A Parallel Algorithm and Implementation to Compute Spatial Autocorrelation (Hotspot) Using MATLAB

Mingjun Li
Marquette University

A PARALLEL ALGORITHM AND IMPLEMENTATION TO COMPUTE SPATIAL AUTOCORRELATION (HOTSPOT) USING MATLAB

By

Mingjun Li

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
In Partial Fulfillment of the Requirements for
The Degree of Master of Science

Milwaukee, Wisconsin

May 2020

ABSTRACT

A PARALLEL ALGORITHM AND IMPLEMENTATION TO COMPUTE SPATIAL AUTOCORRELATION (HOTSPOT) USING MATLAB

Mingjun Li

Marquette University, 2020

Being a spatial autocorrelation visualization tool in recent years, hotspot is often used in various fields, such as disease analysis, crime analysis, and weather conditions analysis and prediction in a certain area. Most of the research in hot spot analysis is in applying the concept to a variety of fields and to gain insights on the statistical significance prevalent in the clustering of data. Only a few of them discussed the efficiency and optimization of the algorithm. Commonly, these kinds of analyses would be based upon a huge dataset about space and time, and the conventional algorithm would take too much time to get the results. This paper mainly discusses whether the algorithm can be processed in parallel with MATLAB and how to further optimize the algorithm to shorten the calculation time and obtain accurate outcomes faster. I will use the toolbox ‘parpool’ in MATLAB on a multi-core node to parallelize the conventional algorithm, and then take advantage of the basic idea of the 'R-tree' to further optimize the parallel algorithm. In the end, the results are satisfactory, because the conventional serial algorithm can be parallelized in MATLAB, and the time consumption was saved about five times compared to the original algorithm. When the algorithm was further optimized, its time consumption is saved about ten times. This paper will be helpful in saving time when doing similar computations and analyses in the future.

ACKNOWLEDGMENTS

Mingjun Li

Through this period of effort, my master thesis was finally completed, which meant that the master’s life was coming to an end. At the graduate level, I benefit a lot. Apart from my own efforts, it is inseparable from the care, support, and encouragement of teachers, classmates, and friends.

In the process of writing this thesis, I would like to express my deepest appreciation to my academic advisor, Dr. Satish Puri, who has put in a lot of effort, his guidance through each stage of the process was fantastic which inspired me a lot.

Furthermore, I also want to thank all the teachers who have given me great care and support during my studies, as well as my classmates and friends who cared about me. Teachers especially the committee members, Dr. Daniel Rowe, Dr. Praveen Madiraju, and Dr. Debbie Perouli, who have given valuable comments and advice on my thesis in order to improve it. Friends especially Yang Jie and Cui Tao, helped me resolve the problems I encountered when I dealt with the original data and developed the algorithm.

TABLE OF CONTENTS

INTRODUCTION

1. Parallel Programming

Looking back at the entire human history, people have always tried to make breakthroughs in almost all fields that we can imagine, just like the motto of the Olympic games says ‘faster, higher and stronger’. That's why we have the modern civilization that people now see. Although the computer has not been utilized for a long time, it has also followed the trend of history, constantly improved, upgraded, and breakthrough, which allows it becoming faster and more stable in performance.

From 1986 to 2002, the performance of integrated circuits increased, on average, 50% per year [27]. Since 2002, however, due to physical limitations, the growth rate has dropped to 20% per year, which seems to be an acceptable growth rate, but its bottleneck almost there. Therefore, from 2005, the majority of manufacturers began to produce multicore processors in an attempt to keep increasing the speed and performance of operations by using parallel computing methods [26].

The bad news of this attempt is that traditional programming languages, such as C and C ++, cannot directly convert serial programs into parallel programs, so parallel programming came into being. [26].

2. Ways to do Parallel Programming

There are many ways to write a parallel program, but all the different approaches have the same basic idea, which is to partition the job to be done into separate cores for calculation, each core takes a part of the entire job. For example, if you are walking on the street under a heavy sun without wearing a hat and a pair of sunglasses, then you could wear the hat and sunglasses one by one with one hand, here we assume wearing each of them would take you one unit of time, then to finish wearing both of them would take you two units of time. However, you can also use both of you left hand and right hand to do wearing them within in just one unit of time. This is a simple example to illustrate how to separate the job into multiple workers. In this example, the combination of wearing the hat and wearing that pair of glasses is the job need to be done, and we can partition this job into two parts which are wearing the hat and wearing the pair of sunglasses, your left hand and right hand are two workers that can do the job parallelly, each hand takes a part of the entire job.

There are two widely used concepts of partitioning the job: task-parallelism and data-parallelism. As the name suggests, it is to divide tasks or data into separate cores to run [26]. The difference is that the task-parallelism is to partition the task and then finish each of the part of partitioned task in parallel. For example, let us assume there are two chefs A and B make lunch for ten people, each lunch contains a hamburger and several fries, if chef A cooks the burgers and chef B cooks the fries, then this is task-parallelism since the whole task is to cook burgers and fries, each of them deals with a part of the task. On the other hand, data-parallelism is to use a same processing method to process different data or different parts of a dataset in parallel. In the example above, if chef A cooks five shares and chef B cooks another five shares, this is data-parallelism.

OpenMP is one of the useful methods to do parallel programming, it is a standard for running in parallel on a shared memory system by adding some OpenMP compilation instructions in the source code and calling OpenMP library functions [28].

It is good because the programmers only need to tell the compiler how many workers they want that to do the job at the same time, the compiler will automatically make a schedule and assign the job to each worker. The programmers don't need to worry about how the workers finish the job [13].

MPI is another useful method to do parallel programming, it is the abbreviation of Message-Passing Interface. It is not a brand-new programming language but defines a library of functions which can be called from C, C++, and Fortran programs. It is mainly used on distributed-memory systems. As we know, each processor of distributed-memory systems has its own independent memory. In general, the MPI works as the following way, let’s say we have a processor T, it divides the whole data into a number of parts as even as possible, and then sends the divided data to other processors. After all the other processors receive the data, they execute the same program on their own part of data, and then send the result back to processor T. T receives all the sub results, and then combine them to get the final results.

The Message-Passing model has two advantages: (i)More able to adapt to many architectures; (ii)Available on almost all parallel machines in C and Fortran.

Moreover, we also have Map-Reduce which is one of the core components of the Hadoop ecosystem: a distributed computing framework. It is mainly used to process simple and massive data in large-scale machine clusters, such as processing one million documents on ten thousand computers.

The MapReduce has three main steps: Map, Shuffle, and, as hinted by the name, Reduce. The Map stage first divides the documents, then assigns the divided documents to different machines (cores/nodes), and then each machine extracts the user-defined keywords in the documents and counts how many times they appeared. During the Map phase, there is no communication allowed. Shuffle stage is to sort these keywords. And the Reduce stage is to take the results from the map stage, and then group by each keyword together to get the final result that the programmers need.

But MapReduce has three disadvantages: (i)It has limited expressive power. No matter how complicated the operation is, it is only highly abstracted into two functions which are Map and Reduce. But some problems cannot be solved simply by using Map and Reduce, as a consequence, people will have a hard time when dealing with such problems; (ii) The I / O procedures pass through the disk every single time, and this is very time-consuming especially doing loops; (iii)High latency. The Reduce stage can be performed only after all the Map tasks completed, which is difficult for multi-stage tasks.

Spark, it is but a computing framework, and, essentially, the basic computing model also belongs to MapReduce. In another word, it is a reinforcement version of MapReduce, but it has its own unique and valuable thing which is Resilient Distributed Dataset (RDD) [34]. It is the immutable distributed collection of objects, each dataset in RDD would be divided into logical parts, and each part can be computed on different nodes of the cluster. The RDD can be created from an input data, from transformations of other RDDs, more than that, it also supports actions. Mostly due to the RDDs, the Spark fully draws on the advantages of MapReduce and overcomes its shortcomings to make operations richer, which not only include Map and Reduce, but also has ‘groupBy’, ‘join’, and ‘filter’, etc. So, it provides greater expressiveness.

It also has higher efficiency. It only accesses the disk when the data is read for the first time and the result is stored for the last time. All other intermediate processes are completed in memory. This reduced lots of I / O time loss.

What is more, there are some other good methods to do parallel programming, like Pthread, GPU, and so on, even some programming software comes with excellent toolboxes to do parallel programming, such as MATLAB's ‘Parfor’ toolbox which is I applied on the Everest in this paper.

