IMS: Incremental Max-P Regionalization With Statistical Constraints

Yunfan Kang, Yiyang Bian, Qinma Kang, and Amr Magdy, *Senior Member, IEEE*

***Abstract*—Spatial regionalization is the process of grouping a set of spatial areas into spatially contiguous and homogeneous regions. This paper introduces an *Incremental Max-P regionalization with statistical constraints* (IMS) problem; a regionalization process that supports enriched user-defined constraints based on statistical aggregate functions and supports incremental updates. In addition to enabling richer constraints, it allows users to employ multiple constraints simultaneously to significantly push the expressiveness and effectiveness of the existing regionalization literature. The IMS problem is NP-hard and significantly enriches the existing regionalization problems. Such a major enrichment introduces several challenges in both feasibility and scalability. To address these challenges, we propose the *FaCT* algorithm, a three-phase heuristic approach that finds a feasible set of spatial regions that satisfy IMS constraints while supporting large datasets compared to the existing literature. *FaCT* supports local and global incremental updates when there are changes in attribute values or constraints. In addition, we incorporate the Iterated Greedy algorithm with *FaCT* to further improve the solution quality of the IMS problem and the classical max-p regions problem. Our extensive experimental evaluation has demonstrated the effectiveness and scalability of our techniques on several real datasets.**

***Index Terms*—Spatial, regionalization, max-P, aggregate.**

I. Introduction

REGIONALIZATION is the process of clustering a set of areas into spatially contiguous and homogeneous regions. Unlike traditional clustering of multi-dimensional points, the basic units in regionalization are spatial polygons that are grouped based on both attribute values and adjacency in space. Regionalization is widely used in numerous applications such as epidemic analysis [1], service delivery systems [2], water quality assessment [3], weather temperature classification [4], rainfall erosivity estimation [5], health data analysis [6], spatial crowdsourcing [7], [8], and constituency allocation [9]. A fundamental challenge in regionalization is determining the appropriate spatial scale for the phenomenon under study. For example, as shown in Fig. 1,[1] different climate variables can exhibit distinct spatial patterns. Season-accumulated snowfall shows strong coherence across large geographic units—aggregating at the state level (≈ 50 regions) sufficiently captures regional variation. In contrast, the mean annual temperature varies sharply at finer scales, necessitating county-level granularity (≈ 3,000 regions) to resolve meaningful patterns. This spatial-scale mismatch underscores the need for adaptive regionalization techniques that account for the intrinsic heterogeneity of different attributes. However, most existing methods require users to manually specify the number of regions, placing a burden on analysts to try multiple values, which increases the computational cost and limits the query expressiveness.

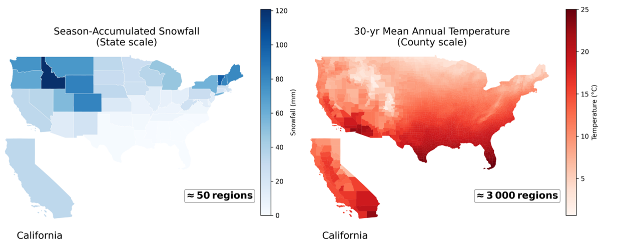


Fig. 1. Spatialscale mismatch in U.S. climate data: (a) statelevel snowfall (≈ 50 regions) vs. (b) countylevel temperature (≈ 3 000 regions).

The latest regionalization formulation, called the *max-p-regions* problem (or MP-regions), has addressed the scale problem [10] and is widely used across applications [11], [12], [13], [14], [15], [16], [17]. With the observation that researchers usually know a condition that a region must satisfy to be suitable for the analysis, the MP-regions automatically finds the maximum number of regions satisfying a user-defined constraint, without requiring users to input p. In many critical applications, from analyzing a pandemic like COVID-19 to designing sales territories, the ideal number of regions is unknown beforehand. Fixing the number of regions, p, in advance, as required by traditional clustering, imposes an arbitrary structure that can obscure meaningful insights. This focus on maximizing p is therefore not merely a technical detail; it is the critical mechanism that drives a truly data-driven and objective partitioning. By pushing p to its maximum possible value—subject to essential constraints such as spatial contiguity and minimum attribute thresholds—the model is forced to reveal the most

Received 27 August 2024; revised 9 October 2025; accepted 12 October 2025. Date of publication 15 October 2025; date of current version 25 November 2025. This work was partially supported by National Science Foundation, USA, under Grant IIS-2237348 and Grant CNS-2031418, in part by Google-CAHSI, and in part by Microsoft unrestricted gift. Recommended for acceptance by A. Asudeh. *(Corresponding author: Yunfan Kang.)*

Yunfan Kang is with the Cyber GIS Center for Advanced Digital and Spatial Studies, Department of Geography & GIS, University of Illinois Urbana Champaign, Urbana, IL 61801 USA (e-mail: yfkang@illinois.edu).

Yiyang Bian and Amr Magdy are with the Department of Computer Science and Engineering, University of California, Riverside, Riverside, CA 92521 USA (e-mail: ybian011@ucr.edu; amr@cs.ucr.edu).

Qinma Kang is with the Department of Computer Science, School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai 264209, China (e-mail: qmkang@sdu.edu.cn).

Digital Object Identifier 10.1109/TKDE.2025.3621843

[1]Source: NOAA NOHRSC (2023–24 seasonal snowfall) and PRISM Climate Group (1991–2020 mean annual temperature).

granular and meaningful spatial structure inherent in the data. It is fundamental to achieving the highest possible resolution for local pattern discovery and ensuring the most equitable foundation for fair resource allocation. The power of this approach is evidenced by its adoption in leading geospatial software, including ESRI ArcGIS Pro [18] and PySAL [19]. As introduced in the MP-regions formulation [10], maximizing the number of regions reveals fine-grained spatial structure but does not, on its own, ensure meaningful partitions. To address this, minimizing intra-region heterogeneity is incorporated as a secondary objective, promoting internally coherent clusters with consistent characteristics. This further enhances analytical robustness and guards against arbitrary groupings. In this two-fold framework, region count drives the partition's resolution, while homogeneity serves as a refinement and tie-breaking criterion to select the most interpretable solution.

However, the MP-regions problem has several limitations in both scalability and expressiveness. Currently, the state-of-the-art algorithms take 3-60 minutes to process a dataset with $\sim$ 3k areas, depending on the threshold value. As a result, existing real applications, listed above, use relatively small data, ranging from tens of areas [13], [15], [20] to only a few hundred [11], [12], [14], [16], [17]. It is explicitly stated in [16] that to use the MP-regions algorithms "*one must be willing to reduce the number of geographic units of analysis.*" Such severe scalability limitations prevent the existing MP-regions formulation from supporting a wide variety of user-defined constraints that enable users to express flexible regionalization queries for various applications.

More fundamentally, the problem of expressiveness is a structural limitation of the classic MP-regions formulation. Its reliance on a single, lower-bound SUM constraint on one attribute precludes common analytical queries that require multiple simultaneous conditions, range-based thresholds, or non-monotonic functions like AVG or VAR. However, real-world applications often require multiple, complex constraints simultaneously. Effective COVID-19 analysis may target regions with a total population $\geq$ 200000, an average monthly income between \$3000 to \$5000, and public transportation use $\geq$ 10000 [1]. Similarly, population studies may combine metrics like *minimum* population, *maximum* school drop-out rate, and the *average* age—to form meaningful regions [13]. Other applications, like designing police patrol sectors, also depend on balancing multiple criteria [20]. In addition, internal consistency—like low variance in key indicators—is important for effective regionalization. For instance, during COVID-19, grouping areas with similar infection rates helped ensure consistent policies and resource allocation [1]. However, in cases like equity planning or urban–rural integration, allowing higher variance can support broader goals. These examples show the need for flexible, user-defined constraints, such as adjustable variance thresholds, which traditional MP-regions methods do not support. Enhancing their expressiveness is therefore necessary. Such applications highlight the need to enhance MP-regions' expressiveness, which currently lacks support for advanced user-defined constraints.

This paper introduces the Incremental Max-P regionalization with statistical constraints (IMS) problem, which addresses the existing MP-regions limitations. IMS is designed to handle significantly larger datasets and enriches regionalization queries with four novel features: (1) a full suite of SQL-inspired aggregate functions (MIN, MAX, AVG, VAR, SUM, COUNT), (2) support for range-based comparisons ($\leq$, $\geq$, and between), (3) the use of multiple simultaneous constraints, and (4) efficient incremental updates for explorative analysis.

The IMS problem is NP-hard (see Section IV), so finding an optimal solution is intractable. An exact MIP solver [21], [22] fails to solve an instance with just 25 areas in a reasonable time [23]. Even finding an approximate solution is challenging due to: (1) non-monotonic constraints (e.g., AVG, VAR), where adding or removing an area gives no guarantee of satisfying the constraint, and (2) the risk of forming oversized regions, which contradicts the primary objective of maximizing the number of regions, p.

To address these challenges, we introduce *FaCT*, a three-phase algorithm for IMS. The first phase derives bounds to check feasibility under user-defined constraints. If infeasible, it suggests how users can adjust the data to satisfy constraints. This flexibility supports adaptive tuning and exploratory analysis across datasets. The second phase constructs a feasible solution that satisfies constraints while maximizing p.

To handle the challenge of non-monotonic constraints, this phase is divided into three independent steps, each dedicated to a specific constraint family. The steps are designed with the following features: (a) Each step targets a specific constraint type to maximize p. (b) Step order reduces search space and constraint conflicts. (c) The independent steps handle arbitrary constraint combinations and updates.

To mitigate the challenge of oversized regions, which hinders maximizing p, *FaCT* adopts a seed-based growing strategy with bounded merging. Each step initiates region growth from diverse seeds, which define an upper bound on the achievable p, and restricts merging based on aggregation properties. This ensures that regions remain compact and feasible, preserving the primary objective of maximizing the number of valid regions. In addition, *FaCT* also offers (d) parameter tuning for different use cases, and (e) datasets with multiple connected components, while the MP-regions formulation supports only a single connected component. The third phase of *FaCT* performs a local search adapting the Tabu search [24] and swapping areas between regions to improve the overall heterogeneity of the regions.

This paper is a major extension of our work in [25]. Firstly, we introduced the novel *variance* constraint with range operators, advancing regionalization expressiveness. It also opens opportunities for improving related methods. The integration of *variance* constraint poses substantial computational challenges due to its complexity, yet it significantly enhances the expressiveness of regionalization, enabling a broader spectrum of applications and analysis tasks. Secondly, the algorithm framework is modified to support global and local incremental updates, enabling fast query revisions without full recomputation. The incremental updates also allow users to reconfigure the scale of the regions efficiently, better addressing the scaling issue, which is a major concern of the regionalization problems and the core motivation for the MP-regions problem. Thirdly, we integrate Iterated Greedy, which improves both IMS and the widely used MP-regions formulation. This highlights the potential of metaheuristics in regionalization and spatial analysis.

Our experimental evaluation demonstrates the effectiveness of *FaCT* on real datasets. We evaluate the impact of different threshold ranges, different combinations of constraints, and the scalability to process datasets. Our contributions are summarized as follows:

- We define an incremental max-p regions problem with statistical constraints (IMS) that reveals the full potential of max-p regionalization.
- We prove the NP-hardness of the IMS problem.
- We propose *FaCT*; a three-phase heuristic algorithm that provides approximate solutions for the IMS problem and responds to the updates in attributes and constraints effectively and efficiently.
- We introduce *FaCT-IG*, the combination of the iterated algorithm and the *FaCT* construction to further improve the p value of the IMS and the MP-regions problem.
- We perform an extensive experimental evaluation that shows the effectiveness of our techniques on real datasets.

The rest of this paper is organized as follows. Section II presents the related work. Section III defines the problem and Section IV proves its NP-hardness. Our proposed solution is detailed in Section V, and its time and space complexity are analyzed in Section VI. Section IX shows the experimental evaluations and Section X concludes the paper.

II. Related Work

The regionalization problem originates from the demand for aggregating homogeneous regions in the analytical tasks performed by social scientists and statisticians [26]. Regionalization is referred to by different names, including region building [27], conditional clustering [28], clustering with relational constraints [26], contiguity-constrained clustering [29], constrained classification [30], maximum split clustering under connectivity constraints [31], and p-regions problem [32], [33] that is used in a variety of applications such as detecting functional regions [34], [35], network-constrained urban management [36], and partitioning compact regions [37], [38], [39]. The max-p-regions problem (MP-regions) [10] has been introduced to eliminate the requirement for inputting the number of regions in advance. It aggregates areas into a maximum number of regions so that each region satisfies a single user-defined constraint with a summation aggregate, e.g., total population $\geq$ 20K. Existing variants [13], [40], [41] use the road network-based connectivity as an additional spatial constraint to aggregate regions or use multiple attributes to impose the constraint. However, all existing variants still support only a single constraint of one type as the MP-regions problem.

Existing regionalization techniques that perform multi-criteria districting [13], [20], [42], [43], [44] use multi-objective optimization to partition the space based on multiple attributes. These techniques still put the burden on the user to determine the spatial scale, i.e., the number of output regions. Additionally, they allow the user to define only a single constraint on one attribute, which limits query expressiveness. Our IMS problem addresses these limitations by enabling users to define multiple constraints across different attributes, thereby providing more flexible and expressive queries. Furthermore, it inherits the automatic discovery of the appropriate number of regions from the MP-regions approach, effectively resolving the issue of spatial scale.

