EMP: Max-P Regionalization with Enriched Constraints

Yunfan Kang[a, b]
[a]Department of Computer Science and Engineering
[b]Center for Geospatial Sciences
University of California, Riverside
ykang040@ucr.edu

Amr Magdy[a, b]
[a]Department of Computer Science and Engineering
[b]Center for Geospatial Sciences
University of California, Riverside
amr@cs.ucr.edu

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00189

***Abstract*—Spatial regionalization is the process of grouping a set of spatial areas into spatially contiguous and homogeneous regions. This paper introduces an *enriched max-p-regions* (EMP) problem; a regionalization process that allows enriched user-defined constraints based on SQL aggregate functions. In addition to enabling richer constraints, it enables users to employ multiple constraints simultaneously to significantly push the expressiveness and effectiveness of the existing regionalization literature. The EMP 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 greedy approach that finds a feasible set of spatial regions that satisfy EMP constraints while supporting large datasets compared to the existing literature. 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 [7], service delivery systems [3], water quality assessment [13], weather temperature classification [18], rainfall erosivity estimation [46], health data analysis [8], spatial crowdsourcing [12], [48], and constituency allocation [44]. Identifying regions is a computational and conceptual challenge. One of the fundamental challenges is determining the spatial scale of the studied phenomena. For example, *snowing* is spatially correlated at the level of multiple states, e.g., the USA midwest states, while *temperature* is spatially correlated at a county level. Most regionalization routines require the number of regions, p, as an input parameter, e.g., in the USA, $p = 50$ is a state level and $p = 3000$ is a county level. This puts the burden of determining the spatial scale on the data analysts, forcing them to analyze multiple p values, which increases the computational cost and limits the query expressiveness.

The latest regionalization formulation, called the *max-p-regions* problem (or MP-regions), has addressed the scale problem [15] and become popular in different applications, including urban spatial structures [4], measuring intra-urban poverty [17], population growth analysis [21], crime analysis [41], regional inequality analysis [43], spatial uncertainty [49], and neighborhood delineation [50]. With the observation that researchers usually know a condition that a region must satisfy to be suitable for the analysis, the MP-regions automatically discovers the maximum number of regions that satisfy a given user-defined constraint instead of forcing users to input p. However, the MP-regions problem has several limitations in both scalability and expressiveness. Currently, the state-of-the-art algorithms take 3-60 mins 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 [11], [21], [43] to only a few hundred [4], [17], [41], [49], [50]. It is explicitly stated in [49] 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.

Existing MP-regions formulation allows a *single* user-defined *threshold lower bound on one attribute*, e.g., total region population $\geq$ 100K. However, many analysis tasks cannot be catered with a single constraint that involves a simple threshold. In many real applications, it is common to have multiple constraints with different aggregate functions. For example, studies show that the COVID-19 virus transmission pattern is closely related to prosperity and labor mobility [7]. In order to make region-specific recommendations for preventing the virus spread, policymakers can issue a query to identify reasonably populated regions with a total population $\geq$ 200000, an average monthly income between \$3000 to \$5000, and public transportation passengers $\geq$ 10000. Another example is studying the changes in population requires considering multiple factors that influence the phenomenon [21] with different aggregation functions such as the *minimum* population of each area, the *maximum* school drop-out rate, the *average* age of the population, and *total* unemployment. A third example is the patrol sector partition problem [11] that aims to consider the number of calls and the workload balance. These example applications significantly benefit from

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00189

enriching the expressiveness of MP-regions queries that will enable exploring different scenarios and expanding analysis capabilities. Currently, such advanced regionalization queries cannot be expressed using the existing MP-regions formulation due to its limited support for user-defined constraints, limiting its full potential in many real applications.

This paper introduces an enriched MP-regions problem (EMP) that addresses the existing MP-regions limitations. EMP enables users to analyze one to two orders of magnitude larger datasets than the currently supported ones. Furthermore, EMP enriches user-defined constraints with three novel features: (1) EMP provides an enriched set of aggregates that contains three families of aggregate functions, namely, *extrema aggregates*, *centrality aggregates*, and *counting aggregates*. The three families of aggregates are inspired by the standard SQL aggregate functions from which we adopt minimum and maximum as extrema aggregates, average as a centrality aggregate, and summation and count as counting aggregates. (2) EMP allows using a range comparison operator that expresses all possible comparisons, less-than-or-equal ($\leq$), larger-than-or-equal ($\geq$), and range comparison. This enables users to set both lower and upper bounds for their constraints instead of only lower bounds in existing formulations. (3) EMP supports multiple constraints instead of a single constraint. The EMP formulation addresses the limitation of the existing MP-regions formulation and reveals the full potential for regionalization queries in different applications.

The EMP problem is an NP-hard problem (see Section IV), so finding an optimal solution is intractable. We formulated the EMP as a mathematical optimization problem using mixed-integer programming (MIP) problem [28] and solved it with the world's fastest MIP solver Gurobi [25]. The optimizer takes 33.86s to find an optimal solution for 9 areas, 10 hours for 16 areas, and is not able to return a feasible solution for 25 areas after 110 hours of execution [29]. This clearly shows the infeasibility of finding an exact solution for hundreds or thousands of areas in a reasonable runtime. However, even finding an approximate solution faces two main challenges. First, the newly introduced constraints are non-monotonic. Therefore, adding or removing areas to and from a certain region does not provide any guarantee of satisfying or violating these constraints. This requires exhaustive exploration for the whole search space to find the solution, which is intractable with such an NP-hard problem. Second, while satisfying multiple non-monotonic constraints, a region can easily become overlarge. Having oversized regions reduces the number of regions p, which contradicts the objective of maximizing p.

To address these challenges, we introduce $FaCT$, a three-phase algorithm to solve the EMP problem. The first phase derives theoretical bounds to verify the feasibility of finding a solution given the user-defined constraints. In addition, in the case of infeasibility, it enables the user to potentially alter input data to satisfy the constraints. This provides great flexibility to tune either data or query parameters adaptively and enables rich exploratory analysis capabilities for a wide variety of datasets. The second phase constructs a feasible solution that satisfies the input constraints while maximizing the number of regions p. The phase is divided into three independent steps with the following features: (a) Each step is carefully designed to exploit the mathematical properties of one type of constraint to maximize the p to its theoretical upper bound. (b) The input and output of each step are designed exquisitely to reduce the search space for the following steps and mitigate the conflicts introduced by the different mathematical properties of different constraints. (c) The steps are independent of each other, enabling $FaCT$ to construct feasible solutions progressively and handle any arbitrary combinations of constraint types effectively and efficiently. (d) The algorithm provides multiple parameter tuning options so that the users can choose the best aggregate strategy according to the use case and the characteristic of the dataset. (e) The algorithm supports 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 [23] and swaps areas between regions to improve the overall heterogeneity of the regions.