Everest is a computer with shared memory nodes. It has 36 CPU cores and each of them is a multi-core CPU of Intel Xeon E5-2695 with a 45MB cache and a base frequency of 2.10GHz. The total memory in the machine is 512 GB.

Matlab is an abbreviation of Matrix Laboratory. It is an advanced technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computing. It has a lot of toolboxes, I use ‘Parpool’ in this paper, which is to create parallel pool on the default cluster with a number of workers for running parallel language features. And the language feature I use is ‘Parfor’, it allows to execute for-loop iterations in parallel on workers. The parfor-loop must be independent, that means one loop iteration cannot depend on a previous iteration. No matter how many loops there will be, the parfor-loop must be the outer loop.

3. Spatial Autocorrelation & Hotspots

Statistically, the correlation can tell whether there is a connection between changes in two or more events (statistics variables). If the value of an event goes higher because the value of another event goes higher, then we consider these two events have a positive correlation. On the other hand, if one goes higher because another one goes lower, then we consider these two events have a negative correlation. For example, the yield of a certain fruit is often related to local soil and weather conditions. If the value of a variable at a certain location is high (e.g. the soil is always very fertile) and the value of another variable at the same location is also high (e.g. the weather is always pleasant), then the correlation between these two variables is positive. On the other hand, if one variable has a high value while the other variable has a low value, then it is a negative correlation [1]. But if we analyze the same attribute variable on different observations (such as the amount of an element in the soil in different regions), it is called spatial autocorrelation. Spatial autocorrelation has the similar property, it is an important form of spatial dependence, and there are some valuable applications in the field of geo-statistics. It refers to the correlation between the spatial locations of the research objects.

Consequently, the so-called spatial autocorrelation is to analyze the characteristics of the spatial distribution of the spatial unit and its surrounding units, through statistical approaches to calculate the degree of spatial correlation based upon some kind of specific value of interested variable.

The Tobler’s First Law of Geography points out that “everything depends on everything else, but closer things more so [4]”. We consider it as that closer locations are more correlated, further apart locations are less correlated. If something moves far enough, they are not correlated.

Based on that understanding, I think that the positive spatial autocorrelation has the similar values in neighboring locations occurred more frequently than for spatial randomness. Like values can be either high or low, which are corresponding to hot spots and cold spots respectively. Otherwise, a negative spatial autocorrelation has dissimilar values (i.e. high VS low) in neighboring locations occur more frequently than for spatial randomness.

Moran’s I Index and Geary ’s C Ratio are currently the most popular methods for measuring spatial autocorrelation. The difference between the two methods is, in simple terms, the range of values obtained. The result of Moran's I computation ranges from negative one to positive one. Result greater than zero indicates clustering, and less than zero indicates dispersion. On the other hand, the result of Geary's C ranges from zero to two, which says if the result is closer to zero, it means clustering, if closer to two means more dispersion, otherwise, if closer to one means more random. But these two methods are concentrating on the statistical analysis of the entire study area [2]. They can only indicate clustering rather than distinguishing hotspots and / or cold spots.

Motivated by this shortcoming, Getis and Ord introduced the local statistics, Getis-Ord Gi* statistics [16], which enables us to detect the trace of spatial association that may not be evident when using global statistics [3]. What is more, Getis-Ord Gi* statistics are useful to identify spatial anomalies [25], cold spots and hot spots, where their values are significantly low or high and be surrounded by other low or high values as well [2] like Figure 1 [37] shows.

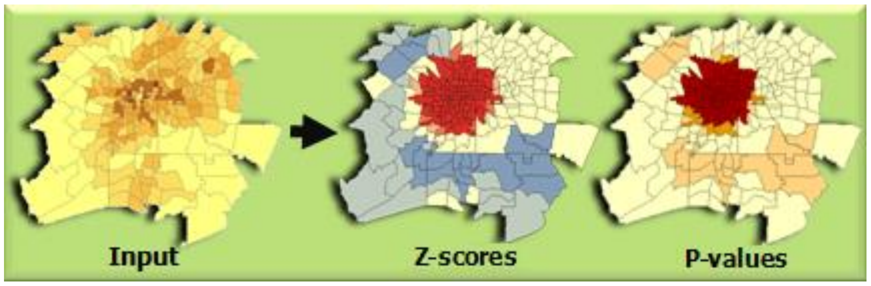


Figure 1 [37] - Sample hotspot map

The Getis-Ord local statistic is given as [16, 22, 23]:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S \sqrt{\frac{\left[n \sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j} \right)^2 \right]}{n-1}}}$$

where x_j is the attribute value of feature j, w_(i,j) is the spatial weight between feature i and j, n is equal to the total number of features and:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n}$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2}$$

4. Related work

In the recent years, a growing number of people use hotspot as a visualization method to analyze spatial autocorrelation, a big part of them concentrate on taking advantage of it to demonstrate the current implementation in different fields, such as diseases, crime, public service, and meteorology. Another part of them pays more attention to the parallel algorithm design and algorithm optimization.

4.1 Implementation:

For example, the team of Do Thi Thanh Toan[14], they used the daily data on dengue fever between January 1998 and December 2009 in old Hanoi, Vietnam did the Spatial-temporal analysis and the hotspot detection. From the hotspot map, they found that six out of the fourteen districts of Hanoi had significant cluster patterns. Some of these patterns indicate that closer villages tend to have more similar baseline incidence rates than those further apart and the majority of dengue cases occurred when the season with more rainfall and higher temperature. Their research helped the local medical department to intensify their remedial measures in the identified areas of high dengue fever prevalence and improved future strategies for more effective control.

Another example is that Long Tien Truong and Sekhar V.C. Somenahalli[2] done a research with thirteen years of pedestrian-vehicle crash data in Adelaide, the capital city of South Australia by using the method of Getis-Ord Gi* statistics rather than using Kernel Density Estimation (KDE) [15] which is the most widely used method for studying spatial patterns of road crash data. This is because the main weakness of the KDE method is that it cannot be tested for statistical significance [17,18]. In the hotspots analysis, they got the result that the pedestrian-vehicle crashes happened at intersections were not the most severe but that happened at mid-block locations due to high turning movements and the lack of appropriate pedestrian facilities.

What this research contributed just like they concluded that 'it offers a sound basis for identifying pedestrian-vehicle crash hot spots and prioritizing unsafe bus stops in order to study the causal factors, determine effective countermeasures, and plan a safer bus transit system'.

What is more, Jonas Lukasczyk's team [31] used the KDE approach to analyze a crime dataset from the City of Chicago's Data Portal website which including incidents of thefts and robberies in Chicago from January 1, 2001, to June 22, 2015. In this research, they have found that proper choosing the spatial bandwidth really matters to the final outcome, this is because a lower bandwidth would show more details. Like their statements in the research, if they chose the range equals to seven kilometers, the outcome was only showing one hotspot. But when they switched the range to three kilometers, the outcomes showed two smaller hotspots. Their research must be very helpful to the local police station since they could know on which area they should pay more attention in order to reduce property damage to local residents, to reduce the crime rate, and also to make people feel safe.

4.2 Algorithm design and optimization:

Liu Yan's team [29] designed a MapReduce approach to compute the spatial statistics of Gi * (d) in parallel. The algorithm they designed was based on data parallelism since the MapReduce is a data-centric computing model which means the entire data is distributed evenly on each node on the distributed file system, and that means the computations could be done on each node with partial data. So, theoretically, in this case, the computations on every single node could be done at the same time if the instruction applied on all parts of data stored on different nodes is ideal and no dependency among them either.

For their approach, in the case of Gi*(d), they decomposed the whole dataset into a number of (depends on the amount of the entire dataset and the number of pieces of data that each cell contains) cells where each cell consist of several pieces of data (points with related information in the spatial domain) and stored them on different nodes via HBase based upon Hadoop.

Then they sent the Map tasks to every storage nodes and used the Map function to deal with the input as InputSplit [30] and to compute the partial results, such as the mean and the standard deviation of a specific interested value of all the cells, as well as the spatial weight matrix of each pair of those cells if the distance between two cells within a specific distance. Next, they had used the Reduce function to get the output which is a set of <key, value> pairs. The job that Reduce functions did was to aggregate all the partial results from all the nodes and computed the global outcome of Gi*(d).