We classify the popular approaches to tackle the NP-hard regionalization problems into two main categories. The first category is the clustering methods [45], [46] with two phases. The first phase clusters the centroids of polygons using a traditional clustering algorithm. The second phase then imposes the spatial connectivity constraint to ensure that each region is geographically connected. The second category adopts a two-phase contiguity-constrained heuristic optimization approach [10], [41], [47], [48], [49], [50]. The set of spatial areas is usually represented by a graph that encodes the spatial contiguity relationships. With the spatial contiguity constraint imposed explicitly, a feasible initial solution is constructed in the first phase and then optimized using heuristic or mixed-heuristic search methods in the second phase. The construction methods include tree partition [47], [51] and greedy aggregation [10], [40], [48], [49], [50]. The heuristic search phase optimizes the objective function to improve the region homogeneity [47], [48], [50], [51] or geographical compactness [48] without violating the contiguity constraint.

Our work follows the second category of contiguity-constrained heuristic optimization search. We consider multiple constraints with different aggregate functions that do not exist in the literature, so none of the existing methods can obtain a feasible solution that satisfies our enriched constraints.

III. Problem Definition

This section formally defines the incremental max-p regionalization with statistical constraints (IMS) problem. Let $A = \{a_1, a_2, \ldots, a_n\}$ denotes the set of areas. Each area a_i is defined by 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 set of spatially extensive attributes and d_i is a dissimilarity attribute. The dissimilarity attribute d_i is used in computing output regions' heterogeneity. The rest of the section presents basic definitions, then defines the IMS problem formally.

Definition III.1 (User-defined Constraint). A user-defined constraint c is a condition that is defined as "$l \leq f(s) \leq u$" or "$f(s) \in [l, u]$". c is identified with a 4-tuple: (f, s, l, u), where f is an aggregate function, s is a spatially extensive attribute, $l \in [-\infty, \infty)$ is a number that represents a lower bound, and $u \in (-\infty, \infty]$ is a number that represents an upper bound.

IMS allows f to be MIN, MAX, AVG, VAR, SUM, or COUNT that computes minimum, maximum, average, variance, summation, and count aggregates, respectively, with the same semantics as the SQL standards. Following standard definitions in the regionalization literature [10], [52] and cartography [53], we define s as a spatially extensive attribute (e.g., population, total income) that is additive and scales with the size of the spatial unit. For example, the *population* value of a state is divided over its cities, so the *population* is a spatially extensive attribute. In contrast, spatially intensive attributes [53], such as *temperature*, are normalized or averaged values that capture relative characteristics independent of area size. Examples of s include the population, labor force, and households in each area. When $l = -\infty$, the range has an open-ended lower bound and the constraint becomes $f(s) \leq u$. Similarly, when $u = \infty$, it becomes $f(s) \geq l$.

Definition III.2 (Region). A region R is a set of g areas: $R = \{a_1, a_2, \ldots, a_g\}$, such that: (1) $g \geq 1$, i.e., R contains at least one area. (2) Areas in R are spatially contiguous, i.e., $\forall$ $a_i, a_j \in R$, $\exists$ a sequence of areas $(a_k, \ldots, a_l)$ s.t. both (a_i, a_k) and (a_l, a_j) are spatial neighbors, and every two consecutive areas in the sequence are spatial neighbors.

Definition III.3 (Heterogeneity). Heterogeneity $H(P)$ of a set of regions P is defined as:

$$H(P) = \sum_{\forall R \in P} \sum_{\forall a_i, a_j \in R} |d_i - d_j| \qquad (1)$$

While (1) defines heterogeneity using a specific, widely accepted formulation from the regionalization literature [10], [11], [13], [14], [15], [16], [52], our framework is not limited to this measure. The optimization of $H(p)$ occurs during the flexible local search phase in Section V, which can easily accommodate alternative heterogeneity definitions, including measures based on multi-attribute dissimilarity.

IMS Problem: The IMS problem is defined as follows:

Input: Given: (1) A set of n areas: $A = \{a_1, a_2, \ldots, a_n\}$. (2) A set of user-defined constraints $C = \{c_1, c_2, \ldots, c_m\}$.

Output: (1) A set of regions $P = \{R_1, R_2, \ldots, R_p\}$, where $1 \leq p \leq n$ and each region R_i satisfies the below IMS constraints and objectives. (2) A set $U_0 = A - \bigcup_{i=1}^{p} R_i$, i.e., U_0 contains all areas that are not assigned to any feasible region R_i, $\forall\, 1 \leq i \leq p$. Areas in U_0 may not be spatially contiguous.

IMS Constraints:

- $R_i \cap R_j = \emptyset, \qquad \forall R_i, R_j \in P \wedge i \neq j$
- R_i satisfies all constraints $c_j \in C, \forall 1 \leq i \leq p, 1 \leq j \leq m$

Objectives:

- Maximizing the number of regions p.
- Minimizing regions' overall heterogeneity $H(P)$.

IMS has two objectives. In case they contradict during building the output regions, the first objective (maximizing the number of regions) is favored over the second one (minimizing regions' heterogeneity). This allows users to obtain the maximum number of desirable regions without providing the number of regions as an input, as favored by the domain experts [10], which addresses a major limitation in the previous regionalization problems. The second objective uses a dissimilarity attribute that is not necessarily a spatial attribute. For example, social scientists may produce regions that are homogeneous in terms of average income level.

To clarify the distinction from MP-regions, we highlight two fundamental differences that arise from IMS's enriched constraints. First, the enriched constraints are not always monotonic. Assuming all spatially extensive attribute values are positive, adding or removing an area in MP-regions guarantees a monotonic change toward satisfying the constraint threshold. This is not the case for IMS due to its support for AVG, VAR, MIN, and MAX constraints. Additionally, IMS allows upper-bounded threshold ranges, meaning that adding areas without further validation can violate the upper bound of SUM and COUNT constraints. Second, unlike MP-regions, IMS allows unassigned areas U_0 to exist in the final partition. This design choice makes it possible to handle more complex and potentially conflicting constraint combinations. In IMS, allowing unassigned areas is a necessary design choice, not a formal optimization objective. It allows the algorithm to effectively handle complex constraint combinations without forcing infeasible assignments. For instance, MIN/MAX constraints act as filters, while non-monotonic AVG/VAR constraints require delicate balancing; forcing all areas into regions under these conditions could lead to violations or distort statistical patterns. While our algorithm strives to assign as many areas as possible, its primary objectives remain satisfying all user-defined constraints and maximizing the number of valid regions p. This design allows for more expressive and realistic regionalization outcomes.

IV. NP-Hardness of IMS

This section proves the NP-hardness of the IMS problem. The NP-hardness of the IMS problem follows the results that the MP-regions problem is NP-hard [10], [54], [55]. The MP-regions problem takes as input: (1) a set of spatially contiguous areas $A = \{a_1, a_2, \ldots, a_n\}$. Each area a_i is associated with an identifier i, a spatial polygon b_i, a spatially extensive attribute s_i and a dissimilarity attribute d_i, (2) a threshold value t. For the purpose of the proof, we solve both the MP-regions and the IMS problem as decision problems with a fixed $p = k$.

Theorem 1. The IMS problem is NP-hard.

Proof. Let X $= (A, t)$ be an instance of the MP-regions problem. We construct an instance Y $= (A', C)$ of the IMS problem as follows: (1) For each area $a_i \in X.A$, add to $Y.A'$ an area $a'_i = \{i, b_i, S'_i, d_i\}$, $S'_i = \{s_i\}$. (2) Let $C = \{(SUM, S'[0], t, \infty)\}$. (3) Let the number of regions p equal k. This reduction is of polynomial time of $O(n)$ where n is the number of areas in A. Therefore, an algorithm that decides on the instance Y of the IMS problem decides on the instance X of the MP-regions problem as well. As the MP-regions problem is NP-hard, then the IMS problem is NP-hard, and the proof is complete. □

V. Proposed Solution

This section presents *FaCT*, a three-phase algorithm for solving IMS with low heterogeneity. The first phase is a ***feasibility phase*** that checks feasibility under user-defined constraints. The second phase is a ***construction phase*** that builds an initial solution maximizing p while satisfying all constraints. This is the algorithm's core contribution. It balances two challenging objectives: satisfying multiple non-monotonic constraints and maximizing spatially contiguous regions. The third phase is a ***local search phase*** that improves the initial solution of the *construction phase* in terms of the heterogeneity score. Fig. 2 illustrates the FaCT algorithm's three-phase workflow, depicting the flow of data from the initial user inputs to the final optimized region set. The following sections detail each phase.

A. Feasibility Phase

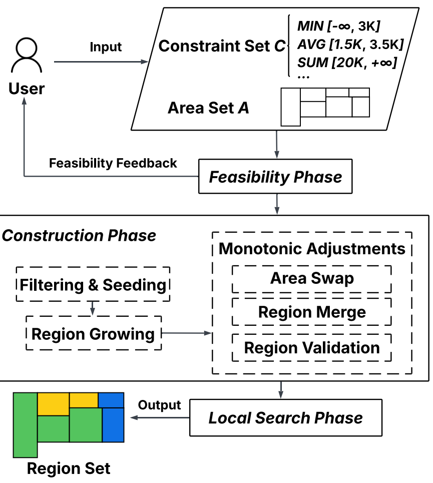


Fig. 2. FaCT framework.

The IMS problem handles arbitrary combinations of attribute constraints. The feasibility phase alerts users early to: (1) the infeasibility of finding a solution for the given set of constraints on the given dataset, and (2) the possibility of removing some areas to satisfy the given constraints. The fact that our algorithm can detect and filter out infeasible areas for all constraints, including non-monotonic constraints, gives it major practical advantages when employing multiple constraints. It not only alerts users to infeasibility but also automatically removes problematic areas, enabling flexible analysis across diverse datasets and constraint configurations.

Given a set of user-defined constraints $C = \{(MIN, s_{min}, l_{min}, u_{min}), (MAX, s_{max}, l_{max}, u_{max}), (AVG, s_{avg}, l_{avg}, u_{avg}), (VAR, s_{var}, l_{var}, u_{var}), (SUM, s_{sum}, l_{sum}, u_{sum}), (COUNT, s_{count}, l_{count}, u_{count})\}$, we check the feasibility of applying each type of constraint as follows:

(1) **For AVG constraints**, we compute the average value AVG(s_{avg}) of attribute s_{avg} over all areas. If $AVG(s_{avg}) < l_{avg}$ or $AVG(s_{avg}) > u_{avg}$, we infer that there exists no partition where all regions satisfy the AVG constraint without removing any area. This is justified by Theorem 3 below.

Theorem 2. If $\exists P = \{R_1, R_2, \ldots, R_p\}$ so that every region $R_i \in P$ satisfies $c = (AVG, s_{avg}, l_{avg}, u_{avg})$ and P contains all areas in a set A, then the average value $AVG(s_{avg})$ of s_{avg} over all areas of A satisfies c, i.e., $l_{avg} \leq AVG(s_{avg}) \leq u_{avg}$.

Proof. If $p = 1, i.e., P = \{A\}$, the proposition is trivially true. If $p > 1$, as all regions in P satisfy c, we have:

$$l_{avg} \leq R_i.AVG(s_{avg}) \leq u_{avg}, \forall i, 1 \leq i \leq p \tag{2}$$

where $R_i.AVG(s_{avg})$ represents the average value of attribute s_{avg} over areas in region R_i. Let $|R_i|$ denote the number of areas in region R_i. As P contains all areas, we have:

$$\sum_{i=1}^{p} |R_i| = n \tag{3}$$

Also, by definition:

$$AVG(s_{avg}) \times n = \sum_{i=1}^{p} (|R_i| \times R_i.AVG) \tag{4}$$

$$AVG(s_{avg}) = \frac{\sum_{i=1}^{p}(|R_i| \times R_i.AVG)}{n}$$

$$\leq \frac{\sum_{i=1}^{p} |R_i| u_{avg}}{n} \text{~} = \text{~} u_{avg} \tag{5}$$

$$AVG(s_{avg}) = \frac{\sum_{i=1}^{p}(|R_i| \times R_i.AVG)}{n}$$

$$\geq \frac{\sum_{i=1}^{p} |R_i| l_{avg}}{n} \text{~} = \text{~} l_{avg} \tag{6}$$

Equations (5) and (6) $\Rightarrow l_{avg} \leq AVG(s_{avg}) \leq u_{avg}$.

Hence, the Theorem 2 is true, and the proof is complete.□

Theorem 3. For an area set A and an AVG constraint $c = (AVG, s_{avg}, l_{avg}, u_{avg})$, if the average value of s_{avg} over A, $AVG(s_{avg})$, does not satisfy c, i.e., $AVG(s_{avg}) < l_{avg}$ or $AVG(s_{avg}) > u_{avg}$, then $\nexists P = \{R_1, R_2, \ldots, R_p\}$ such that P partitions all areas in A and $R_i \in P$ satisfies c, $\forall 1 \leq i \leq p$.

Theorem 3 is proved by proving its contrapositive Theorem 2.

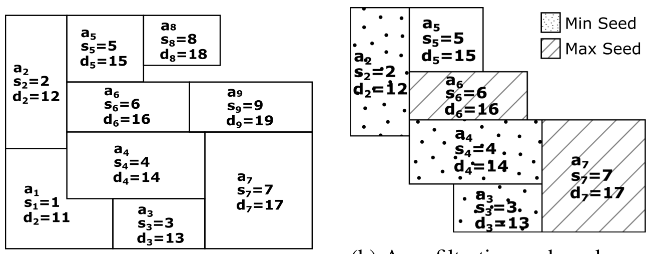
(2) **For MIN constraints**, we compute the minimum and maximum values over all areas for attribute s_{min}, $MIN(s_{min})$ and $MAX(s_{min})$, respectively. Two different cases could cause infeasibility: (a) If $MAX(s_{min}) < l_{min}$ or $MIN(s_{min}) > u_{min}$, then no area satisfies the MIN constraint and no valid regions can be constructed, so there is no feasible solution. (b) If $MIN(s_{min}) < l_{min} < MAX(s_{min})$, all areas with $s_{min} < l_{min}$ are infeasible areas that cannot be part of any valid region, so they must be filtered out to build a feasible solution. Therefore, there is room to find a feasible solution after removing infeasible areas.

