5-2-2018

Exploring Parallel Efficiency and Synergy for Max-P Region Problem Using Python

Viney Sindhu

EXPLORING PARALLEL EFFICIENCY AND SYNERGY FOR MAX-P REGION PROBLEM USING PYTHON

by

VINEY SINDHU

Under the Direction of Sushil Prasad, Ph.D.

ABSTRACT

Given a set of n areas spatially covering a geographical zone such as a province, forming contiguous regions from homogeneous neighboring areas satisfying a minimum threshold criterion over each region is an interesting NP-hard problem that has applications in various domains such as political science and GIS. We focus on a specific case, called Max-p regions problem, in which the main objective is to maximize the number of regions while keeping heterogeneity in each region as small as possible. The solution is broken into two

phases: Construction phase and Optimization phase. We present a parallel implementation of the Max-p problem using Python multiprocessing library. By exploiting an intuitive data structure based on multi-locks, we achieve up 12-fold and 19-fold speeds up over the best sequential algorithm for the construction and optimization phases respectively. We provide extensive experimental results to verify our algorithm.

INDEX WORDS: Clustering, Geospatial, Homogeneous Regions, Multiprocessing, Optimization

EXPLORING PARALLEL EFFICIENCY AND SYNERGY FOR MAX-P REGION PROBLEM USING PYTHON

by

VINEY SINDHU

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2018

EXPLORING PARALLEL EFFICIENCY AND SYNERGY FOR MAX-P REGION PROBLEM USING PYTHON

by

VINEY SINDHU

Committee Chair: Sushil Prasad

Committee: Rajshekhar Sunderraman

Yanqing Zhang

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2018

DEDICATION

I dedicate this thesis to my parents, Bijender Sindhu and Sudesh Sindhu, whose love and support laid the foundation for discipline necessary to complete this work.

ACKNOWLEDGEMENTS

This project would not have been possible without the support of many people. I would like to thanks Dr. Sushil Prasad, Dr. Rajshekhar Sundarraman, Dr. Yanquing Zhang and Danial Aghajarian for their time, guidance, help, and wisdom they have provided. I would also like to thank everyone in the Computer Science Department at Georgia State University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PART 1

INTRODUCTION

Clustering geographical areas into homogenous regions have many applications in various domains such as urban development, districting, transport planning, analyzing crime rates etc [1]. The homogeneous region can be defined as a set of contiguous areas with high degree of similarity for a given attribute such as per capita income, population etc. Generally speaking, problems of this kind are NP-hard [2] as the size of geographical areas to be grouped into homogeneous regions increases, the clustering running time increases exponentially. Hence, it is important to utilize the parallel processing capabilities of modern hardware to perform the computation. Furthermore, improving the accuracy of the solution by efficiently taking advantage of parallel synergy is another important aspect of these kinds of problems.

Many researchers and scientists have contributed to the problem of aggregating areas into homogeneous regions. Duque, Anselin, and Rey [2] referred to it as max-p region problem, Hensen [6] referred to it as clustering under connectivity constraints, Maravalle and Simeone [12] referred to it as regional clustering, Wise [13] referred to it as regionalization. Major challenges in solving this problem are to ensure spatial contiguity of each region, measure homogeneity of each region, explore solution space, and ways to check solution feasibility. Other constraints in solving this problem are the shape of the region, equality of an attribute value across the region, and boundary integrity. Due to these constraints, some different formulations and solution strategies have been contributed by researchers.

Availability of highly disaggregated spatial data and computational resources provides the opportunity for researchers to explore new applications of these clustering models. New opportunities bring new challenges and one of the challenges is ever-increasing data size. DiMOS lab working under the direction of Dr. Prasad has already laid out the roadmap [7,8]

on using GPUs for the parallel processing of geospatial datasets. There are many available algorithms to cluster the areas into homogeneous regions, however, most of them are not scalable. Another challenge is to decide on the number of homogeneous regions to be formed after clustering. Most of the available models require the number of regions as input. Also in order to decide on the best clustering model for a set of given areas, one needs to know about all the available algorithms. Mostly, researchers want to design regions for analysis rather than summarizing and finding the number of regions in the data. In this situation, the researcher may not know the number of regions to be designed, however they may know the condition which will make a region suitable for analysis. This information can be used as criteria to decide the number of regions.

In this thesis, we use the clustering model explained in [2] as the max-p problem. The max-p region model is implemented in python, therefore, to have a benchmark for evaluation of our work, we have also used python for this research. Another reason to use python is that lot of researchers are familiar with python and use it on day to day basis for analysis.

The problem aggregates n areas into an unknown maximum number of homogeneous regions, where each region satisfies a minimum threshold value for a given spatial attribute. The problem is formulated as an Integer Linear Programming (ILP) with a two-part objective function. The algorithm has two phases: first, it tries to form as many initial regions as possible (the first part of the objective function) by starting from random areas called seeds and grow them until each region reaches to the threshold. The remaining areas that do not form regions in the first step are called enclaves and are assigned to the regions. In the second phase, the algorithm tries to optimize the objective function among those solutions with the maximum number of regions (the second part of the objective function). This goal is achieved by exchanging areas in the border of two neighboring regions. Unlike other models, this model does not impose any constraint on the shape of the regions, instead, the number of regions (p) and shape of the regions are data dependent and are decided by the algorithm at runtime. The max-p model tries to maximize the number of homogeneous regions, based on a predefined minimum threshold value. The degree of aggregation bias is minimized by

maximizing the homogeneity within the regions. In this paper, we exploit parallel processing to improve the performance of the max-p model. We also use parallel synergy to improve the accuracy of solutions by trying more solutions in parallel as well as exploiting a fine parallelism model in each solution. Multiple solutions are created simultaneously on multiple processors in parallel and then compared to choose the best feasible solution. Then all the regions of the best feasible solution are processed on multiple processors in parallel to optimize and increase homogeneity within them.

In summary, our key contributions in this work are:

- Parallel implementation of the construction phase of Max-p problem that is up to 12 times faster than the best sequential algorithm for the largest (56×55) lattice size.
- Parallelizing the optimization phase of finding the best feasible solution that is up to 19 times faster than the best sequential algorithm.
- End to end running time of our parallel implementation achieves 13-fold speed up over the best sequential algorithm.
- Using a parallel data structure based on multiple locks to reduce parallelism overhead.
- By taking advantage of parallel synergy, we often improve the secondary objective function better then sequential algorithm by exploring the exponential search space in parallel.

The rest of the thesis is organized as follows. In Section 2, we present literature review. In Section 3, we describe the problem statement. In Section 4, we discuss the algorithm used to solve the problem. In Section 5, we present experimental results and evaluate performance. In Section 6, we summarize the research work.

PART 2

LITERATURE REVIEW

In this section, we present a survey on various methods of clustering areas into a set of homogeneous regions. The algorithm used in one of the methods [14, 15] to aggregate areas into regions has two parts. The first part of the algorithm aggregate areas into regions using conventional clustering algorithm. In this part, areas are clustered based on the attributes and not on the locations. In the second part of the algorithm, regions are created as subsets of the spatially contiguous areas from the already created clusters. The number of regions created from this method depends on the attributes used to calculate homogeneity within regions [16].

Another method [3] tries to aggregate n atomic spatial units into p-compact contiguous regions. Compactness is a property that generalizes the notion of a subset of Euclidean space being closed (that is, containing all its limit points) and bounded (that is, having all its points lie within some fixed distance of each other). The heuristic framework addresses the problem through phases of dealing, randomized greedy and edge reassignment. This MERGE technique uses a novel method of the normalized moment of Inertia (NMI) for computing the compactness of each region. MERGE algorithm has three phases. The first phase is initial growth phase which involves a dealing procedure that ensures that each region grows to a particular viable size. The second phase is growth completion phase, where a randomized greedy algorithm applied to the partial regions from the first phase to create a set of regions that meets all the constraints. Third is local search phase, where heuristic framework allows for backtracking to find improvements by using a form of Simulated Annealing.

The p-Compact-regions Problem can be thought as a regionalization or zonation or the districting problem in different contexts. For example, providing emergency medical service with satisfactory coverage in areas or police deployment in an efficient manner for

designated patrol areas or partitioning an area into compact electoral districts to prevent gerrymandering and many more such cases. Most of the problems consider the only contiguity of surrounding regions without any regard for compactness of individual regions. However, in p-compact regions algorithm, minimization of dissimilarity between each pair of areas within the same region is dismissed in favor of maximizing overall compactness of resultant p-regions. The compactness index (for a particular region) is defined to be directly proportional to the square of the area and inversely proportional to the second MI of the units assigned to that region.