Their MapReduce framework significantly improved computational performance by getting rid of unnecessary computations. This is because they only considered the neighbor cells within a set distance to the query cell. All the other cells outside this distance would be ignored on their spatial contribution. Figure 2 is an example they made in their paper for the purpose of letting people better understand the way they determine the contributed neighbors. As we can see in the picture, they partitioned the original spatial domain into twenty-five cells by using quadtree [32]. In their approach, they let the cell with index 6 be the query cell, and ‘d’ is the distance to determine the contributed neighbors. We can clearly see that the cell 4, 5, 9, 14, and 15 are within the range of distance d to cell 6, consequently, these cells are supposed to be considered in the computation of the spatial weight matrix of cell 6, all the other cells outside this range should not be considered. Differently put, there would be only five cells participated in the computation rather than twenty-five cells.

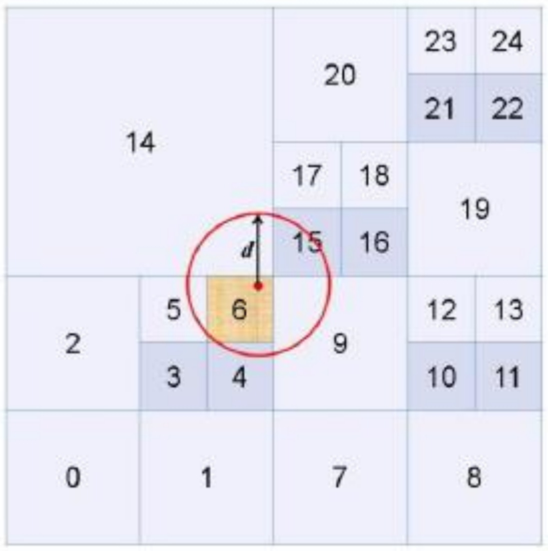


Figure 2 [29] – Finding neighbors for a given point using a Quadtree

Furthermore, there were some other teams worked on the similar project on Apache Spark [33, 34, 35, 36] with designing algorithms to do the spatial-temporal hotspot computation.

As we discussed above in the part of 'Ways to do parallel programming', Spark is a kind of a reinforcement version of MapReduce with richer operations and the RDDs. Therefore, using Apache Spark to do the calculations was in the same direction with the thought of Liu Yan's team which was using MapReduce approach. All the teams who are using Spark approach divided the entire dataset into cells where each cell contains several pieces of original data and stored at the different nodes on HDFS. Then used several MapReduce functions in the form of RDDs to do the necessary computations for getting the final Gi*(d) values.

For example, in [33], the first step of their algorithm (BigCAB) was to project each data point to the corresponding cell only with the interested value, this step allowed them to get the attribute value of each cell, that is the sum of all the data point's value.

Next, the second step was to aggregate the attribute values of the query cell and the neighboring cells. The final step was to do the computations following the Getis-Ord formula to get the result. But they did not straightforwardly follow the Getis-Ord formula, they made some optimizations on that formula. They assumed the weight of each neighbor cell to the query cell to be equal to one, and to calculate the weight matrix they only need to consider those neighboring cells, so they simplify the original formula to the following one:

$$G_i^* = \frac{\sum_{j \in \mathcal{N}(c_i)} x_j - \overline{X} \cdot |\mathcal{N}(c_i)|}{S \cdot \sqrt{\frac{n \cdot |\mathcal{N}(c_i)| - |\mathcal{N}(c_i)|^2}{n-1}}}$$

Where N(ci) represents the set of neighboring cells of a query cell, and |N(ci)| represents the number of neighboring cells that set contains. Through the execution time evaluation of the BigCAB algorithm with changing the sizes of the input dataset, they concluded that this algorithm’s scalability is good.

Take [35] as another example, they had similar steps to read the data, to partition the data into cells, and to do the calculation, but had a different simplification that selected a fixed number K of cells with the largest attributed value, these K cells were the candidates as the program’s output. I have no doubt that this is a tricky way to reduce the number of computations. However, it might lose some real hotspot since the hotspot computation not only depends on the query cell itself but also depends on the neighboring cells.

5. Dataset

The dataset I used for this research is '2013 Yellow Taxi Trip Data' from 'NYC OpenData', it contains 17 columns (variables) times 14,508,115 rows of data, the variables including pick-up and drop-off dates and times, pick-up and drop-off locations, trip distances, travel time, itemized fares, rate types, payment types, and driver-reported passenger counts. This dataset was provided by Taxi and Limousine Commission (TLC)

https://star.cs.ucr.edu/#NYCTaxi&center=39.68632,-88.05145999999999&zoom=5

In this paper, the variables I mainly use are pick-up and drop-off locations which are containing the latitude and the longitude, and the trip distances which contain the miles they traveled to analyze which area leads the taxi drivers high probabilities to earn more money, apparently, since the more miles they traveled, the more money passengers need to pay

METHOD

In the spatial autocorrelation (hotspot or cold spot) computation, using the distance-based weights can greatly improve performance [24]. So, I use the distance between every pairs of points as the basis for calculating the spatial weight matrix in this paper. Thus, the reasonable determination of the distance between two points is the fundamental basis of all work. If the distance is calculated incorrectly, all subsequent conclusions will be biased or even unreliable.

There are some existing popular methods to calculate the distance, for example the Manhattan distance method [40], but I did not use this method in this paper, maybe in the future I will try it. I calculate the spherical distance between each pair of starting points from their latitude and longitude using the following formula [6].

$$d\,(x1, y1, x2, y2) = r*\arccos(\sin(x1)*\sin(x2) + \cos(x1)*\cos(x2)*\cos(y1-y2))$$

where r is the radius of the earth and x1, x2, y1, and y2 are radians units.

After getting relatively (the Earth is not an exact sphere) accurate distance information, to determine the weight matrix about distance is crucial. The weight matrix is a symmetric matrix showing the influence of the distance between every two points on the spatial relationship. Let's assume we have n points, then the dimension of the weight matrix is n times n, and all the entries on the main diagonal are zeros because the distance between each point and itself is zero. But I didn’t use this whole n by n matrix to finish the computation, I just extract the corresponding column to do that. This is because all the distance information between a specific point and all the other points is in the column of that specific point.

For the dataset I used in this paper, I choose the fixed distance method to set the range, which means that each starting point of each car is the center of a virtual circle, and a fixed distance r is set as the radius of this circle. All the points inside this circle would have spatial contributions to the center point so their distances between the center point would be recorded as what they are in the weight matrix. For the points outside the circle, they would have zero spatial contribution to the center point, consequently, their distances to the center point would be zeros in the weight matrix. For example, for point A in Figure 3, let it be the center point rounded by a circle with radius r. The red dots are inside the circle, so only these six red dots have an effect on A in the distance, and the other blue dots are outside the circle, we consider they have no effect on A in the distance. Therefore, in the weight matrix of A, only the entries corresponding to these six red points have values, all the other entries related to those blue points are zero. And then for the six entries with values, we take the inverse of their distance to A, which means that the closer the distance, the greater the impact, and vice versa.