(3) **Similarly for MAX constraints**, there are two cases of infeasibility: (a) If $MIN(s_{max}) > u_{max}$ or $MAX(s_{max}) < l_{max}$, no area that satisfies the MAX constraint, and there is no feasible solution. (b) Areas with $s_{max} > u_{max}$ are infeasible areas that must be filtered out to find a solution.

(4) **For SUM constraints**, we compute the minimum and summation of all areas for s_{sum}, $MIN(s_{sum})$ and $SUM(s_{sum})$, respectively. There are three cases of infeasibility: (a) If $MIN(s_{sum}) > u_{sum}$, no region can have s_{sum} within the range. (b) If $SUM(s_{sum}) < l_{sum}$, even the trivial region containing all areas cannot have s_{sum} within the range. (c) Areas with $s_{sum} > u_{sum}$ are infeasible and must be removed.

(5) **For COUNT constraints**, if the number of areas $n < l_{count}$, there is no feasible solution as a region containing all areas cannot meet the lower bound l_{count}.

The feasibility phase iterates through the area set and computes the needed attribute aggregates for all constraints in a single pass. This pass is enough to filter out infeasible areas, that have $s_{min} < l_{min}$, $s_{max} > u_{max}$, or $s_{sum} > u_{sum}$, to eliminate potential infeasible regions. However, the feasibility pass is not enough to filter out infeasible areas for the AVG constraint. This is performed during the following construction phase while imposing AVG constraints. In addition, the feasibility does not check for the VAR constraint due to its special mathematical property. Instead, VAR is closely tracked and efficiently updated



(a) Example area set

(b) Area filtration and seed area selection

Fig. 3. Step 1: Filtering and seeding.

as we prove in Section V-B. However, feasibility would check if AVG and VAR are specified together and signal the user that this would cause a high probability of failure because these two strict constraints tend to cause infeasible results when used at the same time. All infeasible areas are filtered out from A, added to the set U_0, and they are not considered in any further processing. The remaining areas in set A are passed to the following phases.

B. Construction Phase

The construction phase heuristically constructs a feasible solution that: (a) satisfies all user-defined constraints, (b) maximizes the number of regions p. This phase faces the challenges of simultaneously satisfying multiple non-monotonic constraints while avoiding oversized regions that reduce p. To address these challenges, each constraint family is handled in a separate step. This design offers several benefits. First, each step focuses on one type of optimization, simplifying the problem. Second, this step-wise design flexibly handles arbitrary constraint subsets (as discussed in Section V-D). Third, the order of steps makes use of the mathematical properties of different aggregate functions to increase the probability of producing a feasible solution. It first filters areas using MIN/MAX/SUM to seed regions, then applies AVG or VAR to grow them, and finally adjusts SUM/COUNT as needed. Fourth, separated steps facilitate maximizing the value of p, as detailed later, without violating previously satisfied constraints. Consequently, our construction phase produces an initial partition with near-optimal p for heterogeneity refinement. Fifth, the independence of each step enables incremental updates, reusing unaffected results to reduce computation (as discussed in Section VII). Although reducing the overall heterogeneity score is an objective for the final solution, this phase does not optimize heterogeneity, which is handled later.

The construction phase runs multiple iterations. Each iteration produces a feasible partition, and we maintain the partition with the highest p value. Each iteration is divided into three steps. The first step handles extrema (MIN, MAX), the second step handles centrality (AVG, VAR), and the third step handles counting (SUM, COUNT). Each step is detailed below. Fig. 3(a) provides a running example for illustration. For simplicity and without loss of generality, all constraints are assumed to apply to the same attribute s.

Step 1: Filtering and Seeding: This step handles extrema constraints, i.e., MIN and MAX constraints. They serve two purposes: **(a) filtering out the infeasible areas, and (b) specifying the set of seed areas**. Infeasible areas are already excluded during the feasibility phase (Section V-A). Seed areas are those meeting the bounds of any MIN or MAX constraint. For instance, for two constraints $c_1 = (MIN, s_1, l_1, u_1)$ and $c_2 = (MAX, s_2, l_2, u_2)$, any area with $l_1 \leq s_1 \leq u_1$ or $l_2 \leq s_2 \leq u_2$ is a valid seed area. This generalizes to any number of MIN/MAX constraints.

This step provides three levels of optimization through choosing a seed area that satisfies the bounds of only one constraint. First, it identifies seeds for any arbitrary combination of MIN/MAX constraints. When constraints span different attributes or have disjoint ranges, no area satisfies all. Handling constraints independently increase seed flexibility. Second, the number of seed areas is an upper bound for the number of regions p as each feasible region must contain at least one seed area for each constraint. Maximizing seed usage helps grow more regions and increases p. Third, it enables piggybacking the seed areas selection on the infeasible areas filtration during the *feasibility phase:* The algorithm simultaneously checks seeding conditions and marks valid seeds.

Example: Fig. 3(b) illustrates the result of Step 1 for the example area set in Fig. 3(a). The extrema constraints are set as $\{(MIN, s, 2, 4), (MAX, s, 6, 7)\}$. Areas a_1, a_8, and a_9 are moved to set U_0 because their attribute values are below 2 or above 7. The remaining areas after filtering are shown in Fig. 3(b). Area a_2, a_3, and a_4 are selected as seed areas for the MIN constraint and area a_6 and a_7 are selected as seed areas for the MAX constraint, as shown in Fig. 3(b).

Complexity analysis. All operations of the feasibility phase and Step 1 of the construction phase are performed in one pass over the n areas. Each area is validated against all MIN, MAX, and SUM constraints to be classified as either infeasible, valid, or seed area. The time complexity is $O(m \times n)$, where m is the number of constraints and typically $m << n$.

Remark 1. The time complexity of the feasibility phase and the Filtering and Seeding is $O(m \times n)$.

Step 2: Region Growing: The second step grows regions that satisfy the AVG constraint $c = (AVG, s, l, u)$ or the VAR constraint $c = (VAR, s, l, u)$, without violating the MIN/MAX constraints. This step is divided into three substeps for each constraint. The first substep initializes a set of regions. The second substep assigns the unassigned areas to the regions. The third substep combines regions to ensure each region satisfies all constraints. We first detail each substep for AVG in Substep 2.1 A to 2.3 A and then for VAR in Substep 2.1 V to 2.3 V below.

Substep 2.1 A utilizes the set of seed areas, called $seeds$, from Step 1 to initialize regions. Areas in $seeds$ satisfy at least one MIN/MAX constraint and must be part of a valid region. The algorithm iterates over $seeds$ and classifies all seed areas into three subsets based on their AVG attribute value s: $unassigned\_avg$, $unassigned\_low$, and $unassigned\_high$. $unassigned\_avg$ contains areas that satisfy the condition $l \leq s \leq u$. $unassigned\_low$ contains seed areas that satisfy $s < l$, i.e., s value is lower than c's lower bound. Similarly, $unassigned\_high$ contains seed areas with $s > u$. Only areas in $unassigned\_avg$ satisfy c and are used in region initialization

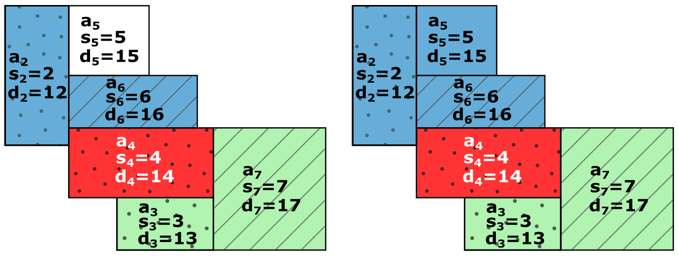
Algorithm 1: Region Growing - Merging Areas.

```
   Input: $c = (AVG, s, l, u)$, $unassigned\_low$,
          $unassigned\_high$, $P$
   Output: $unassigned\_low$, $unassigned\_high$, $P$
1  $removed\_areas \leftarrow \emptyset$
2  $u\_areas \leftarrow unassigned\_low \cup unassigned\_high$
3  foreach area $a \in u\_areas$ do
4      Create a temporary region $R$ with $R.id$ = -1;
5      $R$.addArea($a$);
6      $updated$ = true;
7      $neighbor\_areas$ = $R$.getNeighbors();
8      while $updated$ do
9          $updated$ = false;
10         if $l \leq R.getAverage() \leq u$ then
11             $R$.updateId($P$.size() + 1);
12             $P$.add($R$);
13             $removed\_areas$.addAll($R.areas$);
14         end
15         else
16             foreach area $a_n$ in $neighbor\_areas$ do
17                 if $a_n$ is not assigned to any region then
18                     if ($R.getAverage() < l$ AND $a_n.s > u$)
                         OR (($R.getAverage() > u$ AND $a_n.s <
                         l$)) then
19                         $R$.addArea($a_n$);
20                         $updated$ = true;
21                         break;
22                     end
23                 end
24             end
25         end
26     end
27 end
28 $unassigned\_low$.removeAll($removed\_areas$);
29 $unassigned\_high$.removeAll($removed\_areas$);
30 return $unassigned\_low$, $unassigned\_high$, $P$;
```

directly. As we want to maximize the number of output regions p, we make each area in $unassigned\_avg$ a separate region, and all new regions are added to a region list P. Then, we merge areas in $unassigned\_low$ and $unassigned\_high$ with their spatial neighbors to compose regions that satisfy c. Algorithm 1 gives the merging procedure.

Algorithm 1 initiates each unassigned area as a temporary region R (Lines 4 to 6). While R does not satisfy the constraint c's bounds, the algorithm tries to add a neighbor area a_n that moves R's overall average of attribute s towards c's range (Lines 16 to 21). Once R satisfies c, it is added to the region list P (Lines 10 to 12). If R's neighbors are exhausted and cannot form a valid region, the whole procedure is reverted, and areas of R remain unassigned.

Substep 2.2 A assigns remaining unassigned areas, including seeds and regular areas. Similar to classifying *seeds* areas in Substep 2.1, all areas that are not in *seeds* set are added to either $unassigned\_avg$ set, $unassigned\_low$ set, or $unassigned\_high$ set, depending on their s value. There are **two rounds** for assigning the remaining unassigned areas. In the first round, we try to add the unassigned areas to their neighbor regions. All areas in $unassigned\_avg$ can be safely added to any of its neighbor regions as it is guaranteed not to introduce a violation of c. For each area in $unassigned\_low$ and



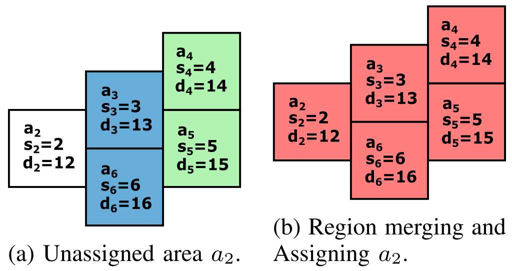
(a) Substep 2.1: Region Initialization
(b) Substep 2.2: Area Assignment

Fig. 4. Step 2: Region Growing - Substep 2.1 & 2.2-Round 1.

$unassigned\_high$, we must check if adding it to the neighbor region violates c. The second round attempts to assign the areas in $unassigned\_high$ and $unassigned\_low$ through controlled region merging. A merge limit parameter caps the number of merging attempts, preventing oversized regions and keeping runtime manageable, allowing users to balance between maximizing the number of regions and improving partition feasibility. Given an unassigned area a, the algorithm tries to merge one of a's neighboring regions R and R's neighbors with a and checks if c is satisfied for the newly merged region. If the merge limit is set to 0, the step is skipped; otherwise, the algorithm performs merging attempts up to the allowed limit. Substep 2.3 A merges neighboring regions to satisfy all MIN/MAX constraints. Since each area in *seeds* satisfies only one MIN or MAX constraint, its region inherits that single satisfaction. The algorithm scans the region list P and merges non-satisfying regions with neighbors that satisfy other constraints. This step preserves AVG constraint c since merging compliant regions keeps c valid. Thus, after Step 2, all regions in P satisfy MIN, MAX, and AVG constraints. However, if no suitable neighbor region exists, the region is discarded, and its areas are added to the unassigned set U_0. This aligns with IMS, which permits unassigned areas under strict or conflicting constraints.

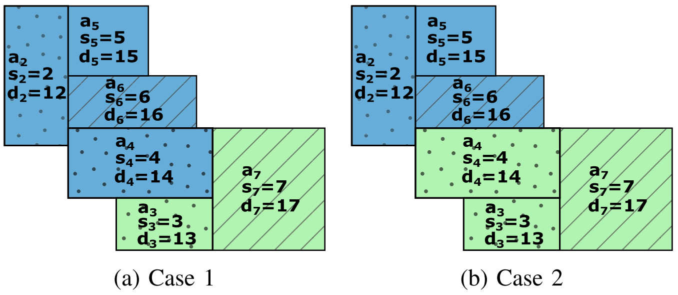
Example: Fig. 4 shows an example of Substep 2.1 and Substep 2.2. Given $c = (AVG, s, 4, 5)$, in Substep 2.1, only seed area a_4 is added to $unassigned\_avg$ and initialized as a new region R_{red} (Fig. 4(a)). a_2 and a_3 are added to $unassigned\_low$ and a_6 and a_7 are added to $unassigned\_high$. The average of $a_2.s_2$ and $a_6.s_6$ is 4, and they are combined to form the region R_{blue}. Similarly, a_3 and a_7 form the region R_{green} with average value 5. So, Substep 2.2 initializes three regions as depicted in Fig. 4(a). Then, in Substep 2.2, the remaining unassigned area a_5 is assigned to its neighbor region R_{blue} as depicted in Fig. 4(b).