A study [30] was conducted on late-stage breast cancer risk in the state of Illinois by constructing geographic areas. There exists a problem with small population numbers that arise when dividing a geographical region into subregions for developing statistical models for data analysis. Often, sparsely populated regions are formed, and they do not provide a true representation of the data. There have been studies that have tried to tackle the small population problem, and the data suppression that accompanies it, namely Spatial smoothing and Hierarchical Bayesian modelling. The REDCAPc method that has been suggested takes into consideration the data confidentiality and privacy issues that accompany data on Cancer.

The Goal of this method [30] is to divide the Region into homogeneous areas which have similar attributes with a minimum threshold for the number of people within a region. Regions are created in a way such that similarity of the population within the region is maximized. The research also took into consideration the classification of each area as being urban or rural, the access to primary care physicians and clinics that provide for cancer screening, the demographic factors of each neighbourhood within the region. This results in larger Geographic sub-regions.

As a result of this, socio-economic barriers become less significant as opposed to social-cultural barriers. This research also showed that late-stage cancer risk is higher in young patients due to less number of screening visits and primary care visits as compared to older patients. It also demonstrated that the proximity to cancer screening facilities is not that significant a factor as compared to access to a primary care center. The differences that exist

between urban and rural populations do not pose as a significant determinant in the Risk.

Constrained Agglomerative Hierarchical Classification [27] imposes a contiguity constraint. The basic idea of the Constrained Agglomerative Hierarchical Classification algorithm was when performing aggregation give preference to those pairs of the cluster's which are structurally close to each other meaning contiguous. In this way, structurally closed cluster pairs get a favor. To decide if two points are contiguous or not depends on the distance between them which is calculated using Euclidean distance formula and this distance is compared against a threshold value known as a contiguity threshold, and If the distance between two points is beyond the contiguity threshold then these points are not close enough to be considered as contiguous and therefore cannot be aggregated.

In a contiguity constraint hierarchy, there is a "relation of contiguity" that is the symmetric and transmissible relation among the clusters. The concept of transmissibility can be explained as if two clusters "h" and "k" are contiguous to each other, then in all the clusters in which "k" is present and all the clusters in the hierarchy build after "k" will also contiguous to "h". The algorithm is modified at each step only the contiguous clusters pairs are selected. The pairs are determined by comparing their Euclidean distance with the contiguity threshold. The process will continue until there are no more contiguous clusters to pair. For the remaining clusters, the transmissibility property is used for classification. There can be a possibility when there are no more pairs of contiguous clusters that mean some clusters are still unclassified in the Hierarchy, this will happen if the threshold selected is too small. Therefore, to overcome this situation, a new contiguity threshold is selected, and the value of the new threshold should be greater than the previous one. With the new threshold value, the last built hierarchical tree will be rebuilt based on the new relation of contiguity between the clusters.

This algorithm provides a flexibility in selecting the contiguity threshold. And, it might be required to try various threshold values before achieving the complete hierarchical classification. The selection of the contiguity threshold can be based on several factors of the data however if the threshold is too small then very few pairs of clusters are achieved and

if the threshold is too large too many pairs of clusters are achieved and these unnecessarily will increase computing time.

Spatial interaction models [28] are concerned with aggregate spatial behaviour patterns and use data which have been spatially aggregated more than once. In the past literature, it is not concluded that the results of spatial interaction modelling have a behavioural significance, or whether or not they reflect the way in which a study region is partitioned into zones. Nor is much known about the extent to which model performance is influenced by the choice of the zoning system. Various attempts have been made to ascertain the extent to which the performance of different kinds of spatial interaction models are influenced by different specifications of spatial zoning systems. There is a basic need for a comprehensive empirical investigation to explore the significance of zoning-system effects on spatial interaction models.

To determine the effect of zoning system, two null hypothesis were tested and which are (a) that the parameter values and goodness-of-fit statistics for spatial interaction models are substantially unaffected by the choice of zoning system (b) that parameters calculated for an M-zone partition will remain valid for a Z-zone partition, where L is less than M. To test the two null hypotheses, a random sample of 261 twenty-two zone partitions and 87 forty-two zone partitions were generated from the set of seventy-three bus's. These sample sizes are determined by the number of partitions which could be examined in one hour of central processing unit (CPU) time on an IBM 370/168. The goodness of fit of the four models is assessed in terms of a residual standard deviation statistic. The size of the standard deviations of the goodness-of-fit statistics and the large range of results indicated the importance of the zoning system on the performance of these models.

The quantitative revolution in the spatial sciences involved the importation of statistical methods which placed little emphasis on the importance of space. The concept that zoning systems should always be defined independently of the model is based on a statistical rather than a geographical perspective of the model building. The central theme underlying most spatial representation problems is how to approach spatial model building in the absence of any fixed spatially-aggregated data. Zoning systems are not only the most direct way

in which space is represented in the model building but they are also the cause of various aggregation, scale, and autocorrelation phenomena. Accordingly, it is believed that these inherently spatial problems can be solved most readily by seeking a geographical rather than a statistical solution, in terms of an appropriate zone-design procedure.

The algorithm developed to solve this problem for spatial interaction models is an extension of Openshaw [28]. It is a heuristic procedure which attempts to improve the quality of a starting classification of TV bsu's into M zones and uses a random search strategy. It is inefficient in the sense that it does not use any historical information to decide where to search next, but it is not restricted to any particular type of function. At best it will result in a set of solutions which tend to converge in the direction of optimality but which differ from the global optimum by some unknown degree of error. In the situations under which it has been tested the procedure has behaved in a satisfactory and reliable manner, although the principle of caveat emptor must apply to some extent.

Zoning systems have a considerable effect on spatial interaction models, are valuable in applied situations and plays an important role to make a spatial model building. If attention is restricted to short-term forecasting or impact studies in the calibration year, the zoning system will probably retain its optimal properties. The results of the paper indicate the need for a critical reassessment of the conventional approach to spatial study and model building. Until some satisfactory theory comes, an optimal approach is the only practical solution to handle the zone-design problem.

Another method [29] is using the semi-supervised clustering by adapting K-Means for regionalization by specifying contiguity constraints based on a neighbourhood representation. For the Reg K-Means algorithm, three real datasets were used and show improvement in intra-cluster variance, minimization of the objective function and computational time as compared to AZP (optimization method). Regionalization problems are related to aggregation of N regions, objects into K geospatial units or geographical areas. Regionalization approaches are very useful in statistical spatial data analysis, the study of geographic patterns or diseases or spatial econometrics of health studies. It today's world algorithms to

tackle such problems are very useful in smart cities scenario, to find clusters of neighbour areas with common characteristics for different purposes. Regionalization aims at grouping regions according to with their specific features as per neighbourhood restrictions represented by methods using spatial contiguity constraints. Semi-supervised algorithms low computational complexity is helpful to perform we applications cauterization scenarios. Reg K-Means is faster than AZP for analysing optimizations and partition compactness for the larger Dataset considering real situations for finding good partitions in regionalization context.

Supervised regionalization methods [31] are considered as a major contributor in performing a qualitative comparison of various aspects of the spatially contiguous regions. Supervised regionalization methods do not focus on model competition and are only referenced for regionalization process. These regionalization methods are divided into eight groups based on the rules that are applied to the pre-defined sets of adjoining regions. The primary focus of the regionalization method lies with the designing of analytical regions, which encompasses various rules like one area can be a part of one region, the sum total of all the regions should be smaller in comparison of all the areas that are a part of the scope data for the analysis.

One of the approaches that these regionalization methods adapt to perform the analysis on the larger spatial units is Statistical spatial data analysis. Statistical spatial data analysis performs collective analysis on the larger spatial units in order to maintain the integrity and confidentiality of the data, to prevent the difference in the results that may occur due to population, to reduce errors due to outliers inaccurate data. The approach followed by Statistical spatial data analysis helps in the better conceptualization of the information that is present on the maps and such type of analysis can be done in two ways. The first approach uses official areas like counties or states; whereas, the second approach analytically combines pre-defined areas that fit the rule or principle used to carry out the analysis. A posterior condition is usually applied to the explicit and indirect methods that are incapable of satisfying the constraint based on the adjoining regions. Such methods can be differentiated

based on the constraints that are applied to the conventional clusters and constraints that aim to achieve maximum spatial regional compactness.

