areas is $O(n^3)$. Computing the number of connected components for a region performs a single graph traversal over the region's areas which takes $O(n)$ time in the worst case. Consequently, the time complexity for the shrinking step is $(O(n^3) + O(n)) = O(n^3)$ in its worst case. **Second, the expansion step** identifies A^u's boundary areas and removes the articulation areas from the list of boundary areas. Identifying the boundary areas BA_r for A^u is similar to the shrinking step and takes $O(n^2)$ in the worst case. Identifying the articulation areas for A^u is done over a single graph traversal using Tarjan's algorithm [37] and takes $O(n)$. So, identifying the A^u's boundary areas and articulation areas takes $O(n) + O(n^2) = O(n^2)$ in the worst case. To add any area from the boundary areas back to the region, its nested areas must be also added with the area. There are w areas to add where $w < n$, i.e., $w = O(n)$ and e nested areas which gives a total complexity of $O(e \times n)$. In the worst case, we need to identify A^u's boundary areas each time an area is added to the region. Therefore, the time complexity for adding w boundary areas is $O(e \times n^3)$ in the worst case. So, the total time complexity for the expansion step is $O(e \times n^3) + O(n) = O(e \times n^3)$ in the worst case.

Overall complexity. The overall time complexity of the three phases excluding the preprocessing is $O(n) + O(n) + O(e \times n^3) = O(e \times n^3)$ in the worst case. The three phases are repeated p times to grow p regions and each region is constructed MR times in the worst case. M sample solutions are generated in parallel using q cores and each sample solution is constructed MS times in the worst case. So, this gives an overall complexity of $\frac{M}{q} \times MS \times MR \times p \times O(e \times n^3)$.