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 enriched max-p regions problem (EMP) that reveals the full potential of max-p regionalization.
- We prove the NP-hardness of the EMP problem.
- We propose $FaCT$; a three-phase greedy algorithm that provides approximate solutions for the EMP 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 VII shows the experimental evaluations and Section VIII 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 [19]. Regionalization is referred to by different names, including region building [10], conditional clustering [33], clustering with relational constraints [19], contiguity-constrained clustering [36], constrained classification [24], maximum split clustering under connectivity constraints [26], and p-regions problem [16] that is used in a variety of applications such as detecting functional regions [30], [31], network-constrained urban management [54], and partitioning compact regions [32], [34], [35]. The max-p-regions problem (MP-regions) [15] 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 [20], [21], [47] 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 [11], [14], [21], [51], [53] use multi-objective optimization to partition the space based on multiple attributes. These techniques still put the burden of determining the spatial scale, i.e., the number of output regions, on the user. They also allow the user to define a single constraint on one of the attributes, limiting the query expressiveness. Our EMP problem addresses these limitations. It enables users to define multiple constraints on different attributes to provide flexible and expressive queries. In addition, it inherits the automatic discovery of the appropriate number of regions from the MP-regions and eliminates the spatial scale problem.

We classify the popular approaches to tackle the NP-hard rationalization problems into two main categories. The first category is the clustering methods [38], [40] 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 [5], [15], [22], [39], [47]. 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 [5], [6] and greedy aggregation [15], [20], [22], [39]. The heuristic search phase optimizes the objective function to improve the region homogeneity [5], [6], [22] or geographical compactness [22] 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 enriched max-p regions (EMP) problem. Let $A = \{a_1, a_2, ..., a_n\}$ denote 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 EMP 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.

EMP allows f to be MIN, MAX, AVG, SUM, or COUNT that compute minimum, maximum, average, summation, and count aggregates, respectively, with the same semantic as the SQL standards. The attribute s is a spatially extensive attribute whose value is divided over the smaller areas when the area is fragmented. For example, the *population* value of a state is divided over its cities, so the *population* is a spatially extensive attribute. This is the opposite to spatially intensive attributes, such as *temperature*, that are not divided when the spatial area is fragmented. 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: $R = \{a_1, a_2, ..., a_g\}$, 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,...,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:

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

This definition of H is popular in the MP-regions literature, so it is recognized by social sciences experts as a primary measure for region heterogeneity. However, our work can support alternative definitions, such as improving spatial compactness or balancing multiple criteria. The reason is that our second phase, which is based on Tabu search as discussed in Section V, can deal with different optimization functions.

EMP Problem. The EMP problem is defined as follows:

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

Output: (1) A set of regions $P = \{R_1, R_2, ..., R_p\}$, where $1 \leq p \leq n$ and each region R_i satisfies the below EMP 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.

EMP 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 $H(P)$.

EMP 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 [15], 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 produce regions that are homogeneous in average income level.

Comparing the EMP problem with the original MP-regions problem [15], there are two major differences that are introduced by the enriched user-defined constraints. First, the enriched constraints are not always monotonic. Assuming that all spatially extensive attribute values are positive, adding an area to a region or removing an area from a region in MP-regions guarantees a monotone change towards satisfying the constraint threshold. This is not the case for the EMP problem due to allowing AVG, MIN, and MAX constraints. In addition, the EMP problem allows upper-bounded threshold ranges, so adding areas to a region without additional validation leads to violating the SUM and COUNT constraints' upper bounds. Second, the EMP problem allows unassigned areas, U_0, to exist in the final area partition, contrary to the MP-regions problem. This provides flexibility to satisfy multiple constraints of different types, especially MIN and MAX constraints that commonly cause invalid areas, as discussed in the following sections. Allowing unassigned areas enables constraints such as MIN and MAX to serve as filters to filter out areas with undesired properties, which introduces more tolerance and enables richer exploration capabilities to regionalization queries.

IV. NP-HARDNESS OF EMP

This section proves the NP-hardness of the EMP problem. The NP-hardness of the EMP problem follows the results that the MP-regions problem is NP-hard [1], [2], [15]. The MP-regions problem takes as input: (1) a set of spatially contiguous areas $A = \{a_1, a_2, ..., a_n\}$. 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 EMP problem as decision problems with a fixed $p = k$.

Theorem 1. *The EMP 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 EMP 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 EMP problem decides on the instance X of the MP-regions problem as well. As the MP-regions problem is NP-hard, then the EMP problem is NP-hard, and the proof is complete. □

V. PROPOSED SOLUTION

This section presents $FaCT$, a three-phase algorithm that finds a feasible solution for an instance of the EMP problem with low overall heterogeneity. The first phase is a ***feasibility phase*** that verifies the existence of any feasible solution given the user-defined constraints. The second phase is a ***construction phase*** that constructs a feasible initial solution with the maximum number of regions p while satisfying all user-defined constraints. This phase represents the major contribution of this algorithm. It balances two challenging objectives: satisfying multiple non-monotonic user-defined constraints and producing a maximum number of 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. The following sections detail each phase.

A. Feasibility Phase

The EMP problem deals with arbitrary combinations of constraints on different attributes. The goal of the feasibility phase is to signal the user at an early stage for: (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 invalid areas for all constraints, including non-monotonic constraints, gives it major practical advantages when employing multiple constraints. The algorithm is not only able to give users a heads-up on the infeasibility of their constraints, but also users can choose to remove input areas that cause such infeasibility automatically. This gives data analysts great flexibility to analyze a wide variety of datasets and constraints.