Regionalization used via conventional rules are very basic in nature as they divide the large datasets using hierarchical partitioning and then segregate the dataset based on the regional rule if the regions are not adjoining in nature. Such aggregation criteria do not enforce the necessity of regional compactness. Changes to data based on the shape of the spatial region are inevitable and the result is always prone to changes based on the centroid issues. Regionalization based on the compactness aims at reducing the product value of the distance of the centroids between the region and ad area and the total number of the population residing in those areas. Another approach that is being used to reduce the impact of such constraints is called integer programming model, under which the assumption that, the total number of residents living in an area is equal to the square feet of that area, is made.

Algorithms based on explicit spatial regional constraint is based on the concept of the hybrid optimization models. This approach focuses on reducing the number of territorial links within a network to redesign the shortest path between the areas. The overall review of the taxonomy of such spatial contiguous regions revealed that it is impossible to achieve perfect regionalization with a single technique due to various attributes and unwanted deficiencies that occur due to large unit areas. Combination of various algorithms and techniques will help in identifying the desirable characteristics, optimizing strategies, preventing aggregation process from being generalized, improving local search process and identifying true profits by designing analytical regions.

Efficient regionalization techniques [19] for socio-economic geographical units using a minimum spanning trees maximizes internal homogeneity in an area. The minimum spanning tree is constructed by first creating a connectivity graph with neighbourhood relationship whose edges have the cost inverse proportional to the similarity of the region. The minimum spanning tree is then partitioned by removal of edges that connect not so similar regions resulting maximized homogeneity. Two-step procedures (non-spatial clustering and

neighbourhood preserving classification), clustering procedure using geo-coordinates as extra attributes, AZP algorithm (explicit neighbourhood relationship) are used. The graph-based regionalization has two challenges, first is reducing the computational cost of optimization and second is to reduce sensitiveness born out of choice of initial position of the tree. These challenges are addressed by first creating MST representing a statistical summary and then implementing heuristic to reduce MST to get spatial clusters.

There is another method [32] called Fuzzy Geographically Weighted Clustering-Ant Colony(FGWC-ACO), which is used to study the characteristics of the population with respect to their geographical area. Although the computational running time of FGWC-ACO is slower than existing FGWC, it is used as an optimization tool to improve demographic clustering accuracy. The experiment shows that by adding context variables to existing FWGC-ACO will result in improvement in computational speed. In the existing FGCW, the performance was slow due to the repeated iterations of ACO algorithm to achieve the best solution. The algorithm follows the behaviour of ant colonies. After evaluating the quality of food, ants chose the shortest path between the food source and the nest. During the return to the nest, ant leaves a chemical pheromone to track the path for the next journey or by the other ants. The quantity of pheromone speaks about the quantity and quality of food. Also, the algorithm explains population characteristics considering the hidden pattern of the different population present in different geographical locations. This fuzzy algorithm categorizes population in different clusters with probabilities. This technique provides a membership value to each cluster.

McMahon [33] discusses two approaches regarding region delineation. The first approach is with the help of visual pattern recognition regions are recognized, these regions have similar landscape properties from its mapped data. But since this procedure has to be done manually, so it requires quite a domain experience. Also, it is bounded by small-scale regions and strenuous to regenerate. The second one was more of a data-driven approach which was used for identifying regions based on spatial variability. Clustering techniques such as k-means and hierarchical clustering are used to segregate the geographical area into smaller

units. The probable limitation to the above clustering technique is that they do not assure that the final regions will be spatially contiguous. So, effective methods are needed that can do clustering that is based on mapped variables but also are spatially contiguous.

Therefore, another method [34] is used to explore the achievability of applying constrained spectral clustering to region delineation problem. Constrained clustering includes set of must-link constraints (ML), cannot-link constraints (CL), or both. Between a pair of data instances, a must-link and a cannot-link constraint define a relationship. A must-link constraint specifies that the samples/cases in the must link relation has to be linked with the same cluster and for cannot-link constraints, it is complete opposite i.e. should not be linked to the same cluster. As compared with traditional clustering the constrained clustering uses domain constraints. Domain constraints are used in Shuai Yuan's and Pang-Ning Tan's activity so that it can guide the algorithm to a prudent solution or away from an unacceptable solution.

Also, to tackle the large dataset, the algorithms [34] should maintain a balance between the region parameters i.e. spatial contiguity and landscape homogeneity. Also, the proper or complete working of an algorithm is very much dependent upon how the spatial constraints are incorporated into the framework. If the algorithms are not able to maintain a balance between spatial contiguity and landscape homogeneity then the results produced might not be contiguous and can possibly have arbitrary shape and size. If considering the other situation where the algorithm is biased to only give the output which is geographically connected then the landscape similarities will be not correct. One of the main advantages of Spectral clustering is its flexibility of integrating groups of similarity functions which has an advantage as compared with a traditional k-means algorithm.

Another method [17, 18] uses x and y coordinates of the centroids of areas as additional attributes in the conventional clustering algorithm. Due to these additional attributes nearby areas will be clustered together. Regions will be spatially contiguous, however, continuity of the areas within the region depends on the weights assigned to x and y coordinates as compared to other attributes [19]. An increase in weights of x and y coordinates will increase

the spatial contiguity within the regions, however, it will lead to decrease in weights assigned to other attributes which in turn decreases the homogeneity within the regions. The main challenge of this method is to decide on how to assign weights to the attributes [17, 27, 38].

There are other methods [16, 18, 26, 28, 36] that ensure spatial contiguity based on the information from the neighbouring structure. For example, adapted hierarchical clustering algorithms [23, 25, 40] allow to merge clusters only if they share common borders. Another example is graph theory based algorithms [19,24,35] which represent areas and neighbouring structures as connected graphs.

The choice of method to aggregate areas into homogeneous regions depends on the problem. For example, if the shape of the region is to be guided by the spatial distribution of variables, then clustering using x and y coordinate method is not appropriate as it generates circular regions. The method purposed in this article ensures contiguity by clustering only those areas into a region which shares a common border.

PART 3

PROBLEM STATEMENT

The main goal of max-p region problem is to cluster the areas into the maximum number of homogeneous regions such that each region satisfies a predefined minimum threshold value for a given spatially extensive attribute [2]. A spatially extensive attribute refers to a parameter that we want to be greater than a minimum threshold for each cluster. The second goal of this model is to achieve the most homogeneity within each cluster. In the other words, max-p aims to minimize total heterogeneity of all clusters. We will define this term later in this section. The way max-p prioritizes these two goals is that if two clustering configurations satisfy minimum threshold criteria, the model chooses the one with the larger number of regions (maximizing p). In case of a tie, the one with less total heterogeneity is selected.[1]

3.0.1 Preliminary Notation

The following is the formulation of max-p problem [2].

Area :

Given n areas $A = \{A_1, A_2, ..., A_n\}$, $(n = |A|)$ and m attributes assigned to each area such that $A_{i,\,y}$ is the y-th attribute of area A_i where $1 \leq y \leq m$. Also, let l_i denote a parameter that we want to be greater than a minimum threshold of area A_i.

Relationship :

Let $d : A \times A \rightarrow R^+ \cup \{0\}$ be the dissimilarity between areas based on the set of attributes Y such that $d_{ij} \equiv d(A_i,\ A_j)$ satisfies the conditions $d_{ij} \geq 0, d_{ij} = d_{ji}$ *and* $d_{ii} = 0$ for $i,\ j = 1,\ 2,\ \ldots,\ n$. Distance functions can be utilized; i.e., d_{ij} can also satify the

[1] The problem statement and equations 3.1 through 3.4 are taken from [2]

subadditivity, or triangle inequality, condition: $d_{ij} \leq d_{ik} + d_{kj}$ for $i,\ j,\ k = 1,\ 2,\ \ldots,\ n$. Let $W = (V,\ E)$ denote the contiguity graph associated with A such that vertices $v_i \in V$ correspond to areas $A_i \in A$ and edges $\{v_i, v_j\} \in E$ if and only if areas A_i and A_j share a common border.

Feasible Partitions of A :

Let $P_p = \{R_1,\ R_2,\ ...,\ R_p\}$ denotes a partition of areas A into p regions with $1 \leq p \leq n$ such that:

$$\begin{cases} |R_k| > 0 \ for \ k = 1,\ 2, ...,\ p \\ R_k \cap R_{k'} = \theta \ for \ k,\ k' = 1,\ 2, ...,\ p \wedge k \neq k' \\ \cup_{k=1}^{p} R_k = A \\ \sum_{A_i \in R_k} l_i \geq threshold > 0 for \ k = 1, 2,\ ....,\ p \end{cases} \tag{3.1}$$

Let II denotes the set of all feasible partitions of A.