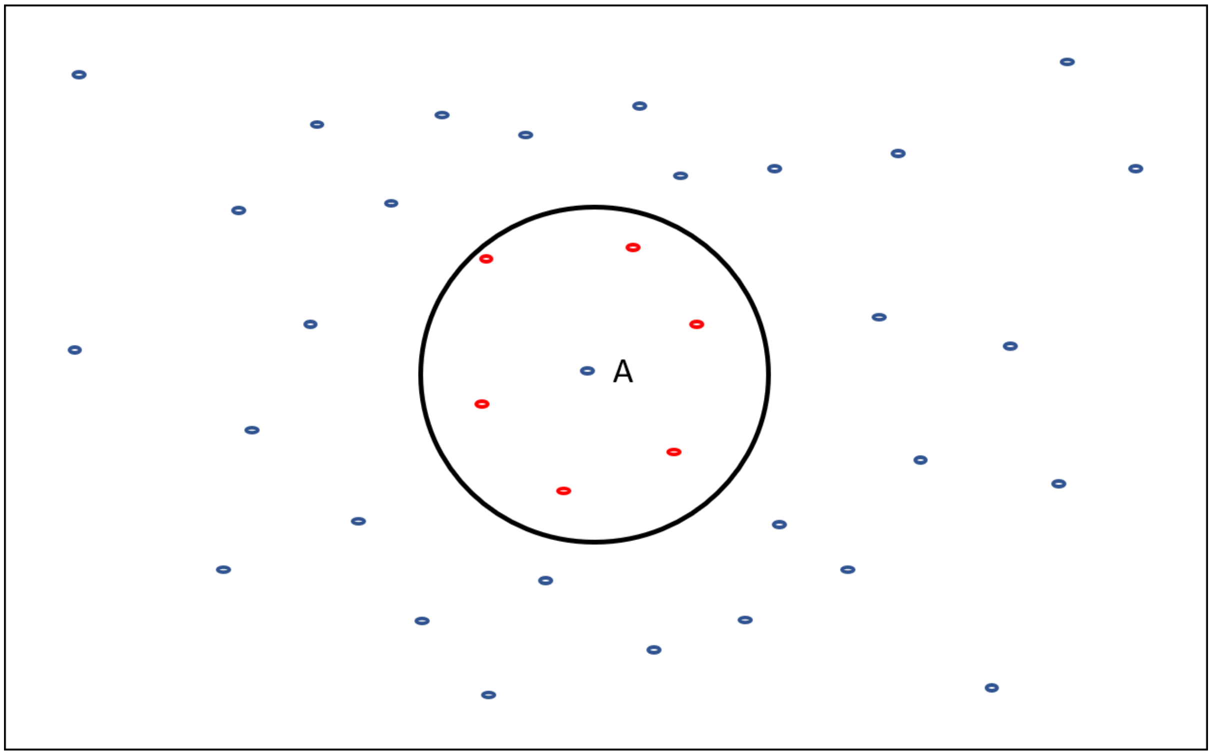


Figure 3 – Determination of contributed points for a query point

Applying the same process to each point, we will get weight matrices for every single point. And this part of computation could be parallelized, because to get the weight matrix of A does not depend on the result of other points' weight matrix. Then follow the Gi*(d) formula above, we can get the Gi* values of each point, the Pseudo Code is showing below, and the original code is in the appendices part 1.

```
Load the data;

Compute the mean and standard deviation of the interested variable;

Activate the 'parpool' and set the number of workers;

parfor i = 1:size of data

    for j = 1:size of data
```

```
        Compute the spherical distance between every point
        if the distance between point j and point i is in the range of i
            weight(j) = 1/distance;
            extract the value of point j
        end
    end
    Then follow the formula to calculate the Gi* values
end
```

In the line ‘parpool (‘local’, number of cores)’, the ‘number of cores’ is the place where we can set how many workers we want that to independently do the job at a time, if we set the ‘number of cores’ equals to twenty-four, then there would be twenty-four threads doing the calculation at the same time in parallel.

However, because the amount of data I used in this paper is large, there are more than fourteen million pieces of data, which means that there would be more than fourteen million points, so the dimension of the weight matrix for each point would be huge, and there are more than fourteen million points have the same huge weight matrix, the total number of times to calculate the weight matrix for each point is about 14508115 * 14508115, for such huge amount of calculation, it will bring a big pressure on memory, and ordinary computers will not have enough space to store them all.

In order to optimize the algorithm so that the number of calculations can be reduced, take less memory space, and be more effective, I need to figure out how to reduce the unnecessary calculations, just like the what those people did who use MapReduce and Spark approaches. I actually do not need to compute the distances between all the points and the query point, it would be good enough to compute only the distances between the points within the contributed range of the query point and the query point. So, I followed the basic idea of an r-tree [11] and then changed the way to calculate weight matrices.

An r-tree is a hierarchical data structure proposed by Guttman as a direct extension of B+-tree [9, 19, 20] in n-dimensions and it was designed for efficient execution of intersection queries [8]. Its structure is very similar to B+-tree [7]. Typical applications include computer-aided design, geographic information systems, computer vision and robotics, multi-keyed indexing for traditional databases, and temporal and scientific databases [8].

An r-tree does not store the original data, but the minimum bounding rectangles (MBR) and each of them contains several data points or the other lower-level MBRs. The MBRs with the spatial object are included in the leaf node of an r-tree. In the r-tree spatial index, some virtual rectangular targets are designed, and those similar or close spatial positions are included in the same rectangle. Virtual rectangles can be further subdivided, that is, they can be nested to form a multi-level spatial index [21].

For example, in Figure 4 [39], we have some data points randomly located in an area, to partition them by using r-tree, we can create nine MBRs which each of them contains a close cluster of points, name them from R1 to R9, just like Figure 5 [39] shows. Then, we can make more higher-level MBRs that containing such close lower-level MBRs just like Figure 6 [39] shows with R10, R11, and R12. In this way, if we want to search for a specific data point, we have a very clear architecture to do that. Let's assume these data points represent the restaurants in a local community and let's assume the right upper one is the one we want to search. Firstly, we can check in which higher-level MBR it is, R11 it is. Next, we only need to check which lower-level MBRs (within R11) it belongs to, clearly R4 it is. Then, if we want to calculate the spatial weight matrix of this point, we can do the computations almost only using the points in R4 rather than all the datasets.

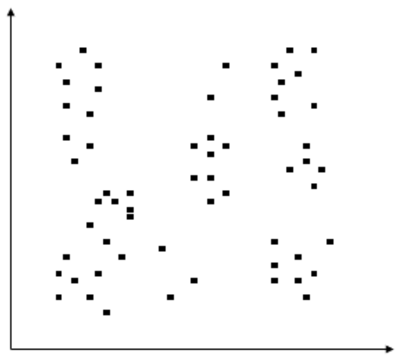


Figure 4 [44] – Raw data

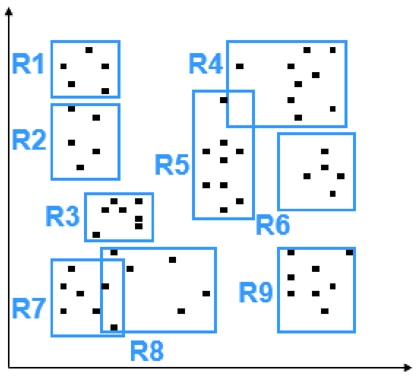


Figure 5 [44] – MBRs creation

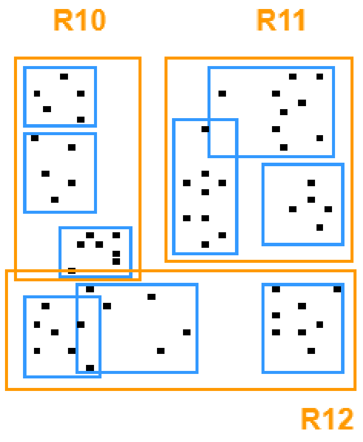


Figure 6 [44] – Higher level MBRs creation

An r-tree can not only used in 2-D but also useful in higher dimension like 3-D showed in Figure 7 [38]. It has the same technique to do the partition and searching.

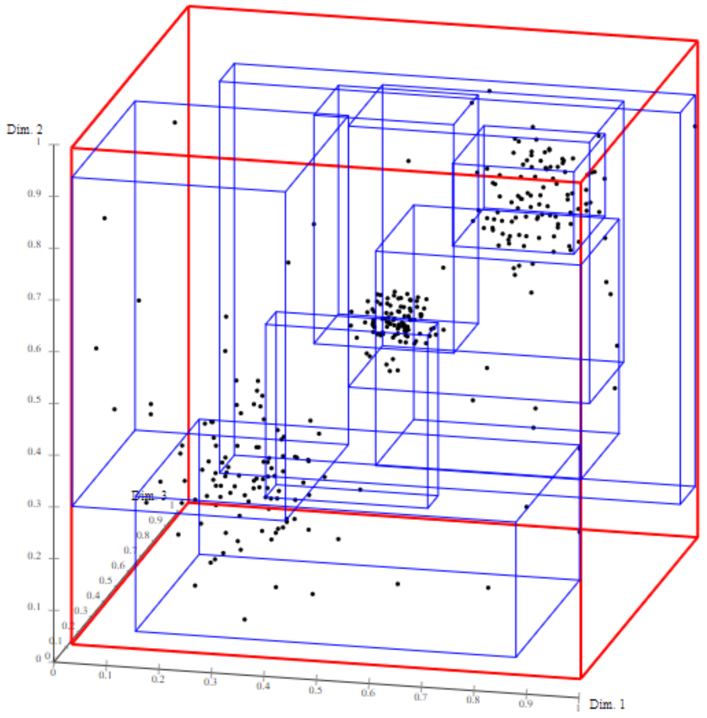


Figure 7 [38] – R-tree in 3D

Compared to the quadtree that Liu Yan's team used, an r-tree, as we discussed above, can be used for indexing multi-dimensional information (up to four dimensions), more than that, the index creation and tuning are easier, and required less storage. But a quadtree, simply put, is also a data structure in which each internal node has exactly four children. It usually used to partition a two-dimensional space by recursively subdividing it into four quadrants. It is more complex and required more storage [42].