Given a set of user-defined constraints $C = \{(MIN, s_{min}, l_{min}, u_{min}), (MAX, s_{max}, l_{max}, u_{max}), (AVG, s_{avg}, l_{avg}, u_{avg}), (SUM, s_{sum}, l_{sum}, u_{sum}), (COUNT, s_{count}, l_{count}, u_{count})\}$, 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 $\exists P = \{R_1, R_2, ..., R_p\}$ 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:

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

$$\begin{aligned} AVG(s_{avg}) &= \frac{\sum_{i=1}^{p}(|R_i| \times R_i.AVG)}{n} \\ &\leq \frac{\sum_{i=1}^{p}|R_i| u_{avg}}{n} \;=\; u_{avg} \end{aligned} \tag{5}$$

$$\begin{aligned} AVG(s_{avg}) &= \frac{\sum_{i=1}^{p}(|R_i| \times R_i.AVG)}{n} \\ &\geq \frac{\sum_{i=1}^{p}|R_i| l_{avg}}{n} \;=\; l_{avg} \end{aligned} \tag{6}$$

Equations 5 and 6 $\implies l_{avg} \leq AVG(s_{avg}) \leq u_{avg}$.
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 $\nexists P = \{R_1, R_2, ..., R_p\}$ 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 there is no area that 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 invalid 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 invalid areas.

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

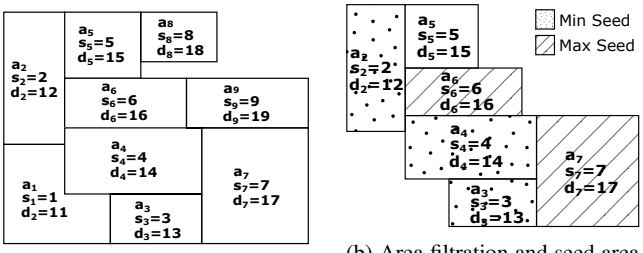
The feasibility phase iterates through the area set and computes the needed attribute aggregates for all constraints in a single pass. This pass is enough to filter out invalid areas, that have $s_{min} < l_{min}$, $s_{max} > u_{max}$, or $s_{sum} > u_{sum}$, to eliminate potential invalid regions. However, the feasibility pass is not enough to filter out invalid areas for the AVG constraint. This is performed during the following construction phase while imposing AVG constraints. All invalid 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 greedily constructs a feasible solution that: (a) satisfies all user-defined constraints, (b) maximizes the number of regions p, and (c) minimizes the number of unassigned areas. This phase faces the challenges of simultaneously satisfying multiple non-monotonic constraints while preventing building oversized regions that contradict maximizing p. To address these challenges, the construction phase satisfies each family of user-defined constraints in a separate step, and each step handles different constraints in isolation. By this, we gain several benefits towards addressing the challenges. First, each step focuses on one type of optimization, making it an easier problem to solve. Second, dividing the logic into consecutive steps provides flexibility to handle queries with any arbitrary subset of different constraints (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. MIN, MAX, and SUM constraints are used as filters to oust invalid areas and provide seed areas for the following steps. Then, AVG constraint, which is the most challenging to satisfy, takes the seed areas and grows as many regions as possible. The last step performs only necessary modifications to satisfy any violated SUM and COUNT constraints, which are easier to satisfy due to the monotonicity of SUM and COUNT aggregate functions. Fourth, separated steps facilitate maximizing the value of p, as detailed later, without violating the constraints that have been satisfied previously. Consequently, our construction phase produces an initial partitioning for the space with a near-optimal p value that is fed to the following phase to minimize the regions' heterogeneity. Although reducing the overall heterogeneity score is an objective for the final solution, this phase is not primarily designed to minimize heterogeneity, which is optimized in the following phase.

The construction phase executes 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 satisfies extrema constraints (MIN and MAX constraints), the second step satisfies centrality constraints (AVG constraints), and the third step satisfies counting constraints (SUM and COUNT constraints). We detail each step below. The area set in Figure 1a is used as a running example to illustrate different steps of the construction. For simplicity and without loss of generality, we assume that all constraints are imposed on the same attribute s.

Step 1: Filtering and Seeding. The first step satisfies extrema constraints, i.e., MIN and MAX constraints. These two constraints have two functionalities: **(a) filtering out the invalid areas, and (b) specifying the set of seed areas**. Invalid areas are excluded during the feasibility phase (Section V-A). We select seed areas as the areas that meet lower



(a) Example area set

(b) Area filtration and seed area selection

Fig. 1: Step 1: Filtering and Seeding

and upper bound values of one of MIN or MAX constraints. 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 rule is generalizable to any number of MIN and MAX constraints.

This step provides three levels of optimizations through choosing a seed area that satisfies the bounds of only one constraint. First, it finds seed areas for any arbitrary combination of MIN/MAX constraints. For constraints that are imposed on different attributes or have disjoint upper and lower bounds, there is no single area that satisfies all of them. So, considering each constraint in isolation from others provides flexibility to find seeds for any query. 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. So, using a maximum number of seed areas and growing the regions based on them contributes to the objective of maximizing p. Third, it enables piggybacking the seed areas selection on the invalid areas filtration during the *feasibility phase*. While checking the validity of an area, the algorithm checks the seeding conditions and mark seed areas.

Example: Figure 1b illustrates the result of Step 1 for the example area set in Figure 1a. 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 Figure 1b. 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 Figure 1b.

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 invalid, 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 $c = (AVG, s, l, u)$, without violating the MIN/MAX constraints. This step is divided into three substeps. 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 detail each substep below.

Substep 2.1 utilizes the set of seed areas, called $seeds$, from Step 1 to initialize a set of initial regions. Areas in $seeds$ already satisfy at least one of the MIN/MAX constraints, so they must be included in any 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 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.

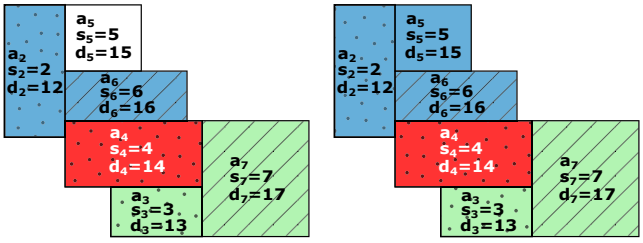
Substep 2.2 tries to assign all remaining unassigned areas, either seed areas or regular areas. Similar to classifying $seeds$ areas in Substep 2.1, all areas that are not in $seeds$ set are added to either $unassigned\_avg$ set, $unassigned\_low$ set, or $unassigned\_high$ set, depending on their s 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 $unassigned\_high$, we must check if adding it to the neighbor region violates c. The second round tries to assign the areas in $unassigned\_high$ and $unassigned\_low$ by merging the regions. Given an unassigned area a, the algorithm tries to merge one of a's neighbor regions R and R's neighbor region with a and checks if c is satisfied for the newly merged region. The merging process terminates when all a's neighbors cannot absorb it or a merge limit, i.e., the number of allowed merge trials, is reached. The merge limit is set to prevent the formation of oversized regions and control the runtime.