We define heterogeneity of a region R_k as Equation 3.2:

$$h(R_k) = \sum_{ij: A_i, A_j \in R_k, i \leq j} d_{ij} \tag{3.2}$$

where $R_k \in P_p$. We also define total heterogeneity of partition P_p as the sum of heterogeneity over all of its regions (Equation 3.3):

$$H(P_p) = \sum_{k=1}^{p} h(R_k) \tag{3.3}$$

Finally, the max-p problem can be formulated as finding $P_p^* \in II$ such that:

$$\begin{cases} |P_p^*| = max(|P_P^*| : P_p \in II) \\ \nexists P_p \in II : |P_p| = |P_p^*| \quad \text{AND} \quad H(P_p) < H(P_p^*) \end{cases} \tag{3.4}$$

Let us illustrate this with a basic example. Table 1 shows a regular lattice with $n = 12$ areas from A_1 to A_{12}. We consider two attributes: 1) y is the average income in an area and l is the number of people in that area where l represents the spatially extensive attribute. The primary objective is to find the maximum number of contiguous regions. We need to group these 12 areas in such a way that each region contains at least 100 people (i.e. threshold = 100). The secondary objective is to find the solution with the lowest heterogeneity based on attribute y. We used Manhattan distance for heterogeneity such that $d_{ij} = |y_i - y_j|$.

A_1 $y_1 = 3000,\ l_1 = 23$	A_2 $y_2 = 3200,\ l_2 = 27$	A_3 $y_3 = 3300,\ l_3 = 33$
A_4 $y_4 = 5200,\ l_4 = 21$	A_5 $y_5 = 5000,\ l_5 = 20$	A_6 $y_6 = 3100,\ l_6 = 20$
A_7 $y_7 = 5300,\ l_7 = 24$	A_8 $y_8 = 5400,\ l_8 = 25$	A_9 $y_9 = 5100,\ l_9 = 25$
A_{10} $y_{10} = 7500,$ $l_{10} = 35$	A_{11} $y_{11} = 7100,$ $l_{11} = 30$	A_{12} $y_{12} = 7300,$ $l_{12} = 40$

(a) Input: $n = 12,\ m = 2$

A_1 $y_1 = 3000,\ l_1 = 23$	A_2 $y_2 = 3200,\ l_2 = 27$	A_3 $y_3 = 3300,\ l_3 = 33$
A_4 $y_4 = 5200,\ l_4 = 21$	A_5 $y_5 = 5000,\ l_5 = 20$	A_6 $y_6 = 3100,\ l_6 = 20$
A_7 $y_7 = 5300,\ l_7 = 24$	A_8 $y_8 = 5400,\ l_8 = 25$	A_9 $y_9 = 5100,\ l_9 = 25$
A_{10} $y_{10} = 7500,$ $l_{10} = 35$	A_{11} $y_{11} = 7100,$ $l_{11} = 30$	A_{12} $y_{12} = 7300$ $l_{12} = 40$

(b) Solution 1: $p = 2,\ H = 36,500$

A_1 $y_1 = 3000,\ l_1 = 23$	A_2 $y_2 = 3200,\ l_2 = 27$	A_3 $y_3 = 3300,\ l_3 = 33$
A_4 $y_4 = 5200,\ l_4 = 21$	A_5 $y_5 = 5000,\ l_5 = 20$	A_6 $y_6 = 3100,\ l_6 = 20$
A_7 $y_7 = 5300,\ l_7 = 24$	A_8 $y_8 = 5400,\ l_8 = 25$	A_9 $y_9 = 5100,\ l_9 = 25$
A_{10} $y_{10} = 7500,$ $l_{10} = 35$	A_{11} $y_{11} = 7100,$ $l_{11} = 30$	A_{12} $y_{12} = 7300,$ $l_{12} = 40$

(c) Solution 2: $p = 3,\ H = 40,100$

A_1 $y_1 = 3000,\ l_1 = 23$	A_2 $y_2 = 3200,\ l_2 = 27$	A_3 $y_3 = 3300,\ l_3 = 33$
A_4 $y_4 = 5200,\ l_4 = 21$	A_5 $y_5 = 5000,\ l_5 = 20$	A_6 $y_6 = 3100,\ l_6 = 20$
A_7 $y_7 = 5300,\ l_7 = 24$	A_8 $y_8 = 5400,\ l_8 = 25$	A_9 $y_9 = 5100,\ l_9 = 25$
A_{10} $y_{10} = 7500,$ $l_{10} = 35$	A_{11} $y_{11} = 7100,$ $l_{11} = 30$	A_{12} $y_{12} = 7300,$ $l_{12} = 40$

(d) Solution 3: $p = 3,\ H = 3,800$

Table (3.1) An example of max-p problem, (a) input areas, (b) a solution with $p = 2$, (c) a feasible solution with $p = 3$, (d) another feasible solution with $p = 3$. Solution 3 is chosen because it has lowest total heterogeneity among partitions with maximum number of regions ($p = 3$).

As shown in Tables 3.1, there are three solutions for partitioning these areas where Solution 2 and 3 reach to the maximum number of regions. Therefore, Solution 1 is not a feasible solution as there are other solutions with more number of regions. Hence, Solution 2 and Solution 3 are the feasible solutions as we can see that it is not possible to have more

than three regions with at least 100 people per region. Also, we can see that Solution 3 has heterogeneity lower than that of Solution 2. Therefore, Solution 3 is the best feasible solution for this example. All the regions of Solution 3 have more than 100 people each. 103 people in R_{white}, 115 people in R_{green} and 105 people in R_{blue}.

PART 4

METHODS

In this section, we present our algorithm to solve the problem of clustering areas into the maximum number of homogeneous regions. The algorithm has two phases, i.e. the construction phase and the optimization phase. The construction phase generates a feasible solution and optimization phase improves the feasible solution by minimizing the heterogeneity within the regions through local swapping of areas between neighbouring regions.[1]

4.0.1 Construction Phase

In this phase, areas are clustered into the maximum number of regions based on a minimum threshold value of the spatially extensive attribute. An initial feasible solution is generated at the end of this phase. We repeat this algorithm multiple times with random seeds (100 times for this experiment) to calculate the set of feasible solutions. We run this algorithm on multiple processors simultaneously for each instance to improve the running time of our algorithm. Once all the processes finish successfully, we compare the output of each process and select the best feasible solution. We use the objective function explained in [2] to decide on which solution is the best out of all the solutions.

Objective function is a minimization function which can be formulated as follows [2].

Parameters:

i = index of areas,

k = index of regions,

c = index of contiguity order (areas are ordered such that areas next to each other share border),

d_{ij} = dissimilarity relationship between areas i and j with $i < j$,

[1] This two phase algorithm is adopted from [2]. We extend the python implementation from [2]

$[H]$ = Total heterogeneity

$$[H]t_{ij} = \begin{cases} 1, \ if \ area \ i \ and \ j \ belong \ to \ same \ region \ k \\ 0, \ otherwise; \end{cases}$$

$$[H]x_i^{kc} = \begin{cases} 1, \ if \ A_i \ is \ assigned \ to \ region \ R_k \ in \ order \ c \\ 0, \ otherwise; \end{cases}$$

Minimize:

$$[H]Z = (-\sum_{k=1}^{n}\sum_{i=1}^{n} x_i^{k0}) * 10^h + \sum_i \sum_{j|j>i} d_{ij}t_{ij}$$

The objective function consists of two parts, one part controls the number of regions and second part controls the total heterogeneity $H(P_p)$. The first part is obtained by adding the number of areas designated as root areas (X_i^{k0}), and the second part is obtained by adding the pairwise dissimilarities between areas assigned to the same region. Since the objective function is formulated as a minimization problem, we multiply the first term by minus one.

These two parts are combined together in such a way that there is an implicit hierarchy where the number of regions comes before the impact of total heterogeneity. We achieve this hierarchy by multiplying the first term by a scaling factor $h = 1 + log(\sum_i \sum_{j|j>i} d_{ij})$. For p regions the objective functions starts at $p * 10^h$. This value increases when we add the total heterogeneity, but h is big enough such that, regardless of the value of heterogeneity, the objective function will never reach $-(p-1) * 10^h$.

The Construction phase has two different parts: region growing and enclave assignment.