Example: Fig. 5 provides an example for Round 2 of Substep 2.2 with merge limit $= 1$. In contrast to the example in Fig. 4, the unassigned area a_2 cannot be added to its neighbor region $R_{blue} = \{a_3, a_6\}$ directly. Adding a_2 to R_{blue} generates a region with an average s value equals 3.67, which is smaller than 4 and violates the constraint $c = (AVG, s, 4, 5)$. Because the merge limit allows one attempt, the algorithm attempts to merge $R_{blue} = \{a_3, a_6\}$ with its neighbor region $R_{green} = \{a_4, a_5\}$, forming a temporary region $R_{red} = \{a_3, a_4, a_5, a_6\}$, as shown in Fig. 5(b). R_{red} accepts a_2 and the average value is 4.4, which satisfies the constraint c.



(a) Unassigned area a_2. (b) Region merging and Assigning a_2.

Fig. 5. Step 2: Region Growing - Substep 2.2 - Round 2.



(a) Case 1 (b) Case 2

Fig. 6. Step 2: Region Growing - Substep 2.3.

Example: Fig. 6 gives an example of Substep 2.3 for the example region partition in Fig. 4(b). $R_{red} = \{a_4\}$ contains only the MIN seed area; hence it does not satisfy the MAX constraint. Because both neighbor regions $R_{blue} = \{a_2, a_5, a_6\}$ and $R_{green} = \{a_3, a_7\}$ satisfy the MAX constraint, R_{red} can be combined with either region to form a region satisfying all constraints. The results are depicted in Fig. 6(a) and (b), respectively.

VAR constraint computation follows the same three-step structure as AVG. Though both are nonmonotonic, they differ in one key aspect. The VAR constraint specifically manages the dispersion of regions. Thus, verifying compliance after adding or merging areas requires specific calculations. To support this, we introduce two equations for efficiently updating region variance and evaluating area or region mergers. For each region, we continuously update two sums: the sum of variance attribute values $\sum s$, and the sum of the squares of these values $\sum s^2$. The symbol $\sum$ here aggregates the values for all areas within a given region. The region's variance can be computed as:

$$\begin{aligned} VAR(R_i) &= \frac{\sum (s_j - \bar{s})^2}{|R_i|} \\ &= \frac{\sum (s_j^2 - 2\bar{s}s_j + \bar{s}^2)}{|R_i|} \\ &= \frac{\sum s^2}{|R_i|} - \frac{2\bar{s}\sum s}{|R_i|} + \frac{|R_i|\bar{s}^2}{|R_i|} \\ &= \frac{\sum s^2}{|R_i|} - 2\bar{s}^2 + \bar{s}^2 \\ &= \frac{\sum s^2}{|R_i|} - \left(\frac{\sum s}{|R_i|}\right)^2 \end{aligned} \tag{7}$$

By leveraging (7), we can deduce the potential change in a region's variance attribute value upon the addition of an area by simply adjusting $\sum s$ and $\sum s^2$. Furthermore, this allows us to identify the permissible range of variance attribute values for areas that can be added to the region. If x symbolizes the variance attribute value of a potential area, the variance post its addition is represented as $VAR(R_i')$, where $|R_i'| = |R_i| + 1$.

$$\begin{aligned} VAR(R_i') &= \frac{\sum s^2 + x^2}{|R_i| + 1} - \left(\frac{\sum s + x}{|R_i| + 1}\right)^2 \\ &= \frac{\sum s^2 + x^2 - \frac{(\sum s + x)^2}{|R_i|+1}}{|R_i| + 1} \\ &= \frac{\frac{|R_i|x^2}{|R_i|+1} - \frac{2\sum s}{|R_i|+1} + \sum s^2 - \frac{(\sum s)^2}{|R_i|+1}}{|R_i| + 1} \end{aligned} \tag{8}$$

To make sure the new region satisfies the VAR constraint, we have:

$$l \le VAR(R_i') \le u \tag{9}$$

Using (9), we can swiftly determine and store the acceptable variance attribute value range for each region by finding the roots of the quadratic equations $VAR(R_i') = l$ and $VAR(R_i') = u$.

We then delve into three substeps for area verification and assignment: Similar to Substep 2.1 A, Substep 2.1 V uses *seeds* to initialize potential regions. If $l \le 0$, each seed forms its own region, since a single area has zero variance. If $l > 0$, seed areas are iteratively merged with neighbors to adjust regional variance toward $[l, r]$. During this addition, preference is given to seed areas that satisfy a constraint that the current temporary region fails to meet. In all other instances, non-seed areas are prioritized to maintain the number of potential regions. The algorithm continues expanding region R until it satisfies constraint c. Following this, R is added to the region list P. Should the neighbors of R be insufficient to form a valid region, the entire operation is reversed, leaving the areas of R unassigned.

Substeps 2.2 V and 2.3 V mirror the procedures of Substeps 2.2 A and 2.3 A respectively. In 2.2 V, unassigned areas are added to neighboring regions if their VAR values fall within the region's acceptable range. In 2.3 V, regions lacking seeds for all constraints are merged with neighbors. Since merging two VAR-compliant regions may still violate the constraint due to differing means, a merge limit is imposed. This allows multiple merging attempts (per (7)) to reach feasibility. A similar heuristic is used when area reallocation or merging violates VAR. If infeasible regions remain, they are removed and their areas gradually reassigned to neighbors based on the acceptable range defined in (9).

Complexity analysis. The approach to compute AVG and VAR shares procedural similarities, and any single query can accommodate only one constraint at a time. Since their differing validation steps take constant time, both constraints have the same complexity. We thus analyze the complexity of the three main substeps generally. Consequently, we refer to Substep 2.1 A and Substep 2.1 V collectively as Substep 2.1.

The same consolidation applies to Substeps 2.2 and 2.3. The Substep 2.1 and the first round of the Substep 2.2 iterate through all seed areas and each takes $O(n)$ time. The second round of 2.2 iterates over unassigned areas, removing at least one per iteration or terminating early. The number of unassigned areas is $\leq n$, hence the worst-case total number of operations is $\sum_{i=1}^{n} i = O(n^2)$. Merges are bounded in number, and each takes constant time. Substep 2.3 iterates through all regions once, thus taking maximum of $O(n)$ time. Thus, Step 2 time complexity is $O(n) + O(n) + O(n^2) + O(n) = O(n^2)$.

Remark 2. The time complexity of Region Growing is $O(n^2)$.

Step 3: Monotonic Adjustments: Step 3 enforces SUM and COUNT constraints while preserving previously satisfied ones. This step builds on MP-regions algorithms [56] Since regions are first constructed for MIN, MAX, AVG, or VAR, further adjustments at both area and region levels are required. As SUM and COUNT are monotonic, areas can be added or removed to bring regions within bounds. For each violating region, we first attempt area swaps with neighbors without breaking other constraints. Spatial contiguity is preserved by checking the donor region remains connected after swaps. If swapping fails, regions below bounds are merged, or areas are removed from those above bounds. Finally, if no further changes are possible, infeasible regions are removed and the partition is finalized.

Example: With the input shown in Fig. 6(a) and the example constraints $c_4 = (SUM, s, 12, \infty)$ and $c_5 = (COUNT, s, -\infty, 4)$, the region $R_{green} = \{a_3, a_7\}$ does not satisfy c_4. The boundary area with the neighbor region $R_{blue} = \{a_1, a_4, a_5, a_6\}$ is a_4. Area a_4 is swapped from R_{blue} to R_{green} after ensuring that both regions still satisfy all the other constraints, and R_{blue} is spatially continuous and still above the lower bound of c_4. The result is depicted in Fig. 6(b). After swapping, the receiver region $R_{green} = \{a_4, a_5, a_7\}$ satisfies all the constraints, so the swapping attempts are terminated.

Complexity analysis. Each area is swapped at most once, as it remains valid in the feasible region post-swap. Each swap updates attributes of both donor and receiver regions in $O(m)$ time. Checking donor region connectivity is $O(n)$ in the worst case. As a result, the time complexity of swapping is $O((2m + n) \times n) = O(n^2)$. The area removal of regions that exceed the upper bounds and removing infeasible regions takes one $O(n)$ pass each. Thus, the overall time complexity is $O(n^2) + O(n) + O(n) = O(n^2)$.

Remark 3. The time complexity of Monotonic Adjustments is $O(n^2)$.

C. Local Search Phase

The partition with the largest number of regions p is passed to the local search to reduce heterogeneity. This phase employs Tabu search [24], a meta-heuristic, to minimize heterogeneity. This aligns with the two-phase regionalization design [10], [48], which separates construction and refinement for better tractability and quality. In brief, Tabu search iteratively moves areas between neighboring regions while preserving all constraints. It starts from the feasible partition from construction and explores the best neighboring solution. Worse solution may be accepted to escape local optima. Moves are recorded in a tabu list with fixed tenure to prevent cycling. When a move leads to a partition better than the best partition so far, the algorithm chooses this move even if the tabu list prohibits it. The search ends after a fixed number of steps without improvement, returning the best partition. This phase preserves the number of regions p while improving heterogeneity.

Complexity analysis. As an incomplete algorithm, Tabu search's complexity is approximated based on its parameters and neighborhood computation. It allows up to n non-improving steps per cycle, resetting on improvement, resulting in a worst-case of $O(\alpha n)$ iterations, where α is the number of improving moves. Each iteration updates valid moves via region connectivity checks on boundary areas, costing $O(n \times n)$ times. A move attempt takes $O(n)$ to compute the pair-wise dissimilarity in the worst case. So, the overall time complexity is $O(\alpha n \times (n^2 + n)) = O(\alpha n^3)$. In practice, most improving moves occur early, so the actual number of iterations is often much less than $2n$.

Remark 4. The time complexity of the Local Search phase is $O(\alpha n^3)$, where α is the number of valid moves that improve the heterogeneity over the existing optimum.

D. Handling Arbitrary Sets of Constraints

While previous discussion assumes all constraint types, queries often contain only a subset. For example, a query could have only one AVG constraint, or one MIN constraint and one COUNT constraint. This section outlines how *FaCT* handles such arbitrary subset. *FaCT* treats missing constraints as having an infinite range $(-\infty, \infty)$, impacting both feasibility and construction. Without MIN/MAX, feasibility skips filtering, and Step 1 selects all areas as seed areas. If the AVG constraint is missing, the Region Growing step adds all areas in *seeds* to *unassigned_avg* and then initializes single-area regions. Areas not in *seeds* are then added to *unassigned_avg* and merged to neighbor regions iteratively. If SUM/COUNT constraints are absent, Step 3 of the construction phase is omitted. Thus, *FaCT*'s modular design enables flexible handling of any constraint subset.

VI. Complexity Analysis

This section analyzes *FaCT*'s time and space complexity. According to Remarks 1, 2, and 3 in Section V-B, the time complexity of the feasibility and the three steps of the construction phases are $O(n)$, $O(n^2)$, and $O(n^2)$, respectively. Thus, the total time complexity of both phases is $O(n^2)$. According to Remark 4 in Section V-C, the time complexity of the Local Search phase is $O(\alpha n^3)$. Therefore, *FaCT*'s total worst-case time complexity is $O(\alpha n^3)$, where α is the number of improving moves in local search. For space, storing n areas with m attributes (from m constraints) takes $O(m \times n)$ space. Seed selection adds $O(n)$, and region growing holds temporary regions of up to $O(n)$ areas. Each initialized region maintains m attributes and its areas. The regions are disjoint, so the space taken by the region list is bounded by $O(m \times n)$. The Monotonic Adjustments step and the Local Search phase updates the region list in place and does not require additional space. Thus, the overall space complexity is $O(m \times n)$.

VII. Incremental Update and Reuse of Results

Incremental updates are widely used in databases and real-time systems. Adding incremental updates to *FaCT* improves scalability and supports interactive exploration, especially with large datasets or tight constraints. As discussed in Section I, defining region scale is a key challenge, motivating the shifting from p-regions to MP-regions. With more complex IMS constraints, adjusting scale becomes significantly slower. As datasets grow, even efficient $O(n)$ steps like filtering and seeding become costly. However, *FaCT*'s heuristic and selection schemes yield high-quality results per constraint, provided that the constraint remains unchanged. We categorize updates into two types and handle them with different strategies:

Local Incremental Update: Local incremental updates address changes in small region subsets. Typical examples include administrative boundary changes, such as county mergers or splits. In terms of the IMS formulation, there can be a dramatic change in the population of certain areas, which will render the current partitioning obsolete. However, such changes are typically local and confined to neighboring areas. Inspired by the Iterated Greedy algorithm [57], we adopt a deconstruction-reconstruction strategy. To be more specific, the incremental update for local changes is divided into three steps: (1) Identify affected regions and their one-hop neighbors to form connected components. (2) Deconstruct these components while leaving other regions unchanged. (3) Reconstruct and merge updated regions with the unchanged partition. This step includes reassigning leftover areas and validating the region list. The partition statistics are reported to inform users if drastic changes affect update quality. As demonstrated in Section VI, the time complexity of the *FaCT* algorithm stands at $O(\alpha n^3)$. Consequently, *FaCT*'s performance is markedly enhanced when the problem size is reduced by the local incremental update.

Global Incremental Update: This relates to significant alterations to one or multiple constraints. These include adjusting thresholds or switching constraint attributes, impacting all regions. Because of *FaCT*'s modular design, intermediate results can be reused when only part of the constraints or attribute values change. For example, if constraints include MIN, AVG, and SUM, users may later adjust SUM's threshold or attribute (e.g., from total to employed population). Step 2 results can be reused in such cases. Since AVG computations are often a bottleneck, incremental updates can significantly reduce computation, especially in online settings. *FaCT* supports global updates via three steps: (1) Identify changed constraints by hashing their definitions and attribute lists. (2) The results that can be reused are determined by the unchanged constraints. (3) The partition is incrementally updated by only recomputing the required constraints. The performance comparisons with full recomputation are provided in Section IX-G.