Substep 2.3 combines neighbor regions to ensure that all regions satisfy all MIN and MAX constraints. Because areas that form $seeds$ are selected by a single MIN or MAX constraint, regions that are aggregated based on a single seed area are guaranteed to satisfy only one of those constraints. To ensure each region satisfies all constraints, the algorithm iterates over the region list P and merges the region that does not satisfy every constraint with one of its neighbor regions

Algorithm 1: Region Growing - Merging Areas

Input: $c = (AVG, s, l, u)$, $unassigned\_low$, $unassigned\_high$, P
Output: $unassigned\_low$, $unassigned\_high$, P

```
removed_areas ← ∅
u_areas ← unassigned_low ∪ unassigned_high
foreach area a ∈ u_areas do
    Create a tempory region R with R.id = -1;
    R.addArea(a);
    updated = true;
    neighbor_areas = R.getNeighbors();
    while updated do
        updated = false;
        if l ≤ R.getAverage() ≤ u then
            R.updateId(P.size() + 1);
            P.add(R);
            removed_areas.addAll(R.areas);
        end
        else
            foreach area a_n in neighbor_areas do
                if a_n is not assigned to any region then
                    if (R.getAverage() < l AND a_n.s > u)
                     OR ((R.getAverage() > u AND a_n.s <
                     l)) then
                        R.addArea(a_n);
                        updated = true;
                        break;
                    end
                end
            end
        end
    end
end
unassigned_low.removeAll(removed_areas);
unassigned_high.removeAll(removed_areas);
return unassigned_low, unassigned_high, P;
```

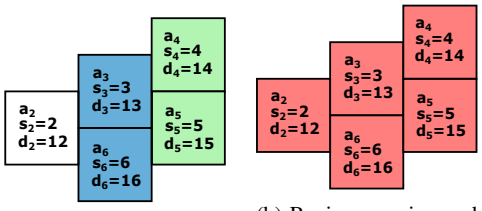


(a) Substep 2.1: Region Initialization (b) Substep 2.2: Area Assignment

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

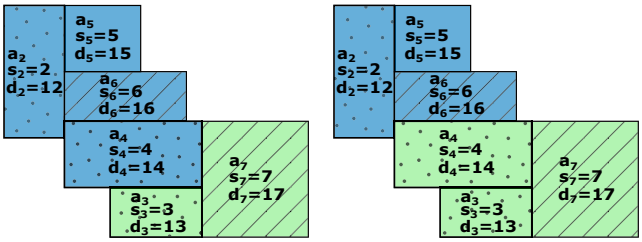
that satisfy other constraints. The iteration stops when all regions in the P satisfy MIN and MAX constraints, i.e., each region contains at least one seed area for each MIN/MAX constraint. This substep does not affect satisfying the AVG constraint c because merging two regions that already satisfy c produces a region that still satisfies it. Thus, after Step 2 concludes, all regions in the region list P satisfy all the MIN, MAX, and AVG constraints.

Example: Figure 2 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} (Figure 2a). a_2 and a_3 are



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

Fig. 3: Step 2: Region Growing - Substep 2.2 - Round 2



(a) Case 1 (b) Case 2

Fig. 4: Step 2: Region Growing - Substep 2.3

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 Figure 2a. Then, in Substep 2.2, the remaining unassigned area a_5 is assigned to its neighbor region R_{blue} as depicted in Figure 2b.

Example: Figure 3 provides an example for Round 2 of Substep 2.2. In contrast to the example in Figure 2, 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 $c = (AVG, s, 4, 5)$. Instead, the algorithm forms a temporary region $R_{red} = \{a_3, a_4, a_5, a_6\}$ by merging R_{blue} and its neighbor region $R_{green} = \{a_4, a_5\}$, as shown in Figure 3b. R_{red} accepts a_2 and the average value is 4.4, which satisfies the constraint c.

Example: Figure 4 gives an example of Substep 2.3 for the example region partition in Figure 2b. $R_{red} = \{a_4\}$ 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 Figure 4a and Figure 4b, respectively.

Complexity analysis. 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 the Substep 2.2 iterates through the unassigned area set for multiple iterations. In each iteration, at least one area is removed from the unassigned area set or otherwise no update is made and the loop breaks. The number of unassigned areas is $\leq n$, hence the worst-case total number of operations is $\sum_{i=1}^{n} i = O(n^2)$. The merge operations are bounded by a constant number and each

operation takes constant time to update the attribute values and append the area list. 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 satisfies SUM and COUNT constraints while maintaining the constraints that have been satisfied in the previous steps. The SUM constraint is used in the MP-regions problem, hence we build this step based on the MP-regions construction algorithms [52]. However, because regions are initialized for the MIN, MAX, and AVG constraints, adjustments are needed on both area level and region level to satisfy the counting constraints. Because SUM and COUNT constraints are monotonic, areas are added to regions that fall under the lower bound or removed from regions that go above the upper bound to make every region valid. We iterate over constructed regions. For each region that violates one of the SUM or COUNT constraints, the algorithm first tries to swap areas with neighbor regions without violating the other constraints. The spatial contiguity of the region is validated by ensuring that the donor region still forms a single connected component after swapping. After no updates can be made by swapping areas, the algorithm tries to merge regions below the lower bounds or remove areas from regions above the upper bounds. When no changes can be made, the infeasible regions are removed, and the partition is considered finalized for the construction phase.

Example: With the input shown in Figure 4a 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 Figure 4b. 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 because the region that becomes feasible after the swap keeps the area to remain valid. Each swap updates and validates all attributes of both the donor and receiver regions, which takes $O(2m)$ operations. 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 take 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 forwarded from the construction phase to the local search phase to optimize the overall regions' heterogeneity. The local search phase uses the Tabu search algorithm [23], a meta-heuristic search algorithm, to minimize the overall heterogeneity.

In brief, the Tabu search algorithm keeps moving areas among neighbor regions without violating the user-defined constraints in any region. Starting from the feasible partition constructed in the construction phase, the Tabu search algorithm moves to the best neighboring partition. Moving to a solution that has worse total heterogeneity is allowed to escape from the local optimal solution. The made moves are stored in a tabu list, with a fixed tabu tenure, and the reverse moves are forbidden to prevent cycles. 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 algorithm stops when no better move can be made in a specified number of steps, and the best partition found is returned as the final result. This phase does not change the number of regions *p*. Instead, it still produces *p* regions but with better overall heterogeneity.