During region growing a random area is selected from the set of unassigned areas. Then, this area is added to the region. We check if this newly formed region reaches the minimum threshold value of the spatial regional attribute. We stop growing the new region if the minimum threshold value is met, otherwise, we keep adding unassigned neighbouring areas to the region until it reaches the minimum threshold value. This is repeated until it is not possible to grow any new region that satisfies the minimum threshold value. Areas which are not assigned to any region during growing part are called "enclaves". At the end of growing

Algorithm 1 : Find best feasible solution [2]

A : set of areas,
l : spatially extensive attribute of areas,
W : neighborhoods,
t : threshold constraint on attribute l at region level,
Ψ : set of partitions before enclave assignment,
ϵ : set of enclave areas,
A^u : set of unassigned areas,
ns : Number of solutions

for each processor;i = 1 to ns **do in parallel**
  while $A^u \neq empty$ **do**
    A_k = select at random, one area from A^u
    $A^u = A^u - A_k$
    $R_k = \{A_k\}$, randomly selected area is assigned to region
    building_region = True
    $T = l_k$, value of attribute l in region R_k
    while building_region = True **do**
      if $T \geq t$ **then**
        $\Psi = \Psi \cup \{R_k\}$, add region R_k to partition Ψ
      else
        $N = neighbors \subset A^u$
        if $N \neq empty$ **then**
          A_i = random area from N
          $A^u = A^u - A_i$
          $R_k = R_k \cup A_i$
          $T = T + l_i$
        else
          $\epsilon \cup \{R_k\}$
          building_region = False
        end if
      end if
    end while
  end while
  if $\Psi \neq empty$ **then**
    feasible = True
    while $\epsilon \neq empty$ **do**
      ϵ_i = random area from ϵ
      η = region $\eta \subset \Psi$, that shares border with ϵ_i
      if $\eta \neq empty$ **then**
        R_k = random region $\subset \eta$
        $R_k^n ew = R_k \cup \epsilon_i$
        $\Psi = \Psi - \{R_k\} \cup R_k^n ew$, update region $R_k in \Psi$
        $\epsilon = \epsilon - \epsilon_i$
      else
        feasible = False
      end if
    end while
  else
    feasible = False
  end if
end for
$P_{feasible} = \Psi$, at this point all the areas have been assigned to a region
return $P_{feasible}$

part, the algorithm generates set of growing regions and the enclaves are randomly placed in an enclave queue.

Check floor function available in pysal library in python is used to check if the current growing regions reach the minimum threshold value for the spatially extensive attribute. Input for this function is the list of areas in any given region. It finds out the values for the spatially extensive attribute for all these areas and adds them and compares with the threshold value to check if it is greater than or equal to the threshold value.

```
def check_floor(self, region):
    selectionIDs = [self.w.id_order.index(i) for i in region]
    cv = sum(self.floor_variable[selectionIDs])
    if cv >= self.floor:
        return True
    else:
        return False
```

Figure (4.1) Check floor function from python pysal library to check if a growing region has reached minimum threshold value

A partial solution generated from growing sub-phase will be processed further by an enclave assignment step where we pop the enclave queue and attempt to assign the enclave to a randomly selected neighbouring region. If the selected enclave is not yet contiguous

to an existing region, it is returned to the end of the enclave queue. The process continues until the enclave queue is exhausted. The algorithm thus requires that the adjacency graph formed on the areas is connected.

We can run the python function on multiple processes using the following code. "Pool" will create the number of processes we want to use for execution and then we pass the function to these processes using the map function in pool library. Once all the processes complete the task, all the results are joined and the connection is closed. It can be seen from the below code snippet that we use p.close before p.join, however, it is mandatory to do so as multiprocessing library accepts it only this way.

```
p = Pool(64)
results = p.map(self.th_func,range(initial))
p.close()
p.join()
```

Figure (4.2) Pool function from python multiprocessing library to create and manage multiple processes

Another way to create multiple processes in python is by using process class available in multiprocessing library. Unlike pool, we have full control on the processes we created and we can assign them work as per requirement. First, we create the instance of the process and then we start it. Once all the processes finish their job, we use join operation to synchronize and close all the processes. Following code snippet shows the working example of Process class. The main difference Pool and Process is that in Pool all the work assignment is done automatically, however, in Process, we have to take care of work assigned to each process.

4.0.2 Optimization Phase

In this phase, the best feasible solution from construction phase is optimized by minimizing heterogeneity within the regions. We use multiple processors to perform this task in parallel. Each processor selects a region and all its neighbours which are eligible to move.

```
for i in range(initial):
    for j in range(k):
        p = Process(target=self.initial_solution, args=(i, j, riter, s_candidates,s_regions,s_enclaves,s_lock,))
        p.start()
        processes.append(p)

for p in processes:
    p.join()
```

Figure (4.3) Process function from python multiprocessing library to create and manage multiple processes

Neighbour areas which can be removed from the region while the region still satisfying minimum threshold value and contiguity constraint are eligible to move. Each processor acquires two locks, one on the region where an eligible area is being added and one in the region from where an eligible area is being moved. Then we move the area which minimizes the value of objective function. Once the movement is complete, then both the locks are released and another processor that is ready to move the area from one region to another can acquire the lock and make the change. This process is repeated for all the regions in parallel until there is no such neighbour area which when is moved will further minimize the value of objective function.

First, we find the neighbours of all the areas in a region. After that, we check for each neighbour that if we remove it from the current region, whether or not the region can maintain its contiguity and also satisfies the minimum threshold criteria. All the neighbour areas which can be removed from the region while the region still satisfying minimum threshold value and contiguity constraint are eligible to move.

Following code snippet shows how we are finding the area which when moved from its current region, further minimizes the objective function value. For each eligible area, we try to move it from its current region to the region assigned to a particular processor. After that, we calculate the objective function value for the new solution and compare it with an already available solution to check if it is smaller than the current value. We take the area which gives us the minimum value of objective function.

Algorithm 2 : Optimize $P_{feasible}$ to minimize heterogeneity [2]

A : set of areas,
l : spatially extensive attribute of areas,
t : threshold constraint on attribute l at region level,
Ψ : set of partitions,
R : set of regions in Ψ,
p : number of regions

```
swapping = True
while swapping = True do
    moves = 0 for each processor: i = 1 to p do in parallel

        R_k = random region in Ψ
        N_Bk = neighbors of all the areas of region R_k
        for N_i in N_Bk do
            R_m = region that contains neighbor area N_i
            if l_m ≥ t and doesn't break contiguity then
                N_e = N_e ∪ N_i, set of areas which are eligible for movement from region R_m to R_k
            end if
        end for
        if N_e ≠ empty then
            N_ei = Area from N_e that minimizes the objective function when moved from region R_m to R_k
            if N_ei ≠ empty then
acquire lock on R_k and R_m
                R_k = R_k ∪ N_ei
                R_m = R_m − N_ei
                move += 1
release lock on R_k and R_m
            end if
        end if
    end for
    if moves = 0 then
        swapping = False
    end if
end while
P_p = Optimized solution with minimum heterogeneity
return P_p
```

```
# find the best local move
if candidates:
    nc = len(candidates)
    moves = np.zeros([nc, 1], float)
    best = None
    cv = 0.0
    try:
        for area in candidates:
            current_internal = shared_list[seed]
            current_outter = shared_list[self.area2region[
                area]]
            current = self.objective_function([current_internal, current_outter])
            new_internal = copy.copy(current_internal)
            new_outter = copy.copy(current_outter)
            new_internal.append(area)
            new_outter.remove(area)
            new = self.objective_function([new_internal,
                                           new_outter])
            change = new - current
            if change < cv:
                best = area
                cv = change
    except:
        pass
```

Figure (4.4) Code snippet to find the areas which minimizes the objective function value

Once we find the area which minimizes the objective function value, then we acquire the lock on both the regions from which area is being moved and also the region to which area is being moved. We acquire the two locks based on the index of the regions. We acquire the lock first on the region with smaller index and then on the region with the larger index. This is done to avoid the race condition. Once the area is moved, then we release the locks on both the regions. Multiple locks are used to reduce the lock overhead.

Let us explain the algorithm with the help of an example. Assume we have the feasible solutions(P_p) with three regions.

Region R_1, R_2 and R_3 are sent to processors P_1, P_2, P_3. P_p is a shared memory list

```
if best:
    # make the move
    lcksttm = time.time()
    old_region = self.area2region[best]
    lock1 = shared_lock[old_region]
    lock2 = shared_lock[seed]
    if old_region < seed:
        lock1.acquire()
        lock2.acquire()
    else:
        lock2.acquire()
        lock1.acquire()
    try:
        ls = copy.copy(shared_list[self.area2region[
        neighbor]])
        fv1 = self.check_floor(ls)
        if fv1:
            so_list = shared_list[old_region]
            so_list.remove(area)
            shared_list[old_region] = so_list
            self.area2region[area] = seed
            ss_list = shared_list[seed]
            ss_list.append(area)
            shared_list[seed] = ss_list
            m_made += 1
            smoves.value += 1
    except:
        pass
    if old_region < seed:
        lock2.release()
        lock1.release()
    else:
        lock1.release()
        lock2.release()
```

Figure (4.5) Code snippet to show movement of area from one region to another by acquiring the locks in respective regions

which can be updated by all the processors. Now on processor P_1, first we will find all the neighbour areas of R_1, which are eligible to move from their current region. Assume, $N_{R_1}(A_4, A_5, A_6)$ is the list of neighbour areas which are eligible to move from their current region.