Algorithm 2: Iterated Greedy Framework.

```
  Input: Area set $A$, constraint set $C$, disturbance
         intensity $D$
  Output: $partition$
1 $P^* = Construct(A, C)$;
2 while not (termination condition) do
3     $P_{dec}, A_{dec} = Deconstruct(P^*, D)$;
4     $P_{rec} = Reconstruct(P_{dec}, A_{dec})$;
5     $P^* = AcceptanceCriteria(P^*, P_{rec})$;
6 end
7 return $P^*$;
```

VIII. Iterated Greedy

As discussed in the previous section, local incremental updates use deconstruction and reconstruction inspired by the Iterated Greedy (IG) algorithm. IG offers strong performance despite its simplicity and is widely used in many domains [57], [58], [59], [60]. In this section, we further extend this idea and explore the potential of utilizing IG to improve the result quality.

Algorithm 2 gives a high-level overview of the IG framework with *FaCT* construction (*FaCT-IG*) for addressing the IMS problem. *FaCT-IG* starts with an initial solution, which we obtain through the construction phase as detailed in Section V-B (line 1). Then it goes through iterations of partial deconstructions (line 3) and reconstructions (line 4).

Deconstruction: The deconstruction phase deconstructs the feasible solution P^* according to the disturbance intensity D. $D \in [0, 1]$ determines to what extent the algorithm partially deconstructs the P^*. In the IMS problem, we choose $\lceil D * |P^*| \rceil$ regions for deconstruction. We iteratively select areas at random and include their regions and one-hop neighbors in a candidate set, reinforcing the greedy criterion. This procedure repeats until the candidate set contains at least $\lceil D * |P^*| \rceil$ regions. Larger regions are more likely to be selected, preserving randomness while favoring size. In return, a larger number of areas provide higher flexibility in reconstructing more regions. Regions in the candidate set and their one-hop neighbors are then deconstructed with their areas returned to set A_{dec}, and the unchanged regions are stored in P_{dec}.

Reconstruction and acceptance criteria: Like the local incremental update (Section VII), the reconstruction phase applies the greedy construction (Section V-B) to A_{dec}. The resulting regions are merged with P_{dec} to form P_{rec}. To align with the MP-regions strategy, the new result is accepted only if it improves over the current solution. Line 5 compares $|P_{rec}|$ and $|P^*|$, i.e. the p value of the partition after reconstruction and the existing best solution. If $|P_{rec}| > |P^*|$, P_{rec} becomes the new initial solution for the following iterations. It is worth noting that there are multiple ways to define the termination condition (Line 2) and the acceptance criteria (Line 5). For example, we can adapt the simulated annealing algorithm to allow the algorithm to tolerate worse results at a certain probability and terminate when the iteration converges.

TABLE I
DESCRIPTION OF THE MULTI-STATE DATASETS

Name	No. of areas	States
10k	10255	CA, NV, AZ
20k	20570	10k + OR, WA, ID, UT, MT, WY, CO, NM, OK, NE, SD, ND
30k	29887	20k +TX, LA, AR, MO, IA
40k	40214	30k + MN, MS, AL, TN, KY, IL, WI
50k	49943	40k + GA, IN, MI, OH, WV

FaCT-IG is a metaheuristic search algorithm, and we build it upon the *FaCT* heuristic algorithm. As a result, its runtime is not strictly comparable with the *FaCT* algorithm and the MP-regions problem due to the difference in framework and the additional parameter tuning procedures. To be fair, we stop *FaCT-IG* if no p improvement occurs for 20 iterations or if it reaches the same iteration limit as MP-regions. Experimental evaluations of the efficiency and the effectiveness of the *FaCT-IG* algorithm are given in Section IX-H.

IX. Experiment Evaluation

This section presents an extensive experimental evaluation of our proposed work to address the IMS problem. The code and data for reproducing the results presented in this section are publicly available [61]. The rest of the section introduces the experimental setup (Section IX-A), studies the impact of different constraint types and threshold ranges on the performance (Section IX-D), shows the scalability of the *FaCT* algorithm (Section IX-F), and summarizes the experimental results (Section IX-I).

A. Experimental Setup

IMS is a novel problem with no direct competitor. Due to the enriched formulation, MP-regions techniques cannot be directly extended to IMS. We evaluate the performance of our method. We also study the impact of different combinations of enriched constraints.

Evaluation datasets: We use nine real datasets that represent the census tracts of the USA, outlined as follows: (1) the Los Angeles City (denoted as 1 k) which includes 1012 areas, (2) the Los Angeles County (denoted as 2 k) which includes 2344 areas, (3) Southern California (denoted as 4 k) as identified by the Southern California Association of Governments (SCAG) [62] that includes 3947 areas, (4) the State of California (denoted as 8 k) which includes 8049 areas, and (5) Five multi-state datasets with $\sim$ 10k to $\sim$ 50k areas, as detailed in Table I. Our default dataset is 2 k. Typical evaluation datasets in the existing literature have between 300-3000 areas [10], [56], so our default dataset size is comparable to the largest datasets in the literature. All datasets are joined with real attributes from the 2010 US census data about facts in each census tract. All datasets share the same three spatial extensive attributes as shown in Table II, one per constraint type, and the dissimilarity attribute is *HOUSEHOLDS*. *POP16UP:* population $age \geq 16$; *EMPLOYED:* employed population; *TOTALPOP:* total population. *HOUSEHOLDS:* household count (used for heterogeneity).

TABLE II
DEFAULT ATTRIBUTES AND RANGES FOR DIFFERENT CONSTRAINTS

Constraint Type	Aggregate	Attribute	Range
Extrema	MIN	POP16UP	$(-\infty, 3000]$
Centrality	AVG	EMPLOYED	[1500,3500]
Counting	SUM	TOTALPOP	$[20000, \infty)$

TABLE III
PERFORMANCE COMPARISON OF METHODS ON THE LA_COUNTY DATASET

LA_County			
Method	**P**	**UA**	**Time (ms)**
GreedyBaseline	2	2334	1401
RandomBaseline	1	1874	96
FaCT	**44**	**161**	**5.366**

The evaluation attributes are selected based on factors that influence the population growth rate [13], which makes the partitions obtained through the experiments useful for studying population growth. The shapefiles of the census tracts and the attribute tables are provided by the US Census Bureau [63] and SCAG [64] and joined using the QGIS software [65].

Experiments run on Java 14 with Intel Xeon W-2123 (3.60 GHz) and 20 GB RAM on Windows 10. Performance is measured by construction and search runtime, result size p, and heterogeneity improvement, defined as the relative reduction in heterogeneity after local search. Our parameters include threshold range and dataset size. Unless mentioned otherwise, the default settings are: dataset $=$ 2 k, random area selection, AVG merge limit $= 3$, Tabu list length $= 10$, and Tabu stopping $=$ dataset size. The default threshold ranges and attributes are shown in Table II for each constraint type. Due to space and similar trends, we report one aggregate per constraint type.

B. FaCT Effectiveness

We first assess the effectiveness of *FaCT* by comparing it against two heuristic baseline methods that represent common strategies for spatially constrained partitioning:

- **GreedyPartition**: This method follows the region-growing strategy from the MP-regions algorithm [10]. After filtering and seeding, it starts from seed areas and greedily adds adjacent areas until all constraints are satisfied.
- **RandomPartition**: This method randomly selects areas filtering and assigns the rest in a constraint-aware manner. If the result is infeasible, it retries.

To better distinguish our method from the baseline, we retain the default constraint settings and tighten the AVG constraint from $[1.5\text{k}, 3.5\text{k}]$ to $[2\text{k}, 4\text{k}]$, informed by the attribute distribution of the 2k dataset. As shown in Table III, FaCT outperforms both baselines, producing 44 valid regions with only 161 unassigned areas in 5.366ms. In contrast, GreedyPartition and RandomPartition yield only 2 and 1 regions, with thousands of unassigned areas. GreedyPartition often stalls with small or fragmented regions, while RandomPartition is fast but typically yields poor-quality results. FaCT's constraint-aware design ensures both high quality and efficiency in generating spatially coherent partitions.

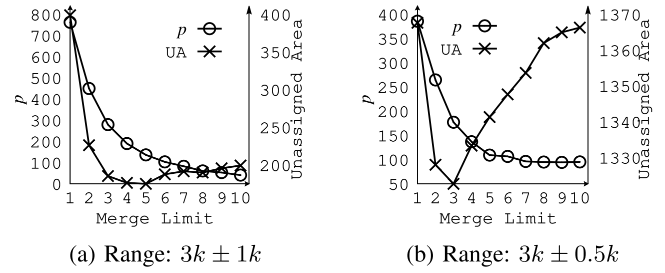


Fig. 7. p and number of unassigned areas with varying merge limit.

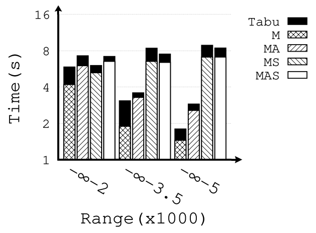


Fig. 10. Runtime for MIN with $l = \infty$.

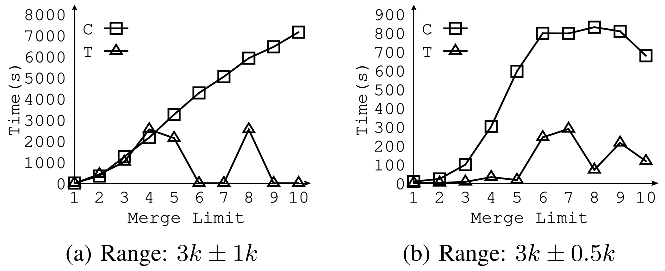


Fig. 8. Construction time and Tabu search time with varying merge limit.

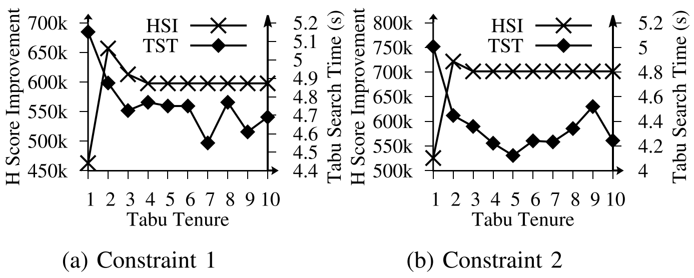


Fig. 9. Impact of Tabu Tenure on Heterogeneity Score improvement and Tabu search time under two constraint settings.

C. Parameter Tuning

This section examines how key parameters affect solution quality, efficiency, and unassigned areas.

1) Merge Limit: This parameter constrains the number of merges in the AVG step. Fig. 7 shows the p value and number of unassigned areas (UA), and Fig. 8 shows the construction time (C) and Tabu search time (T) for two threshold ranges: $3k \pm 1k$ and $3k \pm 0.5k$. Larger merge limits reduce p and increase construction time. These two measures trade off with the number of unassigned areas. For $3k \pm 0.5k$, p improves until the merge limit 3, then declines (Fig. 7(b)). For wider ranges, UA drops until the merge limit 3-5, then rises (Fig. 7(a)). Tabu runtime shows no clear trend, but often grows with the merge limit. For wider thresholds, UA is always low and the merge limit has little effect. Thus, merge limits 2-3 yield minimal UA with good p and runtime, while larger limits sharply increase construction cost.

2) Tabu Tenure: We evaluate how Tabu Tenure affects H Score and search time under two constraint settings (Fig. 9). HSI represents Heterogeneity Score Improvement, while TST denotes Tabu Search Time. Tabu Tenure defines how long a move is banned to avoid cycling.

In Constraint 1 (MIN+AVG), increasing tenure from 1 to 2 boosts H Score and slightly reduces TST (Fig. 9(a)). Further increases yield diminishing returns. In Constraint 2 (SUM only), H Score peaks at tenure 2 and stays stable (Fig. 9(b)). These results suggest that a small tenure value is sufficient to enhance solution quality without incurring excessive search time, as the solution space is limited due to the constraints imposed.

D. Impact of Constraints Types

This section studies the impact of different sets of constraints on regionalization performance. Sets of constraints vary in terms of: (1) set size, i.e., number of constraints, (2) types of constraints, and (3) threshold ranges. Sections IX-D1, IX-D2 and IX-D4 evaluate sets that include MIN, AVG, and SUM constraints, respectively. MIN constraints are denoted as *M*, combinations of MIN and AVG constraints are denoted as *MA*, combinations of MIN and SUM constraints are denoted as *MS*, combinations of MIN, AVG, and SUM constraints are denoted as *MAS*. Time of local search is denoted as *Tabu*.

1) MIN Constraint: This section evaluates regionalization queries with MIN constraints, $c = (MIN, s, l, u)$, in combination with other constraint types. MIN and MAX constraints perform two roles: filtering infeasible areas and selecting seed areas, and both depend on the range threshold $[l, u]$. Hence, we explore three cases: (a) ranges with $l = -\infty$, which shows the impact of u values on seed area selection, (b) ranges with $u = \infty$, which shows the impact of l values on infeasible area filtration, and (c) ranges with bounded values of l and u that show their simultaneous impact.

Fig. 10 shows that runtime varies by constraint type. Table IV gives the number of regions p for different constraint combinations. Using only the MIN constraint yields the highest p, bounded by the number of seed areas. Larger u yields more seeds, fewer iterations, and lower runtime (Fig. 10). Adding constraints leads to fewer regions, as each may include multiple seed areas. With more seed areas, the initialized region becomes smaller, and the SUM constraint takes more operations to satisfy. This causes the runtime of MS and MAS to increase by a small amount, although the time for MIN and AVG decreases. Increasing u leads to an increase in the heterogeneity improvement from 6.96% at $u = 2k$ to 40.2% at $u = 5k$ due to higher p.