Complexity analysis. The Tabu search is an incomplete algorithm, so we derive a complexity approximation based on the parameters and the problem-specific neighborhood computation. Tabu search performs n iterations without heterogeneity improvement and the counter resets on improvement, so the total number of iterations is $O(\alpha n)$ in the worst case, where α is the number of moves that beat the existing optimal heterogeneity score. In each iteration, we update the valid moves using a region connectivity check for each boundary area in the region updated by the previous move, which takes $O(n \times n)$ time in the worst case. A move attempt takes $O(n)$ to compute the pair-wise dissimilarity in the worst case. So, the overall time complexity is $O(\alpha n \times (n^2 + n)) = O(\alpha n^3)$. Experimentally, the moves that improve heterogeneity happen at an early stage of the local search, causing the total number of iterations much smaller 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

Our discussion so far assumes the incoming query contains all types of constraints. However, it is common to have a subset of these constraints. For example, a query could have only one AVG constraint, or one MIN constraint and one COUNT constraint. This section discusses how the $FaCT$ algorithm handles queries with an arbitrary subset of constraints.

The $FaCT$ algorithm handles absent constraints as they exist with an infinite range $(-\infty, \infty)$, which affects both the feasibility and the construction phases. If a MIN/MAX constraint type is absent, the feasibility phase does not remove any invalid areas under MIN/MAX, and Step 1 of the construction phase selects all areas as the 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. Therefore, designing the $FaCT$ algorithm with independent steps enables it to effectively handle queries with any subset of constraints.

VI. Complexity Analysis

This section analyzes the time and space complexity of the $FaCT$ algorithm. 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 $O(\alpha n^3)$. This gives the overall worst-case time complexity of $FaCT$ as $O(\alpha n^3)$, where α is the total number of moves that improves the existing optimal heterogeneity in the local search process.

For space complexity, the area set takes $O(m \times n)$ space to store the n areas, each associated with m attributes assuming there are m constraints. The space complexity for the seed area selection is $O(n)$. The Region Growing step stores temporary regions with $O(n)$ areas. After the region initialization, each region is stored with m attributes and an area set. 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 update the region list in place and does not require additional space. Thus, the overall space complexity is $O(m \times n)$.

VII. Experiment evaluation

This section presents an extensive experimental evaluation of our proposed work to address the EMP problem. The code and data for reproducing the results presented in this section are publicly available [27]. The rest of the section introduces the experimental setup (Section VII-A), studies the impact of different constraints types and threshold ranges on the performance (Section VII-B), shows the scalability of the $FaCT$ algorithm (Section VII-C), and summarizes the experimental results (Section VII-D).

A. Experimental Setup

EMP is a novel problem in the literature, so there is no direct competitor to compare our techniques against. In addition, due to the major enrichment EMP introduces to the MP-regions problem, existing techniques that address MP-regions problem cannot be trivially extended and adopted as baselines to solve the EMP problem. Therefore, our experiments focus on evaluating the performance and effectiveness of our proposed work. 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 1k) that includes 1012 areas, (2) the Los Angeles County (denoted as 2k) that includes 2344 areas, (3) Southern California (denoted as 4k) as identified by the Southern California Association of Governments (SCAG) [45] that includes 3947 areas, (4) the State of California (denoted as 8k) that includes 8049 areas, and (5) Five multi-state datasets with ∼10k to ∼50k areas, as detailed in Table I. Our default dataset is 2k. Typical evaluation datasets in the existing literature have between 300-3000 areas [15], [52], so our default dataset size is comparable to the largest datasets in the literature. All datasets are joined with real attributes from 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* attribute represents the population with age sixteen or above. *EMPLOYED* attribute refers to the employed population. *TOTALPOP* is the total population. *HOUSEHOLDS* is the number of households in each area, and it is used to measure region heterogeneity.

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

TABLE I: Description of the multi-state datasets

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

TABLE II: Default attribute and ranges for different constraints.

The evaluation attributes are selected based on factors that influence the population growth rate [21], 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 [9] and SCAG [37] and joined using the QGIS software [42].

All experiments are based on Java 14 implementations using an Intel Xeon W-2123(3.60 GHz) and 20GB RAM allocated under Windows 10. We use three performance measures: the runtime for both construction and local search phases, the answer set size p, and improvement in regions' heterogeneity, defined as the ratio of the absolute difference between the heterogeneity score before and after the local search phase to the heterogeneity score before the local search. Our parameters include threshold range and dataset size. Unless mentioned otherwise, the default dataset is 2k, area pickup criteria are random, the AVG merge limit is three, the length of the tabu list equals ten, and the maximum number of moves without improvement for Tabu search equals the dataset size. The default threshold ranges and attributes are shown in Table II for each constraint type. For space limitation and similarity of results on aggregates of the same type, we present results for one aggregate function in each constraint type.

B. 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 VII-B1, VII-B2 and VII-B3 evaluate sets that include MIN, AVG, and SUM constraints, respectively. MIN con-

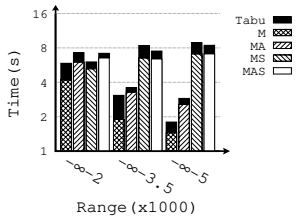


Fig. 5: Runtime for MIN with $l = \infty$

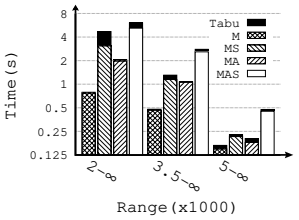
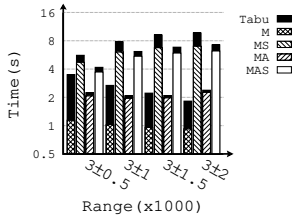
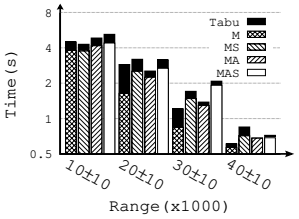


Fig. 6: Runtime for MIN with $u = \infty$



(a) Varying range lengths



(b) Varying range midpoints

Fig. 7: Runtime for MIN with bounded l and u

straints 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 invalid 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 show the impact of u values on seed area selection, (b) ranges with $u = \infty$, which show the impact of l values on invalid area filtration, and (c) ranges with bounded values of l and u that show their simultaneous impact.