Now, we will find the area from N_{R_1}, which then moving to R_1, will further minimize the objective function. Assume, A_4 is the area which minimizes the current value of objective function.

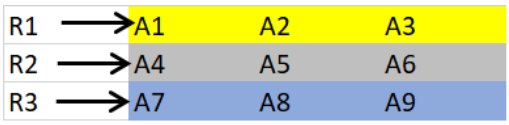

Figure (4.6) Feasible solution after the construction phase with three regions

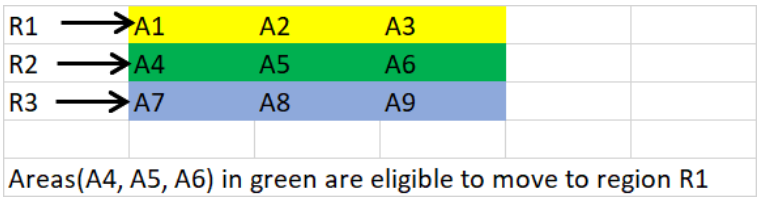

Figure (4.7) Areas eligible to move to region R1

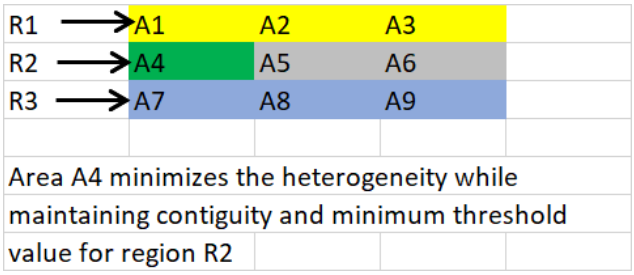

Figure (4.8) Area the minimizes the objective function value

Processor P_1 will acquire lock on region R_1 and R_2 and move the area A_4 from region R_2 to R_1. Once, the movement is complete, P_1 will release the lock on region R_1 and R_2. Now, other processors can acquire the lock and update the list. This is repeated on all the processors concurrently.

2

[2]These codes can be downloaded the following link: https://github.com/vineysindhu/parallel-max-p-region. Also the code for sequential max-p region is available at the following link: https://github.com/pysal

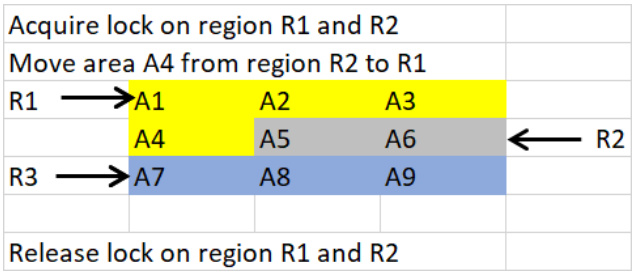

Figure (4.9) Area A4 moved from region R2 to region R1

PART 5

EXPERIMENTS

5.1 Performance Evaluation

In this section first, we briefly explain the system we used for our experiments and then presents our results.

5.1.1 Experimental Setup

All the experiments employ a compute node that has 16-core Intel Xeon CPUE5-2650 V2 CPU running at a clock speed of 2.60 Ghz with 64GB main memory. All the code for this experiment is written in the Python programming language. The *multiprocessing* library available in Python is used to connect to multiple processors. Also, we have used a compute node that has 64-core AMD Opteron(TM) Processor 6272 with 64GB main memory for comparison against 16-core Intel.

5.1.2 Experiments and Results

In this section, we evaluate the performance of our algorithm. We have created regular lattices using *numpy* library available in python. Three different sets of lattices (20×20, 33×33, 55×56) are created for the evaluation. We considered two areas to be neighbours if and only if they share a line (rook). We have also used three different thresholds for spatially extensive attribute i.e. 25, 100 and 300. We have done several experiments to evaluate the performance parallel max-p region algorithm. All the experiments are done to explore either the parallel efficiency or the parallel synergy or both in the exponential space. Parallel efficiency means to reduce running time of the algorithm by using parallel processing. Parallel synergy means exploring exponential space by using parallel processing and trying to find the best feasible solution which gives us the minimum value of objective

function. Our experiments show that we can achieve both efficiency and synergy by using multiple processes. We have divided the results section further into two parts, Evaluating the efficiency and Exploring synergy.

5.1.2.1 Parallel Efficiency: We did several experiments to evaluate the performance of our algorithm in terms of running time efficiency.

Average running time comparison of parallel and sequential max-p region algorithm: First, we have compared the running time of our algorithm with that of sequential max-p implementation presented in [2]. Table 5.1 presents the average running time in seconds for different sets of lattices and different threshold values for both the sequential and the parallel max-p algorithm. All the timings are taken on Intel 16-core system by using 64 processes for the construction phase and for the optimization phase we have used the number of regions as the number of processes. As shown in table 5.1, the speed-up is increasing with lattice size, because smaller lattices form fewer regions. As a result, we are not able to take full advantage of multiprocessing capabilities of the system for small problems. We observed that running time for the optimization phase of a 20x20 lattice for a threshold value of 300 is almost 0, because only one region will be created for $n = 400$ and threshold 300, hence, there will be no optimization phase.

Average running time comparison of the construction phase of parallel max-p region on 16-core Intel processor to that of 64-core AMD Opteron processor: We have also compared the running time of the construction phase of parallel max-p region on 16-core Intel processor to that of 64-core AMD Opteron processor. This comparison is done for two lattices 20x20 and 55x56 with a threshold of 100 and 300 by using the different number of processes to find best feasible solution i.e. 8, 16, 32, 64, 128, 256, 512. Figure 5.1 presents the charts which show that running time of algorithm decreases with increase in the number of processes up to a certain limit and after that, it starts to increase again. It can be interpreted from the graph that for 16-core Interprocessor threshold to decrease

Lattice		Sequential max-p			Parallel max-p			Speedup
		Construction	Optimization	Total	Construction	Optimization	Total	
20×20 (n = 400)	Threshold = 25	10.57	3.33	13.90	1.76	2.43	4.19	3.31
33×33 (n = 1,056)		64.04	16.46	80.50	9.29	8.61	17.90	4.50
56×56 (n = 3,080)		472.89	120.44	593.33	67.60	29.05	96.65	6.14
20×20 (n = 400)	Threshold = 100	56.01	23.28	79.29	5.68	13.25	18.93	4.19
33×33 (n = 1,056)		395.80	101.64	497.44	35.18	27.04	62.22	7.99
56×56 (n = 3,080)		3117.80	1573.83	4691.63	271.32	109.86	381.18	12.30
20×20 (n = 400)	Threshold = 300	154.99	0.01	155	14.86	0.16	15.02	10.31
33×33 (n = 1,056)		1180.36	774.13	1954.49	104.98	83.36	188.34	10.38
55×56 (n = 3,080)		9814.39	6222.92	16037.31	829.53	320.85	1150.38	13.94

Table (5.1) Average running time comparison for sequential max-p and parallel max-p for three different thresholds.

running time is 64 and for 64-core AMD Opteron threshold is 128. Once this threshold is reached, running starts to increase again. Hence, it is clear from the results that we should use 64 processes for 16-core Intel system and 128 processes for 64-core AMD system to get the best running time. We also analysed the CPU usage for each process for the different number of processes. For 16-core Intel system, each process can utilize 100 percent of CPU capacity for up to 16 processes and after that, it decreases to 50 percent for 32 processes. It becomes stable at 17 percent after 128 processes. For 64-core AMD system, each process can utilize 100 percent of CPU capacity for up to 64 processes and starts decreasing after that. It becomes stable at 70 percent after 128 processes.