Ranges with $u = \infty$. Table IV shows p values and Fig. 11 gives the construction time and local search time for different

TABLE IV
p VALUES FOR DIFFERENT THRESHOLD RANGES FOR MIN CONSTRAINT COMBINATIONS

	Ranges with l = -∞			Ranges with u = ∞			Ranges with bounded u and l							
	(-∞,2k]	(-∞,3.5k]	(-∞,5k]	[2k,∞)	[3.5k,∞)	[5k,∞)	[2.5k,3.5k]	[2k,4k]	[1.5k,4.5k]	[1k,5k]	[1k,2k]	[2k,3k]	[3k,4k]	[4k,5k]
M	270	1447	2172	2074	898	173	834	1501	1895	2109	207	789	712	401
MS	184	354	365	342	142	12	281	334	356	362	159	307	215	78
MA	208	1037	1483	1593	812	97	776	1206	1398	1496	175	654	704	386
MAS	170	335	341	337	145	7	275	328	339	344	152	304	215	74

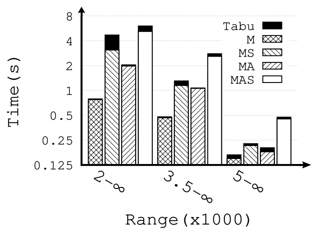
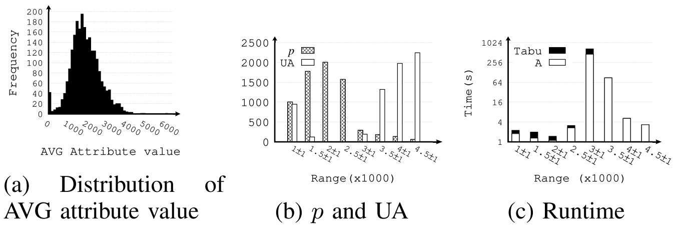
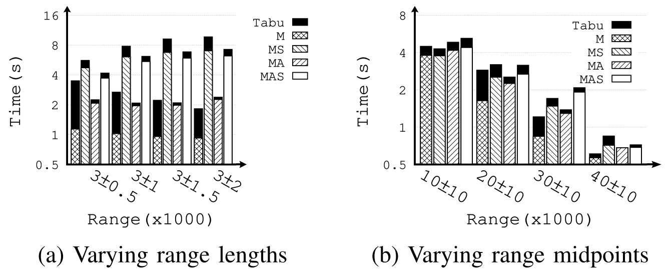


Fig. 11. Runtime for MIN with $u = \infty$.



(a) Distribution of AVG attribute value (b) p and UA (c) Runtime

Fig. 13. Impact of AVG constraint under different configurations: (a) value distribution, (b) p and UA for various midpoints, and (c) runtime comparison.



(a) Varying range lengths (b) Varying range midpoints

Fig. 12. Runtime for MIN with bounded l and u.

constraint combinations. Raising lower bound l filters more areas, leaving the rest scattered. This reduces p and significantly lowers runtime. The heterogeneity score improvement reduces as l increases with up to 17.6% due to decreasing p that narrows the search space of local search.

Ranges with bounded l and u: We explore bounded ranges that vary range length while fixing range midpoint. Table IV gives the p values and Fig. 12(a) shows runtime. When the range length increases, the p value increases because fewer areas are filtered out and more regions are initialized. As a result, both total and construction time increase due to a larger search space. However, there is no clear trend for the Tabu search time.

With a fixed range length and a higher midpoint, both construction and local search times decrease (Fig. 12(b)). At higher values, areas split into smaller components, but seed count remains stable. Thus, Steps 2 and 3 finish faster, reducing runtime. The p value depends on the correlation between attributes of MIN and AVG constraints. If not correlated, a higher midpoint raises the MIN lower bound, filtering more areas while the seed count stays the same if we assume random attribute distribution and constant range size. This shrinks the initialization space, thereby lowering p. However, real attributes may be correlated, such as *POP16UP* and *EMPLOYED*. Raising MIN from $[1k, 2k]$ to $[2k, 3k]$ filters extreme AVG values and increases AVG-compatible seed areas. Thus, when attributes are correlated, raising the constraint midpoint can either increase or decrease p, depending on the direction and strength of the correlation. In all cases, the heterogeneity improvement increases when p increases, by up to 11.3% in this case.

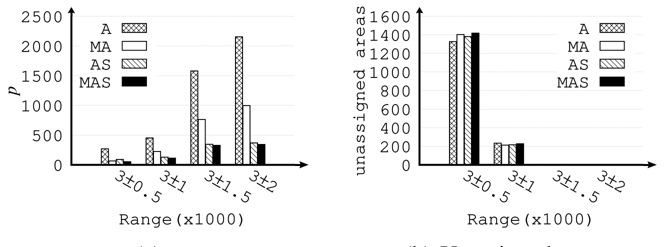
2) AVG Constraint: The average aggregate function is non-monotonic. Satisfying the constraint can be computationally heavy with certain ranges. To illustrate this, we explore two cases: (a) ranges with fixed length and changing midpoints, and (b) ranges with a fixed midpoint but varying lengths.

Ranges with fixed range length. We fix the AVG range length to $2k$ and vary the midpoint from $1k$ to $4.5k$ in $0.5k$ steps. We focus on the performance of the AVG constraint to exclude the impact of other constraints. Fig. 13(a) shows the distribution of the AVG attribute value of the ares in the default dataset, which is a positively skewed distribution. Most values are below $4k$, with outliers reaching up to 6149. Fig. 13(b) shows p and unassigned areas; Fig. 13(c) shows runtime. For midpoints between $1k$ and $2.5k$, about half of the areas fall within range, and combining others is relatively easy. Unassigned areas drop to zero at midpoints $2k$ and $2.5k$, with runtime under $3.2s$. The ranges are also flexible for the Tabu search phase, with the heterogeneity score improved by up to 30.1% . At $3k \pm 1k$, most areas fall below l, but combining with high-value outliers is still possible. In addition, as an area is assigned, the newly formed region may lead to the possibility of assigning the neighboring areas. Hence, the enclave assignment process continues for multiple iterations until no further updates can be made. Thus, runtime increases, but unassigned areas drop significantly to 9% Above $3.5k$, most areas fall below l, making region formation difficult. Most areas remain unassigned, and construction time shortens due to early termination. Midpoints above $3k$ also tighten the AVG constraint, which limits feasible Tabu moves. As a result, the Tabu search time is negligible and the heterogeneity improvements are below 0.4% .

Ranges with fixed midpoint: Following the prior experiment, we fix the midpoint at $3k$, the most challenging case, and vary the range length. We include other constraints to show how AVG can create performance bottlenecks. The p values for different

TABLE V
p VALUES FOR DIFFERENT THRESHOLD RANGES FOR VAR CONSTRAINT COMBINATIONS

	Ranges with l = -∞				Ranges with u = ∞				Ranges with bounded u and l					
	(-∞,400k]	(-∞,500k]	(-∞,600k]	(-∞,700k]	[300k,∞)	[400k,∞)	[500k,∞)	[600k,∞)	[400k,700k]	[450k,650k]	[500k,600k]	[400k,500k]	[500k,600k]	[600k,700k]
V	2344	2344	2344	2344	393	290	218	175	268	224	181	226	181	155
MV	1059	1059	1059	1059	365	283	205	167	247	213	178	206	176	147
VS	189	245	278	308	267	207	160	129	171	150	123	128	122	102
MVS	161	209	236	267	260	203	161	125	177	137	119	129	121	104



(a) p (b) Unassigned areas

Fig. 14. Impact of different AVG range lengths

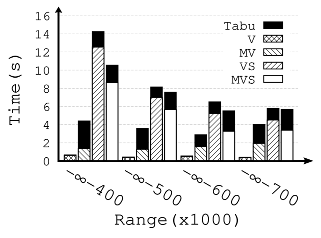


Fig. 15. Runtime for VAR with $l = \infty$.

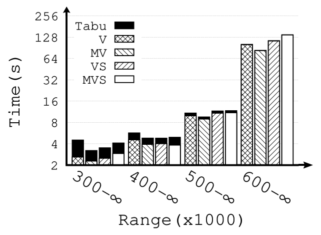


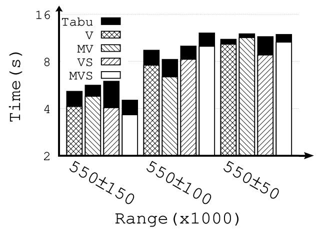
Fig. 16. Runtime for VAR with $u = \infty$.

constraint combinations are given in Fig. 14(a), and the runtime is shown in Fig. 18. The figures show increasing p values with longer ranges, but the runtime depends on the range length. Range length heavily affects construction time, while different constraint combinations with the same range have less impact. This is due to AVG dominating the construction phase runtime. At $3k \pm 0.5k$, the tight range causes early termination due to no feasible enclave assignments. However, 60% of the areas remain unassigned, as shown in Fig. 14(b). Like before, the range $3k \pm 1k$ is the hardest and dominates the total runtime. However, the reduction in the number of unassigned areas is also significant for all constraint combinations. Wider ranges cover more areas, making enclave assignment easier without merging. Thus, the time is much shorter, and all areas are assigned as well. The final heterogeneity score also reduces when the range is enlarged due to the reduction in region sizes. Wider ranges boost heterogeneity improvement up to 21.8% by enabling more valid local search moves. These results show AVG's sensitivity to range selection. Very narrow ranges ($\pm 0.5k$) leave many areas unassigned and limit region formation. Overly wide ranges ($\pm 2k$) make most areas valid, reducing constraint effectiveness and forming large, less meaningful regions. The results suggest that intermediate range settings, such as those centered around the mean with a span approximating one standard deviation, achieve a more favorable trade-off. They reduce unassigned areas while preserving constraint pressure for meaningful partitions. This highlights the need to align constraints with data statistics to balance efficiency and output quality.

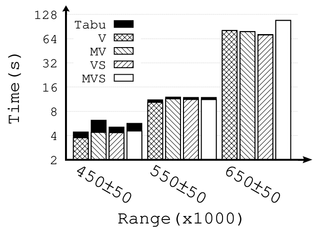
3) VAR Constraint: The variance constraint is a non-monotonic constraint that regulates the statistical dispersion of regions. By adjusting the threshold values, users can guide how clustered or dispersed attribute values become. We study its impact in three scenarios: (a) ranges with $l = -\infty$, which regulates the VAR attribute from being scattered, (b) ranges with $u = \infty$, which defines how clustered the VAR attribute is allowed to be, and (c) ranges with bounded values of l and u that show their joint impact on defining the diversity of the regions in terms of the VAR attribute values. We set thresholds based on the *EMPLOYED* attribute's overall variance ($550k$), as sample variance approximates population variance. We use V to represent VAR, and combine it with other constraints using the same notation as before (e.g., MVS = MIN + VAR + SUM).

Ranges with $l = -\infty$. Fig. 15 gives the runtime of the construction phase when the u changes from $400k$ to $700k$ and the corresponding p values are detailed in Table V. As u increases, the runtime drops and the p value increases for all constraint combinations. This happens because a higher u allows more scattered regions, making construction easier. Unassigned areas also drop from ˜50% to ˜15%, which also corresponds to the reduction in construction difficulty. Heterogeneity improves by 7.4% for MV and VS, but only slightly for MVS.

Ranges with $u = \infty$*:* Fig. 16 shows runtime and Table V gives the p value. Runtime increases but remains manageable until l approaches the dataset's variance. When $l \geq 600k$, the VAR heuristic from Section V-B activates to handle the strict constraint. When the attempt limit is set to 5% of total areas, runtime grows from $19.3s$ to $137.6s$ However, the p value is improved from 104 to 175 while reducing the percentage of unassigned areas from 47.3% to 25.7% . Further increasing the limit yields little benefit to result quality. The ideal limit depends on the dataset and attribute; lowering it can speed up runtime at the cost of quality. The Tabu search time is negligible because of the difficulty of finding a feasible move given the stringent VAR constraint. The heuristic score improvement decreases from 4.5% to 0.08% as l increases.



(a) Varying range lengths



(b) Varying range midpoints

Fig. 17. Runtime for VAR with bounded l and u.

TABLE VI
p VALUES FOR DIFFERENT THRESHOLD RANGES FOR SUM CONSTRAINT COMBINATIONS

	Ranges with u = ∞					Ranges with different lengths		
	[1k,∞)	[10k,∞)	[20k,∞)	[30k,∞)	[40k,∞)	[15k, 25k]	[10k,30k]	[5k,35k]
MP	2298	717	373	245	185	N/A	N/A	N/A
S	2298	720	371	245	186	489	714	1358
MS	1056	581	342	237	179	408	567	785
AS	1545	642	345	233	177	445	640	1095
MAS	758	518	323	226	172	370	497	631

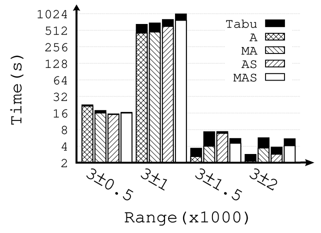


Fig. 18. Runtime for AVG with different range lengths.

Ranges with bounded l and u: Fig. 17(a) shows the runtime for varying range lengths while Fig. 17(b) shows results for shifting midpoints. Trends are similar to the case with unbounded u, as shown in Table V and the two figures. Narrower ranges or higher midpoints lead to longer construction time, especially for $[600k, 700k]$. The number of unassigned areas also increases from 8.3% to 25.8% as the length of the range narrows. V and MV witness an increase in unassigned areas when the midpoint increases, from 12.1% to 26.1% . In VS and MVS, unassigned areas stay at ˜25% due to limited attempts to satisfy both SUM and VAR constraints. The heterogeneity increase is below 1% for all constraints in both cases.