Ranges with $l = -\infty$. Figure 5 shows the runtime of the construction phase and Tabu search. The runtime highly depends on the types of constraints. Table III gives the number of regions p for different constraints combinations. In all cases, the single MIN constraint, M, produces the maximum of regions p that is bounded by the number of seed areas. The p value increases with the increase of the upper bound u. As u increases, the number of seed areas increases and it takes fewer iterations to partition the areas into regions, so the runtime decreases in Figure 5. When the other constraints are added, the regions in the final result contain more than one seed area, so p decreases. 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 actually decreases. Increasing u leads to increasing the heterogeneity improvement from 6.96% at $u = 2k$ to 40.2% at $u = 5k$ due to higher p.

Ranges with $u = \infty$. Table III shows the p values and Figure 6 gives the construction time and local search time for different constraint combinations. As we increase the lower bound l, more invalid areas are filtered out, while typically the remaining areas are scattered. As a result, the p value decreases, and hence runtime decreases significantly due to the lower number of regions. The heterogeneity score improvement reduces as l increases with up to 17.6% due to decreasing p that narrows the search space of the local search.

Ranges with bounded l **and** u**.** We explore bounded ranges that vary the range length while fixing the range midpoint. Table III gives the p values and Figure 7a shows the 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 larger search space. However, there is no clear trend for the Tabu search time.

For a range with a fixed length and shifted midpoint to a higher value, both construction time and local search time decrease, as shown in Figure 7b. For larger values, the area set is chopped into small connected components but the number of seed areas is relatively stable. As a result, Steps 2 and 3 of the construction phase terminate faster and reduce runtime. The p value depends on the correlation between attributes of MIN and AVG constraints. If they are not correlated, then the p value strictly decreases for larger midpoint values. When they are correlated, p might increase or decrease, depending on 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. In this experiment, the length of the AVG range is fixed to be $2k$ and the midpoint shifts from $1k$ to $4.5k$, with the step size = $0.5k$. We focus on the performance of the AVG constraint to exclude the impact of other constraints. Figure 8 shows the distribution of the AVG attribute value of the areas in the default dataset, which is a positively skewed distribution. The AVG attribute values of most of the areas are below $4k$, but there are several outliers with values up to 6149. Figure 9a shows the p value and the number of unassigned areas (UA) for each range while Figure 9b gives the runtime. When the midpoint is set to be $1k$ to $2.5k$, about half of the areas lie within the given range and it is also relatively easy to combine areas whose value is outside the range. As a result, the number of unassigned areas is reduced to 0 when the midpoint increases to $2k$ and $2.5k$ but the runtime is below $3.2s$. The ranges are also flexible for the Tabu search phase, with the heterogeneity score improved by up to 30.1%. When the range is set to be $3k \pm 1k$, more than half of the areas lie below l but it is possible to combine them with one of the outliers whose attribute value is well above the upper bound. In addition, as an area is assigned, the newly formed region may lead to the possibility of assigning the neighbor areas. Hence the enclave assignment process continues for multiple iterations until no further update can

	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

TABLE III: p values for different threshold ranges for MIN constraint combinations.

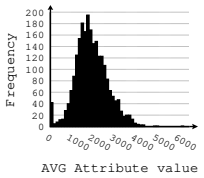
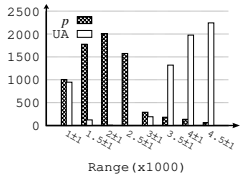
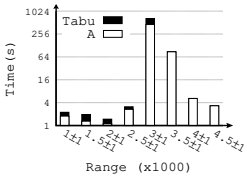


Fig. 8: Distribution of AVG attribute value

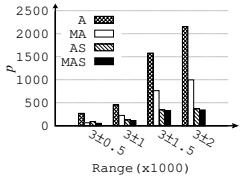


(a) p and UA

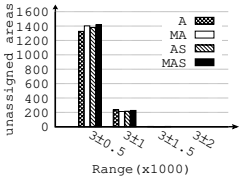


(b) Runtime

Fig. 9: Impact different AVG range midpoints



(a) p



(b) Unassigned areas

Fig. 10: Impact different AVG range lengths

be made. As a result, it takes a much longer time, but the percentage of unassigned areas is reduced significantly to 9%. When the midpoint is further increased to above $3.5k$, it is too hard for the algorithm to form feasible regions as almost all areas fall below the lower bound of the range. As a result, most areas remain unassigned and the construction time becomes much shorter as the algorithm quickly finds that no further changes can be made. When the midpoint is above $3k$, there are limited feasible moves for the Tabu search because the AVG constraint is too tight. As a result, the Tabu search time is negligible and the heterogeneity improvements are below 0.4%.

Ranges with fixed midpoint. Based on the observation from the previous experiment, we set the midpoint to be $3k$, which is the most challenging setting for the algorithm, and alter the length of the range. Combinations with the other constraints are also included to demonstrate how the AVG constraint can bottleneck the performance in special cases. The p values for different constraint combinations are given in Figure 10a and the runtime is shown in Figure 11. The figures show increasing p values with longer ranges, but the runtime depends on the range length. Different range lengths affect the runtime significantly, while different constraint combinations with the same range have much less impact on the construction time. This is because the runtime of the AVG step dominates the total runtime of the construction phase. When the range is $3k \pm 0.5k$, the range is so tight that the algorithm quickly finds no possible enclaves assignment, so the AVG step terminates early. However, 60% of the areas remain unassigned, as shown in Figure 10b. Similar to the previous experiment, the range $3k \pm 1k$ is the most challenging case and the runtime dominates the other constraints. However, the reduction in the number of unassigned areas is also significant for all constraint combinations. When the range is further enlarged, most of the areas lie within the range and it is much easier to assign the enclaves 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. The heterogeneity improvement increases to up to 21.8% because when the range is enlarged, it becomes

	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

TABLE IV: p values for different threshold ranges for SUM constraint combinations.

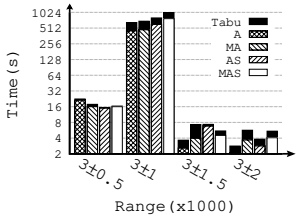


Fig. 11: Runtime for AVG with different range lengths

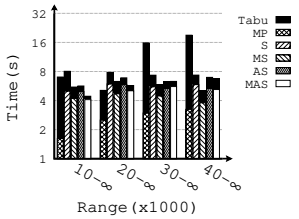


Fig. 12: Runtime for SUM with $u = \infty$

easier to find a valid move for the local search.