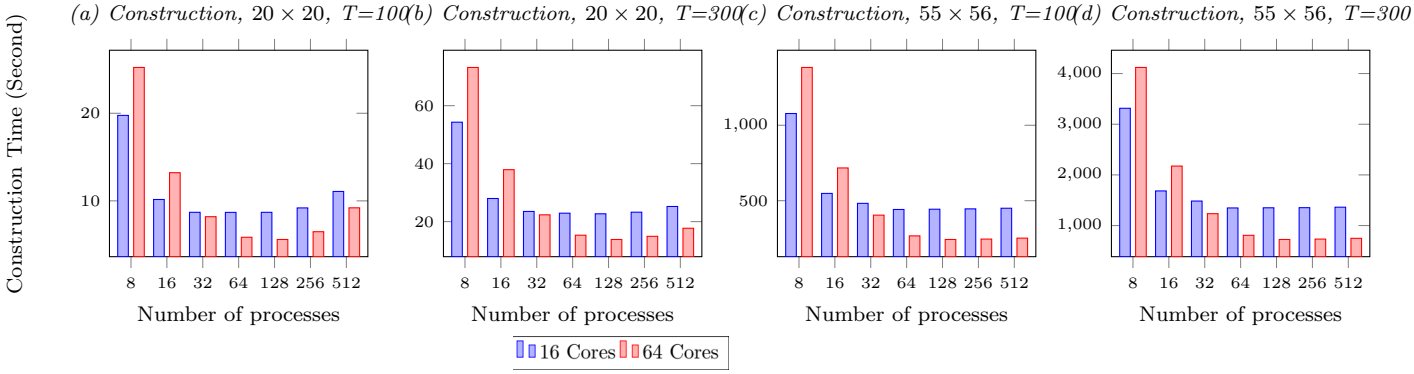

Figure (5.1) Average running time of construction phase on 16-core Intel vs 64-core AMD Opteron

Single lock vs multiple locks average running time comparison of the optimization phase: We started by using single lock on the list of regions in optimization phase. This lead to a lock overhead problem. In order to reduce the lock overhead, we have used multiple locks i.e. one lock for each region. After using multiple locks, we can reduce lock overhead time. Table 5.2 presents the running time of the optimization phase for different lattices and different thresholds. It can be seen that after using multiple locks, the running time of the algorithm is reduced.

Relation between number of areas moved and decrease in objective function value: We have also analysed the effect of movement of areas from one region to another

Lattice	threshold = 100		threshold = 300	
	Single Lock	Multiple Locks	Single Lock	Multiple Locks
20x20 (n = 400)	17.47	13.25	0.29	0.16
33x33 (n = 1,056)	41.53	27.04	133.76	83.36
55x56 (n = 3,080)	156.47	109.86	639.63	320.85

Table (5.2) Single lock vs multiple locks average running time for optimization phase in seconds

on objective function value. The decrease in objective function value is directly proportional to the movement of areas from one region to another in each iteration of optimization phase. The more areas move from one region to another, the more the decrease in objective function value. Table 5.3 presents the decrease in objective function value with the decrease in the number of areas moved from one region to another. There are few outliers also, for example in iteration 3 number of areas moved decreased, however, decrease in objective function value increased.

Since we are doing the filtering of the list of areas and regions multiple times in our code, we decided to run a small experiment to explore various methods to filter the list and see which one gives the best performance. We tried the below methods and found that list compression method is the fastest of all the methods to filter the python lists.

Running time using set(a) - set(b) : 2.86102294921875e-06 seconds

Code - list(set(neighbors)-set(enclaves))

Running time using list compression : 2.384185791015625e-06 seconds

Code - [neighbor for neighbor in neighbors if neighbor not in enclaves]

Running time using One of the lists as set : 3.337860107421875e-06 seconds

Code - [neighbor for neighbor in neighbors if neighbor not in set(enclaves)]

Running time using lambda function : 5.0067901611328125e-06 seconds

Code - list(filter(lambda x: x not in set(enclaves), neighbors))

Iteration	Number of areas moved	Decrease in Objective Function Value	Percentage Decrease
1	12	1.59	0.026
2	12	1.58	0.026
3	11	1.62	0.028
4	11	1.14	0.020
5	10	0.94	0.017
6	11	1.38	0.025
7	10	0.63	0.012
8	7	0.29	0.005
9	6	0.54	0.010
10	4	0.15	0.003
11	2	0.19	0.004
12	4	0.25	0.005
13	4	0.09	0.002
14	1	0.004	0.000
15	1	0.004	0.000
16	0	0	0.000

Table (5.3) Decrease in objective function value with the decrease in areas moved in optimization phase for 20x20 lattice with threshold = 25

5.1.2.2 Exploring synergy or accuracy of the parallel max-p region algorithm: We did the following experiments to explore the synergy of the parallel max-p region algorithm.

Analysis of top solutions after the construction phase: First, we analysed the top feasible solutions that we get after construction phase of the algorithm. Table 5.4, 5.5 and 5.6 presents the analysis for 20x20, 33x33 and 55x56 lattices with threshold 25 respectively. All the solutions present in the tables give the maximum number of regions for their respective lattice size. We can see from the table 5.4 that our algorithm will select the solution with minimum objective function value after construction phase and tries to optimize it. Hence, Solution 1 will be chosen for optimization. However, when we analysed other top solutions, we found that Solution 5, does not have minimum objective function

value after construction phase, but it gives us best results after optimization phase. Similarly, in the table 5.5, Solution 3 gives the best results after optimization, however, it will not be selected at first by the algorithm. Also, in the table 5.6, Solution 5 gives the best results.

Solutions	Objective function value after construction phase	Objective function value after optimization phase
Solution 1	63.13	54.28
Solution 2	63.92	54.14
Solution 3	63.27	54.89
Solution 4	63.60	55.93
Solution 5	63.32	51.49

Table (5.4) Exploring synergy in top solutions for 20x20 lattice with threshold = 25

Solutions	Objective function value after construction phase	Objective function value after optimization phase
Solution 1	173.94	148.38
Solution 2	173.72	148.36
Solution 3	174.49	145.25
Solution 4	174.39	151.35

Table (5.5) Exploring synergy in top solutions for 33x33 lattice with threshold = 25

Comparison of change in objective function value of sequential and parallel max-p region algorithm: We have also analysed the change in objective function value during optimization phase and its relation to the number of areas moved from one region to another. Table 5.7 presents the change in objective function value and the number of areas moved in optimization phase for both sequential max-p and parallel max-p algorithm. As shown in table 5.7, the decrease in objective function value is directly proportional to the number of areas moved from one region to another. The parallel max-p algorithm can perform similarly to the sequential max-p algorithm in terms of decrease in the objective

Solutions	Objective function value after construction phase	Objective function value after optimization phase
Solution 1	498.15	410.82
Solution 2	493.82	411.39
Solution 3	497.09	413.88
Solution 4	497.88	415.94
Solution 5	497.51	408.86

Table (5.6) Exploring synergy in top solutions for 55x56 lattice with threshold = 25

function value. However, there are few outliers where sequential max-p algorithm performs much better than the parallel max-p.

Analysis of objective function value by growing regions in parallel for the construction phase: We grew regions in parallel to explore synergy in the exponential space. Usually we create 100 solutions and each solution in parallel, however, in each solution, the regions are grown in a single process. Here, we created 20 solutions and within each solution, we grew the regions in parallel. We assigned a total of 60 processes, which means 3 processes to each solution. Each solution will use these processes and try to grow maximum possible regions based on lattice size and threshold. For example, for 20×20 lattice and threshold of 25, maximum possible regions are 16. These 3 processes will try to grow 16 regions in parallel by seeding from 3 different areas at a time. If any of the regions is not able to reach the minimum threshold value, all the areas of that region will be added to enclaves. Since we did it in parallel, we were able to explore the space in parallel and get the results similar to that we got by generating 100 solutions. For example, we got the maximum number of regions as 13 and objective function value as 51.52 for 20×20 lattice with the threshold of 25 by generating 20 solutions in parallel. Hence, by exploring space in parallel, we can get synergy with less number of solutions as compared to exploring it in sequence.

Lattice		Sequential max-p				Parallel max-p			
		Initial	Final	Decrease	Areas Moved	Initial	Final	Decrease	Areas Moved
20×20 (n = 400)	Threshold = 25	61.99	50.64	11.35	130	62.05	51.15	10.90	115
33×33 (n = 1,056)		172.68	141.00	31.68	388	172.20	145.25	26.95	269
56×56 (n = 3,080)		495.41	409.91	85.50	1069	494.98	408.84	86.14	917
20×20 (n = 400)	Threshold = 100	64.10	58.21	5.89	79	63.78	57.41	6.37	86
33×33 (n = 1,056)		177.21	160.73	16.48	245	177.39	157.44	19.95	263
56×56 (n = 3,080)		507.28	439.13	68.15	1022	508.20	452.51	55.69	721
20×20 (n = 400)	Threshold = 300	64.82	64.82	0	0	64.82	64.82	0	0
33×33 (n = 1,056)		178.41	165.95	12.46	219	178.34	168.55	9.79	151
55×56 (n = 3,080)		510.42	461.91	48.51	745	511.75	469.81	41.94	702

Table (5.7) Change in objective function value and number of areas moved in optimization phase for sequential max-p and parallel max-p for three different thresholds.