4) SUM Constraint: SUM constraints are the only type of constraints that are used in existing state-of-the-art solutions for the max-p regions (MP-regions) problem. We compare with them, denoted as MP, using a single SUM constraint with an open upper bound, i.e.,$[l, \infty)$. Table VI shows p values; Figs. 19 and 20 show runtime under SUM combinations. With $u = \infty$, higher l lower p, while runtime remains stable for both methods. *FaCT* matches MP-regions in p value under identical constraints (Table VI). But *FaCT* runs in under half the time of MP-regions when $u = 30k$ or $40k$. In addition, the *FaCT* algorithm handles generic types of constraints, which are not supported by the competitor. As l increase, heterogeneity improves-up to 72.6% for MP and S, 13.8% for MS, and less than 1% for AS and

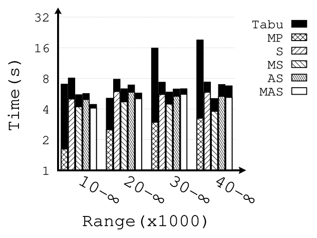


Fig. 19. Runtime for SUM with $u = \infty$.

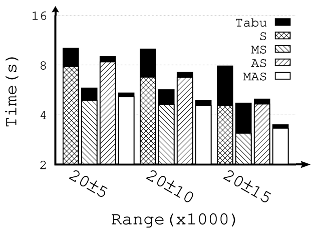


Fig. 20. Runtime for SUM with a changing range length.

TABLE VII
IMPACT OF SPATIALLY EXTENSIVE ATTRIBUTE SELECTION

Attribute	p	H(P) (Before)	H(P) (After)	UA	Time (ms)
TOTALPOP	93	3,284,252	3,159,660	1444	0.016
HOUSEHOULDS	10	36,294,650	36,294,650	1804	0.358
EMPLOYED	49	955,855,969	948,865,529	182	5.046
POP16UP	317	5,208,466	4,676,319	54	0.057

MAS. Larger regions enable more local search moves, boosting heterogeneity. For bounded threshold ranges, in Table VI and Fig. 20, both p values and construction runtime decrease when the lower bound l increases and the trend is similar to $u = \infty$. Heterogeneity improves with range length: up to 30.4% (S), 6.9% (MS), 3.76% (AS), 2.49% (MAS). However, the unassigned areas could reach up to 25.1% when u is bounded for MS, AS, and MAS. This is because areas are removed so that each region does not exceed u. *FaCT* reports output stats to help users adjust threshold ranges effectively.

E. Impact of Extensive Attribute Selection

We tested four spatially extensive attributes in an AVG constraint, keeping other settings unchanged. Table VII shows that attribute choice greatly affects region count, heterogeneity, and unassigned areas. For example, *POP16UP* gave 317 low-heterogeneity regions, but *HOUSEHOLDS* yielded only 10 high-heterogeneity ones, which highlight the role of attribute distribution. Thus, choosing the right attribute is key to meaningful and reliable results.

F. FaCT Scalability

This section evaluates *FaCT*'s scalability. We tested various constraint combinations on nine datasets using default thresholds. The dataset size of the 50k dataset is 17 times the largest dataset used in the existing literature of regionalization[10], [56]. As the AVG constraint is identified as a clear performance bottleneck when the range is set to be $3k \pm 1k$, our discussion

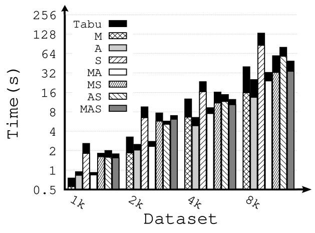


Fig. 21. Runtime varying datasets 1k to 4k.

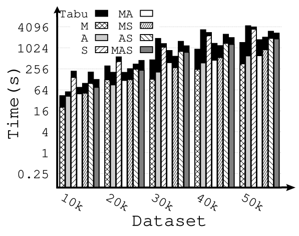


Fig. 22. Runtime varying datasets for 10k-50k.

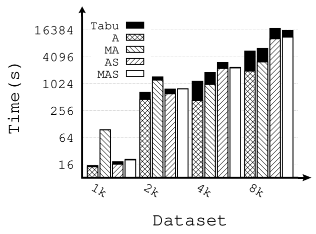


Fig. 23. Runtime varying datasets for AVG constraint with range $3k \pm 1k$.

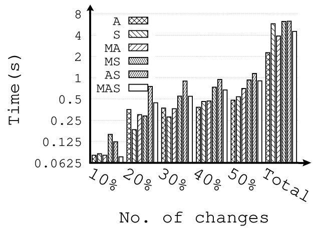


Fig. 24. Runtime for local incremental update.

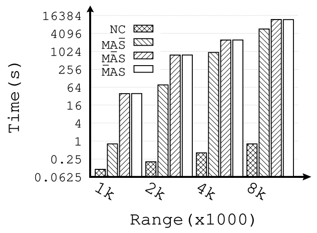


Fig. 25. Runtime for global incremental update when AVG is the bottleneck.

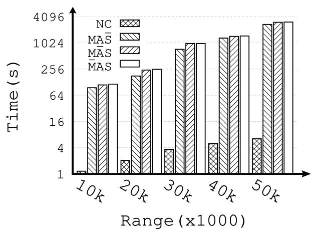


Fig. 26. Runtime for global incremental update for 10k-50 k.

contains two parts. First, we show results with AVG at its default range (Fig. 21, Fig. 22) to assess scalability under normal settings, then examine AVG with a range of $3k \pm 1k$ (Fig. 23) to study scalability in more extreme cases. Figs. 21, 22 show linear runtime growth for M and quadratic for others when AVG isn't a bottleneck. For all datasets, *FaCT* provides a very acceptable runtime for regionalization applications.

Fig. 23 shows that AVG $(3k \pm 1k)$ greatly increases consturction time. As input size grows, runtime increases faster than in normal settings, showing reduced scalability for AVG. However, runtime doesn't always grow with dataset size. For example, the 4k dataset runs faster than 2k, except in AS and MAS. This difference is caused by the merging procedure in the AVG constraint. Generally, more areas are easier to aggregate in regions that satisfy the AVG constraint. In all cases, the construction time scales much better than the Tabu time, and it still provides a solution with almost the same quality.

G. Incremental Update Performance

This section studies the efficiency and effectiveness of the incremental update of queries.

Fig. 24 gives the runtime of the local incremental update with ˜10% to ˜50% regions updated and the comparison with a total reconstruction. Updates are triggered by randomly altering area attributes from 0% to 200% of their original values. The performance of the local incremental update is promising. For the slowest case (AS), updating 10% takes just $0.124s$, and 50% takes $1.146s$ over 100 iterations. This is much faster than $6.301s$ needed for full recomputation. For MP-regions (S), updating 10% to 50% takes only 1.45% to 9.28% of the full time, while preserving the result quality. This update method reconstructs only nearby areas, reducing input size and runtime. Across all datasets, local updates take $< 1s$ and maintain high p values and low unassigned areas. Thus, *FaCT* efficiently handles small local changes using incremental updates.

Secondly, we discuss the performance of the global incremental update. We focus on two scenarios: identified bottlenecks and large datasets. Since only construction changes in global updates, we omit tabu search times. Fig. 25 shows the runtime of 2k-8 k datasets with AVG bottlenecking the construction while Fig. 26 depicts the runtime for large datasets with size $10k$ to $50k$ areas. In both figures, the hat (e.g., $MA\bar{S}$) marks the changed constraint. Thresholds are slightly adjusted to keep constraint difficulty consistent. *NC* denotes no change; the algorithm reloads the previous results. According to Fig. 25, validating the constraints and reloading the previous result is efficient, taking less than 1 s for all the datasets. $\bar{M}AS$ changes filtering/seeding, requiring full recomputation. Thus, $\bar{M}AS$ has the same runtime as *MAS* (see Fig. 18). However, when the algorithm manages to avoid recomputing the AVG constraint by only incremental

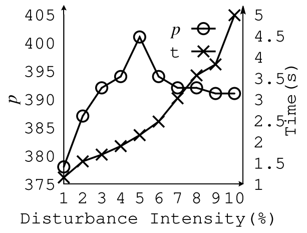


Fig. 27. Effect of disturbance intensity for S.

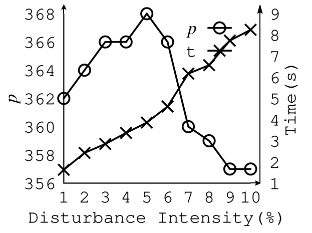


Fig. 28. Effect of disturbance intensity for MAS

TABLE VIII
p VALUES, CONSTRUCTION TIME, AND NO. OF UNASSIGNED AREAS FOR THE *FACT* CONSTRUCTION AND *FACT-IG* CONSTRUCTION

	MP	S	S-IG	MA	MA-IG	MS	MS-IG	AS	AS-IG	MAS	MAS-IG
p	373	371	401	761	844	342	384	345	382	323	370
Time	2.51	5.94	2.15	2.53	2.78	4.48	2.92	5.85	2.62	5.05	3.37
UA	0	0	0	0	16	0	0	1	9	2	19

updating the SUM constraint, the runtime is significantly shorter. Construction time drops 52.6% -97.9% (8 k to 2 k datasets). This avoids the AVG bottleneck noted in Section IX-D2. Fig. 26 shows the runtime for large datasets with no dominant constraint. Reloading unchanged results remains fast even for large datasets. Time savings range from tens $M\bar{A}S$ to hundreds of seconds $MA\bar{S}$, though the ratio is smaller. Result quality (p value and unassigned areas) remains unchanged.

H. Iterated Greedy Performance

This section evaluates *FaCT-IG* performance in improving IMS results compared to standard *FaCT* The comparison focuses on the construction phase using IG We first identify the best disturbance intensity for effective yet efficient IG updates. Experiments show a clear link between disturbance intensity, p value, and runtime. Figs. 27 and 28 illustrate this trend for SUM constraint and for all constraint types. The highest p value occurs when disturbance is 4% to 6% . Higher disturbance increases runtime due to more deconstructed regions and larger input sizes. Above 10%, p value drops while runtime continues to rise for all constraints. With optimal disturbance set, we compare p value, runtime, and unassigned areas. Each setting is tested 10 times, and results are averaged (Table VIII). The results that are obtained by *FaCT-IG* are marked with "-IG" postfix. For instance, S is from *FaCT*, while S-IG is the improved version from *FaCT-IG*. MP refers to the baseline MP-regions algorithm for SUM constraints. Overall, *FaCT-IG* achieves higher p values and is often more efficient. For MP-regions, *FaCT-IG* produces 401 regions, compared to 373 from MP-regions after 100 runs. Since all areas are assigned and p is higher, *FaCT-IG* is strictly better. *FaCT-IG* is faster since it only refines one initial solution instead of repeating full constructions. The difference in the input size of each iteration reduces the runtime drastically. Similarly, the p value of *MS* is also improved from 342 to 384 without sacrificing the number of unassigned areas. However, the result is more controversial for the other constraint combinations involving AVG. MA, AS, and MAS see higher p values but also more unassigned areas. Because the construction of the feasible solution is designed according to the mathematical property of the constraints and enclaves are assigned step by step, merging partial IG results makes them hard to reassign. For monotonic constraints like MP-regions, this isn't an issue. But for others, more unassigned areas may outweigh p and runtime gains.

I. Summary of Results

All experiments demonstrate that the *FaCT* algorithm efficiently generates high-quality, feasible solutions within seconds in most cases. It scales well to large datasets and adapts effectively when more areas are filtered or fewer seed areas are available. While the impact of the centrality constraint varies with its range and attribute distribution, performance remains reasonable overall. Counting constraints allow the MP-regions problem to be treated as a special case of IMS, maintaining both scalability and quality. The local and global incremental update mechanisms offer flexibility for interactive refinement. Effectiveness tests show that *FaCT* produces compact, feasible region structures, validating the strength of its optimization. *FaCT-IG* further enhances results by applying the IG algorithm, outperforming existing methods and demonstrating potential for broader metaheuristic integration.

X. CONCLUSION

This paper introduces an incremental max-p regionalization with statistical constraints (IMS) problem that extends the existing regionalization problems with statistical user-defined constraints. The IMS problem clusters a set of spatial areas into homogeneous regions that satisfy the user-defined constraints. The IMS problem enables enriched types of constraints that support SQL-inspired aggregate functions, MIN, MAX, AVG, VAR, SUM, and COUNT, with range operators. We prove the NP-hardness of the IMS problem. To tackle this problem, we propose *FaCT*; a three-phase algorithm that finds an approximate solution with a maximum number of regions and minimum overall heterogeneity. The first phase checks the feasibility of finding a solution given the input constraints. It also provides users with insightful information to tune their input and enable flexible exploration of various datasets. The second phase constructs an initial solution, and the third phase further optimizes it to provide a final solution. We have empirically demonstrated the capability of *FaCT* to provide efficient and effective performance on various real datasets. *FaCT* supports local and global incremental update to support exploratory analysis and respond to changes in attributes and constraints efficiently. We also combine the

Iterated Greedy metaheuristic algorithm with *FaCT* to further improve the quality of the construction.

In our forthcoming research, we aim to delve deeper into the potential of integrating cutting-edge metaheuristic algorithms to enhance our regionalization analysis. Furthermore, we are keenly interested in exploring multi-objective optimization techniques that can enhance algorithm performance through parallelization.

REFERENCES

[1] R. Benedetti, F. Piersimoni, G. Pignataro, and F. Vidoli, "The identification of spatially constrained homogeneous clusters of COVID-19 transmission in Italy," *Regional Sci. Policy Pract.*, vol. 12, pp. 1169–1187, 2020.

[2] M. P. Armstrong et al., "Decision support for regionalization: A spatial decision support system for regionalizing service delivery systems," *Comput. Environ. Urban Syst.*, vol. 15, pp. 37–53, 1991.