In this paper, I just follow basic ideas of the r-tree and use a similar approach to do the calculations: reducing the amount of calculation (only use those data points inside the range of the query point to calculate the spatial weight matrix) and changing the previously mentioned range to a square rather than a circle. For example, if we take point A as the center point and want to determine which points have influence on point A, we need to determine which points are in the square that affects point A. First, we take the abscissa (latitude) of point A as the standard and extend twenty meters in the directions of up and down. If the difference between the abscissa of all other points and the abscissa of point A is less than twenty meters, proceed to the next stage. Then use the ordinate (longitude) of point A as the standard and extend twenty meters each in the left and right directions, thus created an invisible square. After the above two steps, we consider that all the points in the square have an effect on point A. The inverse of the distance between them and point A is the value of the corresponding point in the weight matrix for point A.

Since all points outside the square are considered to have zero effect on point A, even in the original method, their values in the weight matrix will be zero. Therefore, I only need to calculate the weight matrix formed by these points in the square, and I can get the same result. This not only reduces the huge memory consumption, but also saves a lot of calculation time.

Then use the toolbox ‘parpool’ and ‘parfor’ command in MATLAB on Everest to implement parallel computing. With that toolbox, I only need to set the number of working cores. Each core executes the same instruction on different points. That is the data parallelism. For example, if I set twenty-four cores, then there will be twenty-four cores working at the same time, in another words, it will calculate the weight matrix of twenty-four points at a time. Then in the same loop, after getting the weight matrix of each point, bring it into the previous formula and calculate the Gi * value of that point, which is the result we ultimately want. Pseudo Code is showing below, and the real code is in the appendices part 2.

```
Load the data;
Compute the mean and standard deviation of the interested variable;
Activate the ‘parpool’ and set the number of workers;
parfor i = 1:size of data
    Create a new index k for storing the corresponding values which is in the range of point(i)
    k = 1;
    for j = 1:size of data
        if point(j) in the range of point(i)
            d(i,j) = SphereDist;
            weight (k) = 1/d;
            extract the value of point j
            k = k+1;
        end
    end
    Then follow the formula to calculate the Gi* values
```

end

RESULT AND ANALYSIS

In the end, all the Gi * values from every point in the dataset I got showed a shape of distribution close to the Gamma distribution. Among these results, the minimum value is -122.6698, the maximum value is 1099.7, the average value is -11.9261, and the standard deviation is 82.5568.

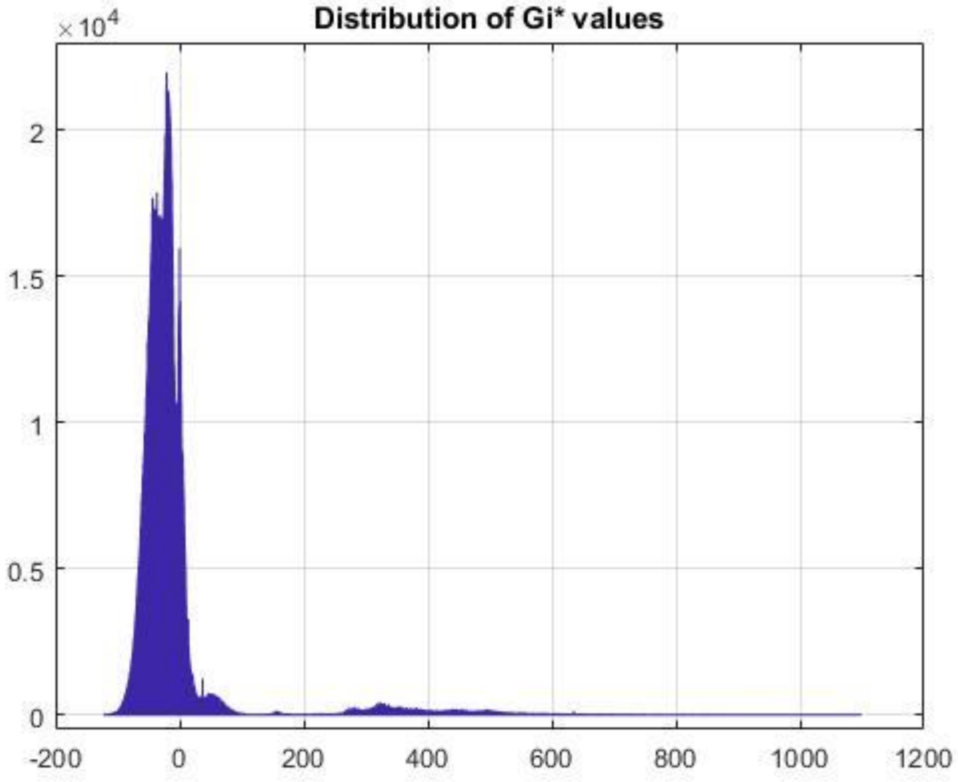


Figure 8 – Distribution of the result

When I use different numbers of cores to do the calculation, the results are exactly the same, but, as expected, the time they took was different. I used 12 cores, 16 cores, 20 cores, and 24 cores to do the calculation, Figure 9 below is a comparison of the time taken with using different cores:

Number of cores	**12**	**16**	**20**	**24**
Time using (seconds)	977,530	841,780	549,220	126,500

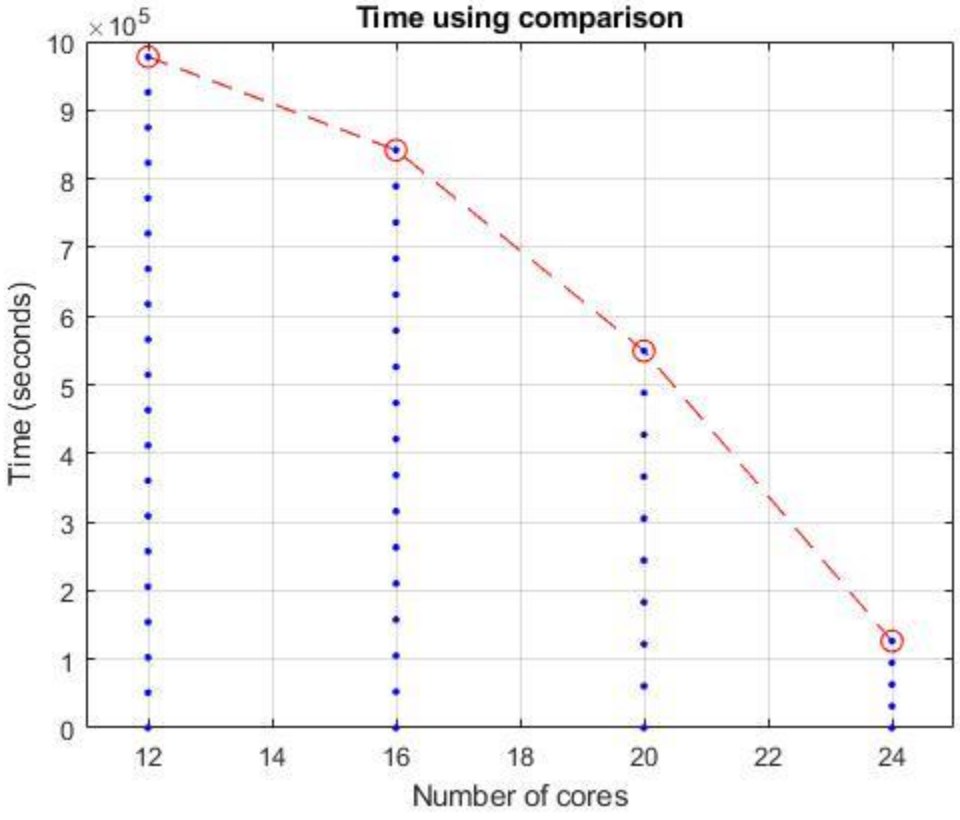


Figure 9 – Time comparison

From the Figure 9, we can clearly see that as the number of cores increases, the duration of time of the calculation is gradually decreasing, the 24-core calculation takes about ten times less time than the 12-core calculation. I can believe that when the number of cores used is less than 12, it will take more time, and when it is greater than 24, the time duration will be less until converges to an upper bound.

In the interest of saving time and CPU resources, I did not use all the data to test the performance of the calculations with the other number of cores, but I selected a part of data (100,000) for more detailed testing and analysis. At the same time, since I followed the basic idea of the r-tree and made some adjustments to the first parallel algorithm, I also compared the performance of these two algorithms before and after.