PART 6

CONCLUSION AND FUTURE WORK

In this thesis, we present a parallel implementation of a constrained clustering problem, where our aim is to cluster areas into a set of the maximum number of homogeneous regions based on a minimum threshold value of a spatially extensive attribute and do so in a computationally efficient manner. We propose a heuristic solution to this problem which can help us achieve our goal and also aide in minimizing aggregation bias. This algorithm can help us to solve many real-life problems. For example, police districting needs headquarters to be allocated in all the territories. Hence areas can be aggregated into regions such that regions are homogeneous in terms of crime types, and each region contains a minimum potential to handle emergency calls. Once the regions are designed, we can decide on the best location for headquarters.

Future work will include efficiently growing regions for every solution in parallel so that we can achieve parallel efficiency also along with parallel synergy. Also, we can implement this algorithm using MPI in python to take advantage of multiple nodes rather than just one node. Currently, we try to find the solution in the search space by seeding from the random areas. In future, we can try to search the exponential space by seeding from the pre-defined areas, which will decrease the randomness and will give us more control for exploring synergy.

REFERENCES

[1] Bodin, L. "A districting experiment with a clustering algorithm", *Annals of the New York Academy of Sciences(1973)* pp. 209-214.

[2] Duque, J.C., Anselin, L., Rey S.J., "The Max-p-Region Problem", *Journal of Regional Science(2012)*, pp. 397-419.

[3] Li, W., Church, R.C., Goodchild, M.F. "The p-Compact-regions Problem", *Geographical Analysis(2014)* pp. 250273.

[4] Laura, J., Li, W., Rey, S.J., Anselin, L. "Parallelization of a regionalization heuristic in distributed computing platformsa case study of parallel-p-compact-regions problem", *International Journal of Geographical Information Science(2015)* pp. 536-555.

[5] She, B., Duque, J.C., Ye, X. "The Network-Max-P-Regions model", *International Journal of Geographical Information Science(2017)* pp. 962-981.

[6] Hansen, P., Jaumard, B., Meyer, C., Simeone, B., and Doring, V. "Maximum split clustering under connectivity constraints", *Journal of Classification(2003)*, pp. 143-180.

[7] Prasad, S.K., McDermott, M., Puri, S., Shah, D., Aghajarian, D., Shekhar, S., Zhou, X. "A vision for GPU-accelerated parallel computation on geo-spatial datasets", *SIGSPATIAL Special(2014)*, pp. 19-26.

[8] Prasad, S.K., McDermott, M., Puri, S., Shah, D., Aghajarian, D., Shekhar, S., Mokbel, M., Rey, S., Xe, Y., Vatsavai, R.R., Wang, F., Liang, Y., Vo, H., Wang, S. "Parallel Processing over Spatial-Temporal Datasets from Geo, Bio, Climate and Social Science Communities: A Research Roadmap", *IEEE International Congress on Big Data (BigData Congress)(2017)*

[9] Aghajarian, D., Puri, S., Prasad, S.K. "GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform", *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems(2016)*

[10] Aghajarian, D., Prasad, S.K. "A Spatial Join Algorithm Based on a Non-uniform Grid Technique over GPGPU", *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems(2017)*

[11] Prasad, S.K., McDermott, M., Puri, S., Xi, h. "GPU-based Parallel R-tree Construction and Querying", *IEEE International Parallel and Distributed Processing Symposium Workshop(2015)*

[12] Maravalle, M. and Simeone, B. "A spanning tree heuristic for regional clustering", *Communications in Statistics-Theory and Methods(1995)*, pp. 625-639.

[13] Wise, S., Haining, R., and Ma., J. ""Regionalisation tools for exploratory spatial analysis of health data", *Recent developments in spatial analysis: Spatial statistics, behavioural modelling, and computational intelligence Edited by Manfred M. Fischer and Arthur Getis, Springer, New York(1997)*

[14] Fischer, M. "Regional taxonomy. A comparison of some hierarchic and non-hierarchic strategies", *Regional Science and Urban Economics(1980)*, pp. 503-537.

[15] Openshaw, S. "A regionalization program for large data sets", *Computer Applications(1973)*, pp. 136-147.

[16] Openshaw, S. and Rao, L. "Algorithms for reengineering 1991 census geography", *Environment and Planning A(1995)* pp. 425-446.

[17] Webster, R. and Burrough, P. "Computer-based soil mapping of small areas from sample data II. Classification smoothing", *European Journal of Soil Science(1972)* pp. 222-234.

[18] Murray, A. and Shyy, T. "Integrating attribute and space characteristics in choropleth display and spatial data mining", *International Journal of Geographical Information Science(2000)* pp. 649-667.

[19] Assuncao, R. M., Neves, M. C., Camara, G., and Freitas, C. D. C. "Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees", *International Journal of Geographical Information Science(2006)* pp. 797-811.

[20] Zoltners, A. and Sinha, P. "Sales territory alignment: A review and model", *Management Science(1983)* pp. 1237-1256

[21] Williams, J. "Political redistricting: A review", *Papers in Regional Science(1995)* pp. 13-40.

[22] Laura, J., Rey S.J., "Cooperative Spatial Regionalization in a Distributed Memory Environment"

[23] Margules, C., Faith, D., and Belbin, L. "An adjacency constraint in agglomerative hierarchical classifications of geographic data", *Environment and Planning A(1985)* pp. 397-412.

[24] Maravalle, M., Simeone, B., and Naldini, R. "Clustering on trees", *Computational Statistics and Data Analysis(1997)* pp. 217-234.

[25] Lankford, P. "Regionalization: Theory and alternative algorithms", *Geographical Analysis(1969)* pp. 196-212

[26] Openshaw, S. "A geographical solution to scale and aggregation problems in region-building, partition and spatial modeling", *Transactions of the Institute of British Geographers(1977)* pp. 459-472.

[27] Perruchet, C. "Constrained agglomerative hierarchical classification", *Pattern Recognition(1983)* pp. 213-217.

[28] Openshaw, S. "Optimal zoning systems for spatial interaction models", *Environment and Planning A(1977)* pp. 169-184.

[29] Leandro, M., Jose, V., Flavia, B. "RegK-Means: A clustering algorithm using spatial contiguity constraints for regionalization problems", *Brazilian Conference on Intelligent Systems(2017).*

[30] Wang, F., Guo, D., McLafferty, S. "Constructing geographic areas for cancer data analysis: A case study on late-stage breast cancer risk in Illinois", *Applied Geography(2012)* pp. 111.

[31] Duque, J.C., Raul, R., Jordi, S. "Supervised regionalization methods: A Survey", *Research Institute of Applied Economics(2006).*

[32] Nurmala, N., Purwarianti, A. "Improvement of Fuzzy Geographically Weighted Clustering-Ant Colony Optimization using Context-Based Clustering", *International Conference on Information Technology Systems and Innovation(2015).*

[33] McMahon, G., Gregonis, S., Waltman, S., Omernik, J., Thorson, T., Freeouf, J., Rorick, A., Keys, J. "Developing a spatial framework of common ecological regions for the conterminous united states", *Environmental Management(2001)* pp. 293316.

[34] Yuan, S.,Tan, P.N., Cheruvelil, K.S., Collins, S.M., Soranno, P.A. "Constrained Spectral Clustering for Regionalization: Exploring the Trade-off between Spatial Contiguity and Landscape Homogeneity", *Data Science and Advanced Analytics(2015).*

[35] Hansen, P., Jaumard, B., Meyer, C., Simeone, B., and Doring, V. "Maximum split clustering under connectivity constraints", *Journal of Classification(2003)* pp. 143-180.

[36] Duque, J., Church, R., and Middleton, R. "The p-regions problem", *Geographical Analysis, forthcoming(2010)*

[37] Cliff, A. and Hagget, P. "On the efficiency of alternative aggregations in region-building problems", *Environment and Planning(1970)* pp. 285-294

[38] Cliff, A., Haggett, P., Ord, J., Bassett, K., and Davies, R. "Elements of Spatial Structure: A Quantitative Approach", *Cambridge University Press, New York(1975)*

[39] Ferligoj, A. and Batagelj, V. "Clustering with relational constraint", *Psychometrika(1982)* pp. 413-426.

[40] Byfuglien, J. and Nordgard, A. "Region-building: A comparison of methods", *Norwegian Journal of Geography(1973)* pp. 127-151.

[41] Bunge, W. "Gerrymandering, geography, and grouping", *Geographical Review(1966)* pp. 256-263.