3) 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 IV shows the p values and Figures 12 and 13 give the runtime for different constraint combinations with SUM aggregate. At $u = \infty$, the p value decreases with increasing lower bound l while runtime does not increase as much for both MP-regions and $FaCT$. According to Table IV, our $FaCT$ algorithm gives comparable p value to MP-regions when the constraints are set to be the same. The construction time is slightly higher due to the overhead introduced by steps that validate the feasibility of the query and handle the other constraints. However, due to shorter tabu search time, the overall runtime of EMP is less than half of the runtime for the MP-regions when $u = 30k$ or $40k$. In addition, the $FaCT$ algorithm handles generic types of constraints, which are not supported by the competitor. The heterogeneity score increases when l increases, by up to 72.6% for both MP-regions and the single constraint S, 13.8% for MS, and less than 1% for AS and MAS due to the effect of AVG

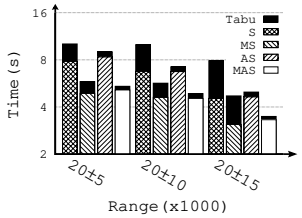


Fig. 13: Runtime for SUM with a changing range length

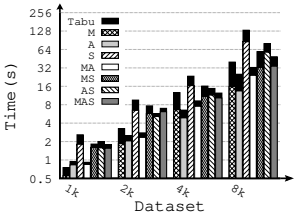


Fig. 14: Runtime varying datasets 1k to 4k

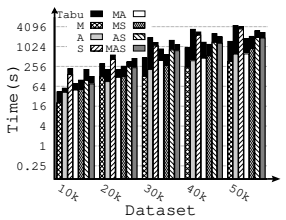


Fig. 15: Runtime varying datasets for 10k-50k

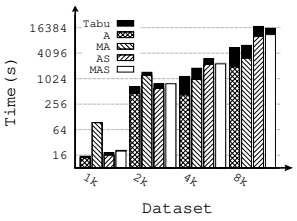


Fig. 16: Runtime varying datasets for AVG constraint with range $3k \pm 1k$

constraint. Increasing the region's size provides local search with more potential moves, hence it better improves overall heterogeneity. For bounded threshold ranges, in Table IV and Figure 13, both p values and construction runtime decrease when the lower bound l increases and the trend is similar to $u = \infty$. The heterogeneity score increases by up to 30.4% for S, 6.9% for MS, 3.76% for AS, and 2.49% for MAS, with increasing range length due to the larger region size. 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. $FaCT$ algorithm reports output statistics to users so they are equipped with information about the impact of different threshold ranges on the given dataset, and are able to tune query parameters insightfully.

C. FaCT Scalability

This section discusses the scalability of the $FaCT$ algorithm. We run the algorithm for different constraint combinations with the default threshold values on the nine evaluation datasets. The dataset size of the 50k dataset is 17 times the largest dataset used in the existing literature of regionalization [15], [52]. As the AVG constraint is identified as a clear performance bottleneck when the range is set to be $3k \pm 1k$, our discussion contains two parts. The first part shows the results for the constraint combinations with the AVG set to be default range (Figure 14 and Figure 15) to show the scalability of $FaCT$ in normal cases, while in the second part we focus on constraint combinations including AVG with range $3k \pm 1k$ (Figure 16) to see how the computation of the AVG constraint scales in extreme cases.

Figure 14 and 15 show that when the AVG constraint is not the bottleneck, the runtime increases linearly with the input size for M and quadratically for the other constraints. For all datasets, $FaCT$ provides a very acceptable runtime for regionalization applications.

Figure 16 shows that when the range of the AVG is set to be $3k \pm 1k$, AVG constraints increase the construction time to a large extent. For other datasets, the runtime increases with increasing input size at a much higher rate than the previous experiment, which implies less scalability advantage for AVG constraint. The construction runtime does not strictly increase with increasing the input size. The construction time for the 4k dataset is shorter than that for the 2k dataset, except for 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.

D. Summary Of Results

All experiments show that the $FaCT$ algorithm can efficiently produce feasible solutions of high quality in a few seconds for the vast majority of the cases. In addition, the $FaCT$ algorithm supports relatively large datasets that serve all existing applications. When more areas are filtered or fewer seed areas are identified by the extrema constraints, both p and the runtime tend to decrease. The behavior of the centrality constraint depends on the range specified and the distribution of the attributes. However, it shows reasonable performance in the vast majority of the cases. The counting constraints can formulate the MP-regions problem as a sub-problem of EMP and gives comparable scalability and result quality.

VIII. Conclusion

This paper introduces an enriched max-p (EMP) regionalization problem that extends the existing regionalization problems with SQL-inspired user-defined constraints. The EMP problem clusters a set of spatial areas into homogeneous regions that satisfy the user-defined constraints. The EMP problem enables enriched types of constraints that support SQL-inspired aggregate functions, MIN, MAX, AVG, SUM, and COUNT, with range operators. We prove the NP-hardness of the EMP 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 for 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.

For future work, we explore multi-objective optimization methods and further improve the algorithm performance through parallelization.

REFERENCES

[1] NP-hardness of MP-regions. https://cs.ucr.edu/~amr/MP_Regionalization_NPHardness.pdf, 2021.

[2] K. Andreev and H. Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[3] M. P. Armstrong, G. Rushton, R. Honey, B. T. Dalziel, P. Lolonis, S. De, and P. J. Densham. Decision Support for Regionalization: A Spatial Decision Support System for Regionalizing Service Delivery Systems. *CEUS*, 15:37–53, 1991.

[4] D. Arribas-Bel and C. R. Schmidt. Self-Organizing Maps and the US Urban Spatial Structure. *EPB*, 40:362–371, 2013.

[5] R. M. Assunção, M. C. Neves, G. Câmara, and C. da Costa Freitas. Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees. *International Journal of Geographical Information Science*, 20(7):797–811, 2006.

[6] O. Aydin, M. V. Janikas, R. Assunção, and T.-H. Lee. Skater-con: Unsupervised regionalization via stochastic tree partitioning within a consensus framework using random spanning trees. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, pages 33–42, 2018.

[7] R. Benedetti, F. Piersimoni, G. Pignataro, and F. Vidoli. The Identification of Spatially Constrained Homogeneous Clusters of Covid-19 Transmission in Italy. *RSPP*, 12:1169–1187, 2020.

[8] N. Bullen, G. Moon, and K. Jones. Defining localities for health planning: a gis approach. *Social Science & Medicine*, 42(6):801–816, 1996.