Figure 10 is a comparison of the time of using the two algorithms to calculate the same 100,000 data with different numbers of cores. We can clearly see that with the increase in the number of cores, the time consumption of both algorithms is decreasing and there is a tendency to converge. In addition, the algorithm using r-tree concept always takes about half time than the regular parallel algorithm does.

Time Comparison						
	1 core	6 cores	12 cores	16 cores	20 cores	24 cores
No r-tree	21829s	4478.3s	2450.9s	1928.08s	1565.14s	1306.92s
r-tree	10376s	2119.04s	1175.68s	920.39s	738.2s	617.81s

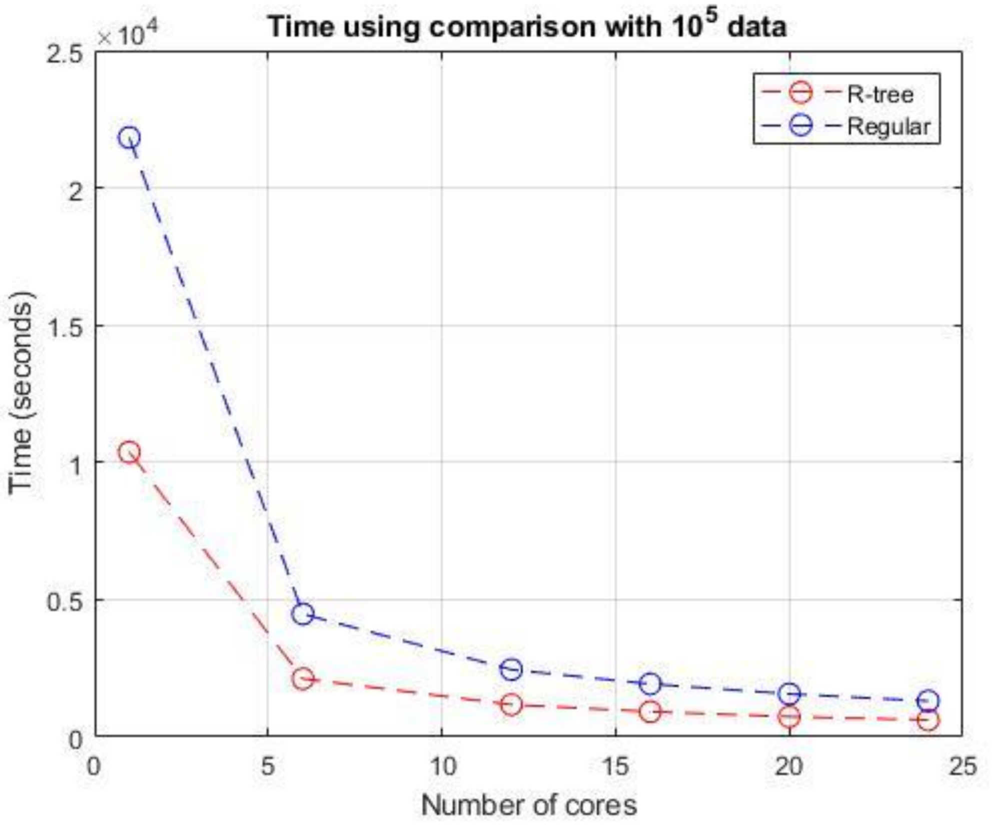


Figure 10 – Time comparison of two algorithms

Figure 11 is showing another comparison between two algorithms of the speedup as the number of cores increases. We can see the speedup of the two algorithms is basically equal.

Speedup Comparison						
	1 core	6 cores	12 cores	16 cores	20 cores	24 cores
No r-tree	1	4.87	8.9	11.32	13.95	16.7
r-tree	1	4.9	8.83	11.27	14.06	16.79

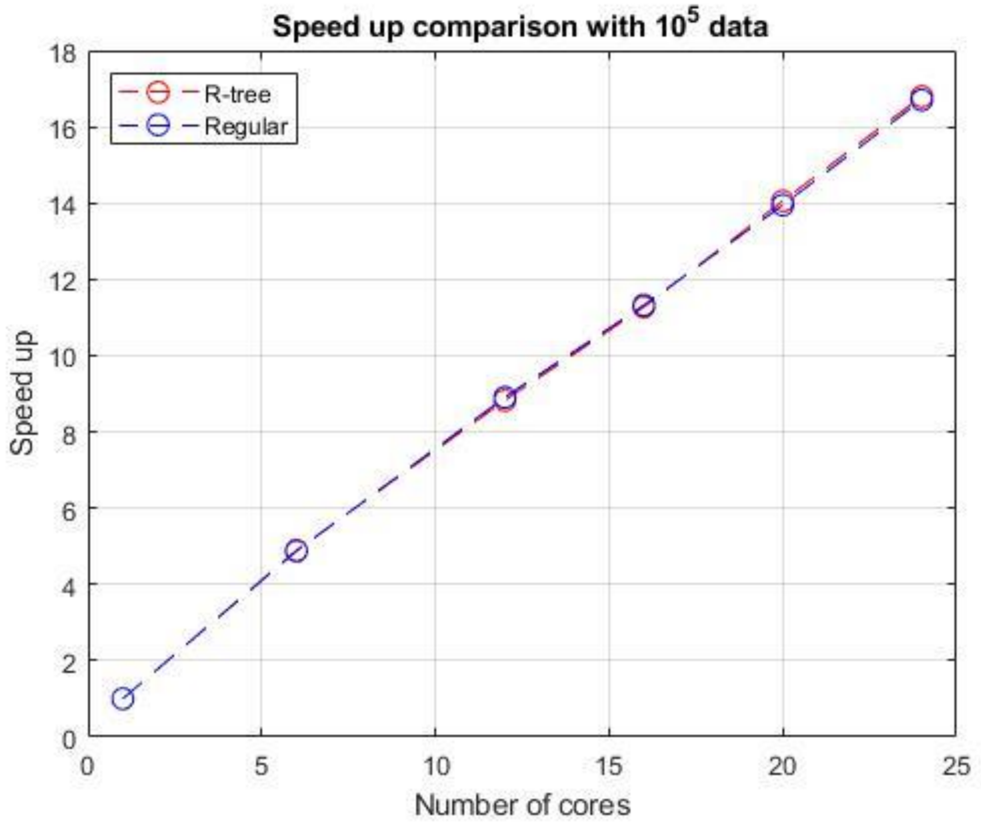


Figure 11 – Speedup comparison

Figure 12 shows the comparison of the efficiency between two algorithms. As the number of cores increases, their efficiency decreases, and the rate of decrease of the efficiency for these two algorithms does not show a big difference.

Efficiency Comparison						
	1 core	6 cores	12 cores	16 cores	20 cores	24 cores
No r-tree	1	0.81	0.74	0.71	0.69	0.68
r-tree	1	0.81	0.74	0.705	0.702	0.69