[3] K. S. Cheruvelil, P. A. Soranno, M. T. Bremigan, T. Wagner, and S. L. Martin, "Grouping lakes for water quality assessment and monitoring: The roles of regionalization and spatial scale," *J. Environ. Manage.*, vol. 41, pp. 425–440, 2008.

[4] A. El Kenawy, J. I. López-Moreno, and S. M. Vicente-Serrano, "Summer temperature extremes in northeastern Spain: Spatial regionalization and links to atmospheric circulation (1960–2006)," *IEEE Trans. Autom. Control*, vol. 113, pp. 387–405, 2013.

[5] S. Schönbrodt-Stitt, A. Bosch, T. Behrens, H. Hartmann, X. Shi, and T. Scholten, "Approximation and spatial regionalization of rainfall erosivity based on sparse data in a mountainous catchment of the yangtze river in central China," *J. Environ. Sci. Pollut. Res.*, vol. 20, pp. 6917–6933, 2013.

[6] N. Bullen, G. Moon, and K. Jones, "Defining localities for health planning: A GIS approach," *Social Sci. Med.*, vol. 42, no. 6, pp. 801–816, 1996.

[7] Z. Chen, P. Cheng, L. Chen, X. Lin, and C. Shahabi, "Fair task assignment in spatial crowdsourcing," in *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2479–2492, 2020.

[8] T. Song et al., "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 1009–1020.

[9] D. J. Rossiter and R. J. Johnston, "Program group: The identification of all possible solutions to a constituency-delimitation problem," *Environ. Plan. A*, vol. 13, no. 2, pp. 231–238, 1981.

[10] J. C. Duque, L. Anselin, and S. J. Rey, "The Max-P-regions problem," *J. Regional Sci.*, vol. 52, no. 3, pp. 397–419, 2012.

[11] D. Arribas-Bel and C. R. Schmidt, "Self-organizing maps and the US urban spatial structure," *Eur. Phys. J.*, vol. 40, pp. 362–371, 2013.

[12] J. C. Duque, J. E. Patino, L. A. Ruiz, and J. E. Pardo-Pascual, "Measuring intra-urban poverty using land cover and texture metrics derived from remote sensing data," *Landscape Urban Plan.*, vol. 135, pp. 11–21, 2015.

[13] R. Fragoso, C. Rego, and V. Bushenkov, "Clustering of territorial areas: A multi-criteria districting problem," *J. Quantitative Econ.*, vol. 14, no. 2, pp. 179–198, 2016.

[14] J. E. Patino, J. C. Duque, J. E. Pardo-Pascual, and L. A. Ruiz, "Using remote sensing to assess the relationship between crime and the urban layout," *Appl. Geogr.*, vol. 55, pp. 48–60, 2014.

[15] S. J. Rey and M. L. Sastré-Gutiérrez, "Interregional inequality dynamics in Mexico," *Spatial Econ. Anal.*, vol. 5, pp. 277–298, 2010.

[16] S. E. Spielman and D. C. Folch, "Reducing uncertainty in the american community survey through data-driven regionalization," *PLoS One*, vol. 10, 2015, Art. no. e0115626.

[17] D. A. Stow, C. D. Lippitt, and J. R. Weeks, "Geographic object-based delineation of neighborhoods of Accra, Ghana using QuickBird satellite imagery," *Photogrammetric Eng. Remote Sens.*, vol. 76, pp. 907–914, 2010.

[18] "Build balanced zones (spatial statistics)—ArcGIS Pro," Accessed: May 5, 2025 . [Online]. Available: https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/buildbalancedzones.htm

[19] PySAL Developers, "Max-P regionalization — spopt manual, PySAL spatial optimization module," 2023. [Online]. Available: https://pysal.org/spopt/notebooks/maxp.html

[20] M. Camacho-Collados, F. Liberatore, and J. M. Angulo, "A multi-criteria police districting problem for the efficient and effective design of patrol sector," *Eur. J. Oper. Res.*, vol. 246, no. 2, pp. 674–684, 2015.

[21] Y. Kang, "Modeling the enriched Max-P region problem ASA Mixed-integer programming problem," 2022. [Online]. Available: https://github.com/YunfanKang/FaCT/blob/main/EMP-MIP/EMP_MIP_Formulation.pdf

[22] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2018. [Online]. Available: https://docs.gurobi.com/projects/optimizer/en/current/index.html

[23] Y. Kang, "Solving the MIP formulation of EMP with Gurobi solver," 2022. [Online]. Available: https://github.com/YunfanKang/FaCT/blob/main/EMP-MIP/

[24] F. Glover and M. Laguna, "Tabu Search," in *Handbook of Combinatorial Optimization*. Berlin, Germany: Springer, 1998, pp. 2093–2229.

[25] Y. Kang and A. Magdy, "EMP: Max-P regionalization with enriched constraints," in *Proc. IEEE 38th Int. Conf. Data Eng.*, 2022, pp. 1914–1926.

[26] A. Ferligoj and V. Batagelj, "Clustering with relational constraint," *Psychometrika*, vol. 47, no. 4, pp. 413–426, 1982.

[27] J. Byfuglien and A. Nordgård, "Region-building—a comparison of methods," *Norwegian J. Geogr.*, vol. 27, pp. 127–151, 1973.

[28] L. P. Lefkovitch, "Conditional clustering," *Biometrics*, vol. 36, no. 1, pp. 43–58, 1980.

[29] F. Murtagh, "A survey of algorithms for contiguity-constrained clustering and related problems," *Comput. J.*, vol. 28, no. 1, pp. 82–88, 1985.

[30] A. Gordon, "A survey of constrained classification," *Comput. Statist. Data Anal.*, vol. 21, no. 1, pp. 17–29, 1996.

[31] P. Hansen, B. Jaumard, C. Meyer, B. Simeone, and V. Doring, "Maximum split clustering under connectivity constraints," *J. Classification*, vol. 20, no. 2, pp. 143–180, 2003.

[32] J. C. Duque, R. L. Church, and R. S. Middleton, "The P-regions problem," *Geographical Anal.*, vol. 43, no. 1, pp. 104–126, 2011.

[33] Y. Liu, A. R. Mahmood, A. Magdy, and S. Rey, "PRUC: P-regions with user-defined constraint," in *Proc. VLDB Endowment*, vol. 15, no. 3, pp. 491–503, 2021.

[34] H. Kim, Y. Chun, and K. Kim, "Delimitation of functional regions using a P-regions problem approach," *Int. Regional Sci. Rev.*, vol. 38, pp. 235–263, 2015.

[35] K. Kim, Y. Chun, and H. Kim, "P-functional clusters location problem for detecting spatial clusters with covering approach," *Geographical Anal.*, vol. 49, pp. 101–121, 2017.

[36] X. Ye, B. She, and S. Benya, "Exploring regionalization in the network urban space," *J. Geovisualization Spatial Anal.*, vol. 2, 2018, Art. no. 4.

[37] 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," *Int. J. Geographical Inf. Sci.*, vol. 29, pp. 536–555, 2015.

[38] W. Li, R. L. Church, and M. F. Goodchild, "An extendable heuristic framework to solve the P-compact-Regions problem for urban economic modeling," *Comput. Environ. Urban Syst.*, vol. 43, pp. 1–13, 2014.

[39] W. Li, R. L. Church, and M. F. Goodchild, "The P-compact-regions problem," *Geographical Anal.*, vol. 46, pp. 250–273, 2014.

[40] D. C. Folch and S. E. Spielman, "Identifying regions based on flexible user-defined constraints," *Int. J. Geographical Inf. Sci.*, vol. 28, no. 1, pp. 164–184, 2014.

[41] B. She, J. C. Duque, and X. Ye, "The network-Max-P-regions model," *Int. J. Geographical Inf. Sci.*, vol. 31, pp. 962–981, 2017.

[42] J. H. Danumah et al., "Flood risk assessment and mapping in Abidjan district using multi-criteria analysis (AHP) model and geoinformation techniques,(cote d'ivoire)," *Geoenvironmental Disasters*, vol. 3, no. 1, pp. 1–13, 2016.

[43] T. Vilutienė and E. K. Zavadskas, "The application of multi-criteria analysis to decision support for the facility management of a residential district," *J. Civil Eng. Manage.*, vol. 9, no. 4, pp. 241–252, 2003.

[44] S. Yanık, J. Kalcsics, S. Nickel, and B. Bozkaya, "A multi-period multi-criteria districting problem applied to primary care scheme with gradual assignment," *Int. Trans. Oper. Res.*, vol. 26, no. 5, pp. 1676–1697, 2019.

[45] S. Openshaw, "A regionalisation program for large data sets," *Comput. Appl.*, vol. 3, no. 4, pp. 136–147, 1973.

[46] S. Openshaw, "Classifying and regionalizing census data," *Census Users' Handbook*, pp. 239–270, 1995.

[47] R. M. Assunção, M. C. Neves, G. Câmara, and C. da C. Freitas, "Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees," *Int. J. Geographical Inf. Sci.*, vol. 20, no. 7, pp. 797–811, 2006.

[48] R. S. Garfinkel and G. L. Nemhauser, "Optimal political districting by implicit enumeration techniques," *Manage. Sci.*, vol. 16, no. 8, pp. B–495, 1970.

[49] S. Openshaw, "A geographical solution to scale and aggregation problems in region-building, partitioning and spatial modelling," *Trans. Inst. Brit. Geographers*, vol. 2, pp. 459–472, 1977.

[50] H. Alrashid and A. Magdy, "A scalable unified system for seeding regionalization queries," in *Proc. 18th Int. Symp. Spatial Temporal Data*, 2023, pp. 96–105.

[51] 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 *Proc. 2nd ACM SIGSPATIAL Int. Workshop AI Geographic Knowl. Discov.*, 2018, pp. 33–42.

[52] R. Wei, S. Rey, and E. Knaap, "Efficient regionalization for spatially explicit neighborhood delineation," *Int. J. Geographical Inf. Sci.*, vol. 35, no. 1, pp. 135–151, 2021, doi: 10.1080/13658816.2020.1759806.

[53] A. Buckley, "Understanding statistical data for mapping purposes," 2013. [Online]. Available: https://www.esri.com/about/newsroom/wp-content/uploads/2018/11/mapstatdata.pdf

[54] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory Comput. Syst.*, vol. 39, no. 6, pp. 929–939, 2006.

[55] "NP-hardness of MP-regions," 2021. [Online]. Available: https://cs.ucr.edu/amr/MP_Regionalization_NPHardness.pdf

[56] R. Wei, S. Rey, and E. Knaap, "Efficient regionalization for spatially explicit neighborhood delineation," *Int. J. Geographical Inf. Sci.*, vol. 35, pp. 1–17, 2020.

[57] W. Li, Q. Kang, H. Kong, C. Liu, and Y. Kang, "A novel iterated greedy algorithm for detecting communities in complex network," *Social Netw. Anal. Mining*, vol. 10, pp. 1–17, 2020.

[58] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2033–2049, 2007.

[59] T. Stützle and R. Ruiz, "Iterated greedy," in *Handbook of Heuristics*. Berlin, Germany: Springer, 2018, pp. 547–577.

[60] Q. Kang, H. He, and J. Wei, "An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 73, no. 8, pp. 1106–1115, 2013.

[61] Y. Kang, "Fact," 2021. [Online]. Available: https://github.com/YunfanKang/FaCT

[62] "Southern California Association of Governments - SCAG," 2020. [Online]. Available: https://scag.ca.gov/

[63] U. C. Bureau, "Explore census data," 2021. [Online]. Available: https://data.census.gov/cedsci/

[64] S. C. A. of Governments, "Scag open data portal," 2021 [Online]. Available: https://gisdata-scag.opendata.arcgis.com/

[65] "Welcome to the QGIS project!," 2020. [Online]. Available: https://www.qgis.org/

Yunfan Kang received the bachelor of engineering degree in computer science from the University of Hong Kong, and the PhD degree in computer science and engineering from the University of California, Riverside. He is a postdoc research associate with the CyberGIS Center for Advanced Digital and Spatial Studies, University of Illinois Urbana-Champaign. His research interests include GeoAI, spatial analysis, and spatial networks. He has published more than 20 works in various journals and conferences.

Yiyang Bian received the bachelor of engineering degree in computer science from Central China Normal University, and the MS degree in computer science from Case Western Reserve University. He is currently working toward the PhD degree in computer science with the University of California, Riverside. His work bridges database systems and geographic information science, aiming to enhance spatial data analysis through intelligent indexing, optimization techniques, and machine learning integration.

Qinma Kang received the master's degree in computer science and technology from Xidian University, and the PhD degree in computer software and theory from Tongji University. He is an associate professor with the Department of Computer Science, School of Mechanical, Electrical and Information Engineering, Shandong University. His research interests include data mining, intelligent computing, blockchain, social networks, and machine learning. He has published more than 30 research papers in leading academic journals and international conferences.

Amr Magdy (Senior Member, IEEE) is an associate professor of computer science and engineering with the University of California, Riverside and the current chair of the ACM SIGSPATIAL Recognition Committee. His research interests include spatial Big Data and systems-oriented techniques. His work has appeared in leading venues, such as IEEE ICDE, *IEEE Transactions on Knowledge and Data Engineering*, ACM SIGSPATIAL, VLDB, the *VLDB Journal*, ACM SIGMOD, and ACM TSAS. He has received several research awards, including NSF CAREER, the Google-CAHSI, and Microsoft Research awards in 2023. His papers are shortlisted as best papers of IEEE ICDE'14, ACM SIGSPATIAL'19'23, IEEE MDM'23, and SSTD'23, and shortlisted for ACM SIGSPATIAL 10-Year Impact Award in 2024. He also won the best demo award in ACM SIGSPATIAL'24, and was recognized with the Distinguished PC Member Award in ACM SIGMOD 2024.