[9] U. C. Bureau. Explore census data. https://data.census.gov/cedsci/.

[10] J. Byfuglien and A. Nordgård. Region-building—a comparison of methods. *Norwegian Journal of Geography*, 1973.

[11] M. Camacho-Collados, F. Liberatore, and J. M. Angulo. A multi-criteria police districting problem for the efficient and effective design of patrol sector. *European journal of operational research*, 246(2):674–684, 2015.

[12] Z. Chen, P. Cheng, L. Chen, X. Lin, and C. Shahabi. Fair task assignment in spatial crowdsourcing. *Proceedings of the VLDB Endowment*, 13(12):2479–2492, 2020.

[13] 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. *JEM*, 41:425–440, 2008.

[14] J. H. Danumah, S. N. Odai, B. M. Saley, J. Szarzynski, M. Thiel, A. Kwaku, F. K. Kouame, and L. Y. Akpa. Flood risk assessment and mapping in abidjan district using multi-criteria analysis (ahp) model and geoinformation techniques,(cote d'ivoire). *Geoenvironmental Disasters*, 3(1):1–13, 2016.

[15] J. C. Duque, L. Anselin, and S. J. Rey. The max-p-regions problem. *Journal of Regional Science*, 52(3):397–419, 2012.

[16] J. C. Duque, R. L. Church, and R. S. Middleton. The p-regions problem. *Geographical Analysis*, 43(1):104–126, 2011.

[17] 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. *LUP*, 135:11–21, 2015.

[18] 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). *TAC*, 113:387–405, 2013.

[19] A. Ferligoj and V. Batagelj. Clustering with relational constraint. *Psychometrika*, 47(4):413–426, 1982.

[20] D. C. Folch and S. E. Spielman. Identifying regions based on flexible user-defined constraints. *International Journal of Geographical Information Science*, 28(1):164–184, 2014.

[21] R. Fragoso, C. Rego, and V. Bushenkov. Clustering of territorial areas: A multi-criteria districting problem. *Journal of Quantitative Economics*, 14(2):179–198, 2016.

[22] R. S. Garfinkel and G. L. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):B–495, 1970.

[23] F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[24] A. Gordon. A survey of constrained classification. *Computational Statistics & Data Analysis*, 21(1):17–29, 1996.

[25] L. Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[26] P. Hansen, B. Jaumard, C. Meyer, B. Simeone, and V. Doring. Maximum split clustering under connectivity constraints. *Journal of Classification*, 20(2):143–180, 2003.

[27] Y. Kang. Fact. https://github.com/YunfanKang/FaCT, 2021.

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

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

[30] H. Kim, Y. Chun, and K. Kim. Delimitation of Functional Regions Using a P-Regions Problem Approach. *IRSR*, 38:235–263, 2015.

[31] K. Kim, Y. Chun, and H. Kim. p-Functional Clusters Location Problem for Detecting Spatial Clusters with Covering Approach. *Geographical Analysis*, 49:101–121, 2017.

[32] J. Laura, W. Li, S. J. Rey, and L. Anselin. Parallelization of a Regionalization Heuristic in Distributed Computing Platforms–a Case Study of Parallel-P-Compact-Regions Problem. *IJGIS*, 29:536–555, 2015.

[33] L. P. Lefkovitch. Conditional clustering. *Biometrics*, pages 43–58, 1980.

[34] W. Li, R. L. Church, and M. F. Goodchild. An Extendable Heuristic Framework to Solve the P-Compact-Regions Problem for Urban Economic Modeling. *CEUS*, 43:1–13, 2014.

[35] W. Li, R. L. Church, and M. F. Goodchild. The p-Compact-Regions Problem. *Geographical Analysis*, 46:250–273, 2014.

[36] F. Murtagh. A survey of algorithms for contiguity-constrained clustering and related problems. *The computer journal*, 28(1):82–88, 1985.

[37] S. C. A. of Governments. Scag open data portal. https://gisdata-scag.opendata.arcgis.com/.

[38] S. Openshaw. A regionalisation program for large data sets. *Computer Applications*, 3(4):136–147, 1973.

[39] S. Openshaw. A geographical solution to scale and aggregation problems in region-building, partitioning and spatial modelling. *Transactions of the institute of british geographers*, pages 459–472, 1977.

[40] S. Openshaw. Classifying and regionalizing census data. *Census users' handbook*, pages 239–270, 1995.

[41] 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. *Applied Geography*, 55:48–60, 2014.

[42] Welcome to the QGIS project! https://www.qgis.org/, 2020. May 2020.

[43] S. J. Rey and M. L. Sastré-Gutiérrez. Interregional Inequality Dynamics in Mexico. *SEA*, 5:277–298, 2010.

[44] D. J. Rossiter and R. J. Johnston. Program group: the identification of all possible solutions to a constituency-delimitation problem. *Environment and Planning A*, 13(2):231–238, 1981.

[45] Southern California Association of Governments - SCAG. https://scag.ca.gov/, 2020. May 2020.

[46] 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. *JESPR*, 20:6917–6933, 2013.

[47] B. She, J. C. Duque, and X. Ye. The network-max-p-regions model. *IJGIS*, 31:962–981, 2017.

[48] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu. Trichromatic online matching in real-time spatial crowdsourcing. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1009–1020. IEEE, 2017.

[49] S. E. Spielman and D. C. Folch. Reducing Uncertainty in the American Community Survey through Data-driven Regionalization. *PloS ONE*, 10:e0115626, 2015.

[50] D. A. Stow, C. D. Lippitt, and J. R. Weeks. Geographic Object-based Delineation of Neighborhoods of Accra, Ghana using QuickBird Satellite Imagery. *PE&RS*, 76:907–914, 2010.

[51] T. Vilutienė and E. K. Zavadskas. The application of multi-criteria analysis to decision support for the facility management of a residential district. *Journal of Civil Engineering and Management*, 9(4):241–252, 2003.

[52] R. Wei, S. Rey, and E. Knaap. Efficient Regionalization for Spatially Explicit Neighborhood Delineation. *IJGIS*, 35:1–17, 2020.

[53] S. Yanık, J. Kalcsics, S. Nickel, and B. Bozkaya. A multi-period multi-criteria districting problem applied to primary care scheme with gradual assignment. *International Transactions in Operational Research*, 26(5):1676–1697, 2019.

[54] X. Ye, B. She, and S. Benya. Exploring Regionalization in the Network Urban Space. *JGSA*, 2:4, 2018.