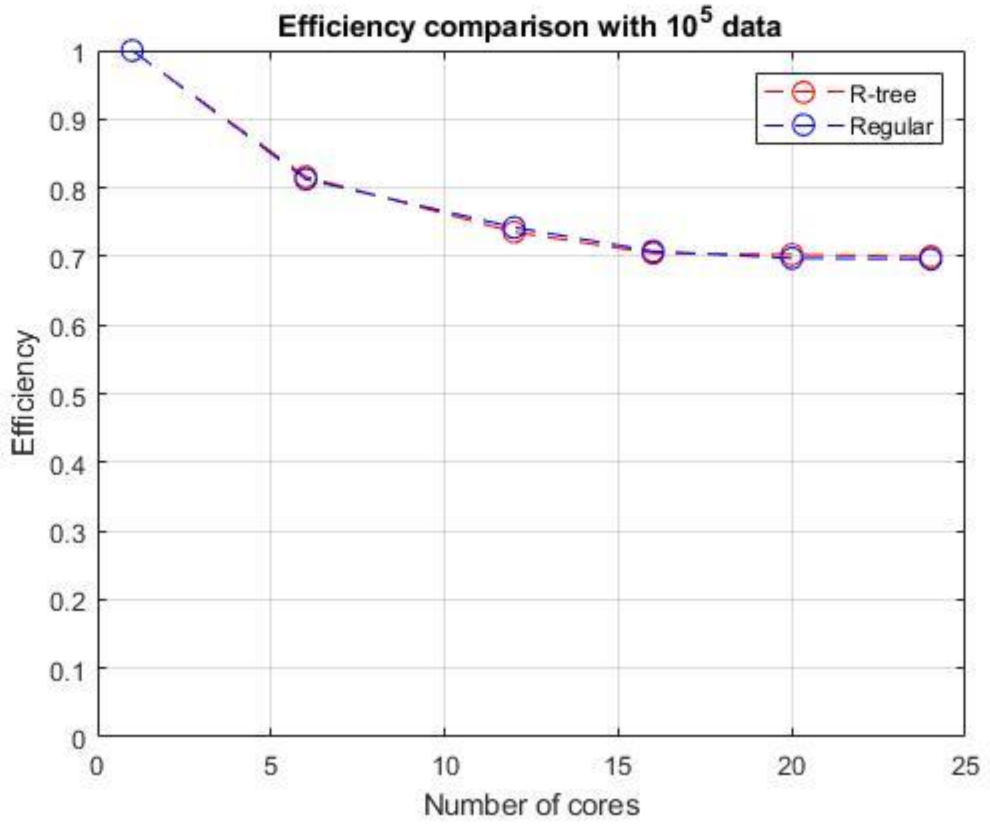


Figure 12 – Efficiency comparison

Based on the results (Gi* values) I got, we can analyze in which areas New York taxi drivers having passengers can make them more money, because generally speaking, the farther the destination, the greater the fare. Therefore, we can have the corresponding conclusions by analyzing which location has a significantly high Gi* value and also be surrounded by high Gi* value locations, which means the hotspot.

Furthermore, we can also get other valuable information, such as which area has a greater demand for taxis in a specific time period. To get this information, we can first filter all the data for a specific period from the original data, like 6-9 pm. and use the area as the unit instead of the point as the example before, to count the number of passengers in each area during the period. (If there are m people taking n taxis, maybe not only one passenger per car, then the number of passengers is calculated as m because m is the number of people in need of the taxi), and then we can set the center point of each area as the key point which means to use this key point's longitude and latitude to be the area's, and set a proper distance as the radius, and then calculate the Gi * value of each area through the formula. After that, we can get the result of which area is in the specific time period the taxi driver is more likely to have business.

What we discussed above are analyses that can help drivers improve their earnings. If combined with other dataset, we can even provide information to increase safety factors. For example, if we combined with the local crime rate data, we could know which areas have more crimes and what kinds of the crimes have happened with high probabilities. Taking these areas as endpoints and filtering out the corresponding data from our original data, we can know that the passengers from which starting points are more willing to go to those relatively dangerous areas. Consequently, taxi drivers will have the data to help them made a good choice.

In the method of this paper, a maximum of 24 cores are used to perform parallel computing, and the result needs to be about one and a half days. But I believe that if I use more cores to calculate, I will spend less time so that I can get valuable information much faster.

To create a hotspot map, please see appendices part 3.

CONCLUSION AND FUTURE WORK

In the real world, some tasks cannot be split and then executed in parallel. But from what we have discussed before, we can draw a conclusion that the computation of spatial autocorrelation (hot spot) can be executed in parallel with MATLAB, and it has an obvious advantage in time over the serial algorithm. Not only that, when I use the basic idea of r-tree to optimize the algorithm, it saves much more time, so this proves that r-tree is a very useful tool for spatial retrieval, and it also contributes a lot on the applications of calculating spatial autocorrelation. Of course, however, people are always pursuing better methods, so in the future, I will continue to study in-depth on how to further optimize the algorithm. At the same time, MATLAB is not the mainstream programming language for parallel computing after all. I will try to test the efficiency of the algorithm with other programming languages in the future and try to find a relatively better one.

After all, the algorithms in this paper make the calculation process less time consuming especially the optimized algorithm, and can obtain accurate data in related fields faster, which helpful to analyze the current situation more intuitively and quickly, also to help people make more accurate predictions for the future so that they can have practical plans before the problem really happens.

REFERENCES

[1] Changcan Wu, Spatial Correlation Analysis of GDP in Tianjin Based on Spatial Statistiscs, 2017.

[2] Truong, Long T & Somenahalli, Sekhar V. 2011. Using GIS to Identify Pedestrian-Vehicle Crash Hot Spots and Unsafe Bus Stops. Journal of Public Transportation, 14 (1): 99-114. DOI: http://doi.org/10.5038/2375-0901.14.1.6 Available at: https://scholarcommons.usf.edu/jpt/vol14/iss1/6

[3] Ord, J. Keith, and Arthur Getis. "Local spatial autocorrelation statistics: distributional issues and an application." Geographical analysis 27.4 (1995): 286-306.

[4] Waters, Nigel. (2017). Tobler's First Law of Geography. 10.1002/9781118786352.wbieg1011.

[5] https://pro.arcgis.com/en/pro-app/tool-reference/spatial-statistics/h-how-hot-spot-analysis-getis-ord-gi-spatial-stati.htm

[6] 刘凡俊, 李登有. 球面的距离公式及其应用[J]. 数学教学研究, 2013, 32(3):39-40,43.

[7] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD international conference on Management of data (SIGMOD ’90). Association for Computing Machinery, New York, NY, USA, 322–331. DOI:https://doi.org/10.1145/93597.98741

[8] S. T. Leutenegger, M. A. Lopez and J. Edgington, "STR: a simple and efficient algorithm for R-tree packing," Proceedings 13th International Conference on Data Engineering, Birmingham, UK, 1997, pp. 497-506.

[9] Theodoridis, Y., & Sellis, T. (1996, June). A model for the prediction of R-tree performance. In PODS (Vol. 96, pp. 161-171).

[11] Guttman, A., R-trees: a Dynamic Index Sturcture of Spatial Searching, Proc. ACM SIGMOD, p. 47-57, 1984

[12] OpenMP, I. C. V. "2 OpenMP." (1997)

[13] Duran, Alejandro, Julita Corbalán, and Eduard Ayguadé. "Evaluation of OpenMP task scheduling strategies." International Workshop on OpenMP. Springer, Berlin, Heidelberg, 2008.

[14] Do Thi Thanh Toan, Wenbiao Hu, Pham Quang Thai, Luu Ngoc Hoat, Pamela Wright & Pim Martens(2013) Hot spot detection and spatio-temporal dispersion of dengue fever in Hanoi, Vietnam, Global Health Action, 6:1, 18632

[15] Duong, Tarn. "ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R." Journal of Statistical Software 21.7 (2007): 1-16.

[16] Getis, A., and J.K. Ord. 1992. The analysis of spatial association by use of distance statistics. Geographical Analysis 24(3):189-207

[17] Anderson, T.K.2009. Kernel density estimation and K-means clustering to profile road accident hot spots. Accident Analysis & Prevention 41(3):359-364.

[18] Xie, Z., and J. Yan. 2008. Kernel Density Estimation of traffic accidents in a network space. Computers, Environment and Urban Systems 32(5): 396-406.

[19] D.Knuth, The Art of Computer Programming, vol. 3: Sorting and Searching. Addison-Wesley, Reading, MA, 1973.

[20] D. Comer, “The Ubiquitous B-Tree”, ACM Computing Surveys, vol. 11(2), pp. 121-137, June 1979.

[21] https://www.jianshu.com/p/b55cdce0b533

[22] Wu J, Wang J, Meng B, Chen G, Pang L, Song X, Zhang K, Zhang T, Zheng X: Exploratory spatial data analysis for the identification of risk factors to birth defects. BMC Public Health. 2004, 4: 23-10.1186/1471-2458-4-23.

[23] Feser E, Sweeney S, Renski H: A descriptive analysis of discrete U.S. industrial complexes. Journal of Regional Science. 2005, 45: 395-419. 10.1111/j.0022-4146.2005.00376. x.

[24] D.-Kl.D. Rokos, M.P. Armstrong,Using Linda to compute spatial autocorrelation in parallel,Computers & Geosciences,Volume 22, Issue 4,1996,Pages 425-432,ISSN 0098-3004

[25] Tsai, PJ., Lin, ML., Chu, CM. et al. BMC Public Health (2009) 9: 464. https://doi.org/10.1186/1471-2458-9-464

[26]. Peter S. Pacheco, Editor(s): Peter S. Pacheco, An Introduction to Parallel Programming, Morgan Kaufmann, 2011, ISBN 9780123742605, https://doi.org/10.1016/B978-0-12-374260-5.00001-4.

(http://www.sciencedirect.com/science/article/pii/B9780123742605000014)

[27] M. Herlihy, N. Shavit, The Art of Multiprocessor Programming, Morgan Kaufmann, Boston, 2008.

[28] Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. Using OpenMP: portable shared memory parallel programming. Vol. 10. MIT press, 2008.

[29] Liu, Yan, et al. "A MapReduce approach to G i*(d) spatial statistic." Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems. 2010.

[30] Apache. 2010. The Apache Hadoop Project. Retrieved Jun 12, 2010, from http://hadoop.apache.org/core/.

[31] Lukasczyk, Jonas, et al. "Understanding hotspots: a topological visual analytics approach." Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2015.

[32] Samet, Hanan. "The quadtree and related hierarchical data structures." *ACM Computing Surveys (CSUR)* 16.2 (1984): 187-260.

[33] Nikitopoulos, Panagiotis, et al. "BigCAB: Distributed Hot Spot Analysis over Big Spatio-temporal Data using Apache Spark (GIS Cup 2016)."

[34] Gomes, Salles Viana, et al. "An efficient map-reduce algorithm for spatio-temporal analysis using Spark (GIS Cup)."

[35] Peng, Shangfu, et al. "Simplification and Refinement for Speedy Spatio-temporal Hot Spot Detection Using Spark (GIS Cup)."

[36] Mehta, Paras, Christian Windolf, and Agnès Voisard. "Spatio-temporal hotspot computation on apache spark (gis cup)." *24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2016.

[37] https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-statistics-toolbox/hot-spot-analysis.htm

[38] https://upload.wikimedia.org/wikipedia/commons/thumb/5/57/RTree-Visualization-3D.svg/400px-RTree-Visualization-3D.svg.png

[39] Dr Eamonn Keogh, Computer Science & Engineering Department, University of California – Riverside, lecture slides

[40] Shahid, R., Bertazzon, S., Knudtson, M. L., & Ghali, W. A. (2009). Comparison of distance measures in spatial analytical modeling for health service planning. BMC health services research, 9, 200. https://doi.org/10.1186/1472-6963-9-200

[41] https://qgis.org/en/site/. QGIS. Retrieved 16 October 2013.

[42] http://informix-spatial-technology.blogspot.com/2012/01/comparison-on-b-tree-r-tree-quad-tree.html

[43] https://data.police.uk/data/

[44] https://slideplayer.com/slide/4931692/

APPENDICES

Part 1: Matlab code for parallelly computing Gi* statistic.

```
close all; clear all; clc

load  'startX.csv'

load  'startY.csv'

load  'miles.csv'

start_x = startX;

start_y = startY;

mile    = miles;

[length(start_x),length(start_y),length(mile)]

fileGi = fopen('Gi.csv','w');

tic

n = length(start_x)

x_bar = mean(mile)

S = sqrt((sum(mile.^2))/n - x_bar^2)

Gi_star = zeros(n,1);

parpool('local',24)

parfor i = 1:n
```

```
 tem_distance = [];

tem_mile = [];

 k = 1;

for j = 1:n

        d =  SphereDist2([start_x(i),start_y(i)],[start_x(j),start_y(j)]);

         if d ~=0 && d <= 0.012 % 0.012 mils = 20 meters

                tem_distance(k) = 1/d;

                tem_mile(k) = mile(j);

                 k = k+1;

        end

end

Gi_denominator_2 = sqrt((n*sum(tem_distance.^2)-(sum(tem_distance))^2)/(n-
1));

if Gi_denominator_2 == 0

         Gi_star(i) = 0;

 else

        Gi_numerator     = tem_distance * transpose(tem_mile) - x_bar *
sum(tem_distance);

        Gi_star(i) = Gi_numerator / (S * Gi_denominator_2);
```

```
          end
end
time = toc
fprintf(fileGi,"%f\n",Gi_star);
fclose(fileGi);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function part below:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function rad = D2R(theta) % Angle into radians
          rad = theta*pi/180;
end
function d = SphereDist2(x,y,R)
          if nargin < 3
                    R = 6378.137;
          end
          x = D2R(x);
          y = D2R(y);
          h = HaverSin(abs(x(2)-y(2)))+cos(x(2))*cos(y(2))*HaverSin(abs(x(1)-y(1)));
```

```
        d = 2 *R * asin(sqrt(h));

         function h = HaverSin(theta)

                h = sin(theta/2)^2;

        end

end
```

Part 2: Optimized Matlab code for parallelly computing Gi* statistics.

```
close all; clear all; clc

load  'startX.csv'

load  'startY.csv'

load  'miles.csv'

start_x = startX;

start_y = startY;

mile    = miles;

[length(start_x),length(start_y),length(mile)]

fileGi = fopen('Gi_star.csv','w');

tic

n = length(start_x)

x_bar = mean(mile)

S = sqrt((sum(mile.^2))/n - x_bar^2)

Gi_star = zeros(n,1);

parpool('local',24)

parfor i = 1:n

        tem_distance = [];
        tem_mile = [];

        k = 1;

        for j = 1:n

            if abs(start_x(j)-start_x(i))<= 0.003 && abs(start_y(j)-start_y(i))<= 0.003

            && start_x(j)~=start_x(i) && start_y(j)~=start_y(i)

                d =  SphereDist2([start_x(i),start_y(i)],[start_x(j),start_y(j)]);

                tem_distance(k) = 1/d;

                tem_mile(k) = mile(j);

                k = k+1;

             end

        end

        Gi_denominator_2 = sqrt((n*sum(tem_distance.^2)-(sum(tem_distance))^2)/(n-
    1));

        if Gi_denominator_2 == 0

            Gi_star(i) = 0;

        else

            Gi_numerator     = tem_distance * transpose(tem_mile) - x_bar *

            sum(tem_distance);

            Gi_star(i) = Gi_numerator / (S * Gi_denominator_2);
        end
end
time = toc
fprintf(fileGi,"%f\n",Gi_star);
fclose(fileGi);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function part below:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function rad = D2R(theta)
        rad = theta*pi/180;
end
function d = SphereDist2(x,y,R)
        if nargin < 3
                R = 6378.137;
        end
        x = D2R(x);
        y = D2R(y);
        h = HaverSin(abs(x(2)-y(2)))+cos(x(2))*cos(y(2))*HaverSin(abs(x(1)-y(1)));
        d = 2 *R * asin(sqrt(h));

        function h = HaverSin(theta)

                h = sin(theta/2)^2;

        end

end
```

Part 3: Description of the demo of HotSpots using data set 'Crime location data set in Surrey, UK' [43] using QGIS

To create a hotspot map, we can use QGIS [41] which is a free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data.

The first step is to load the default 'OpenStreetMap' and the dataset to be visualized like Figure 13 and Figure 14 are showing.

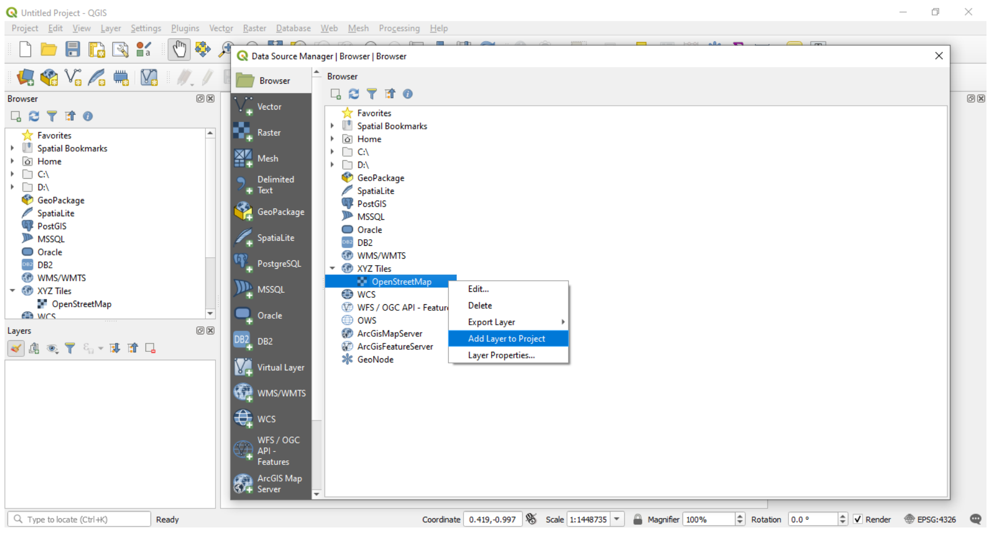
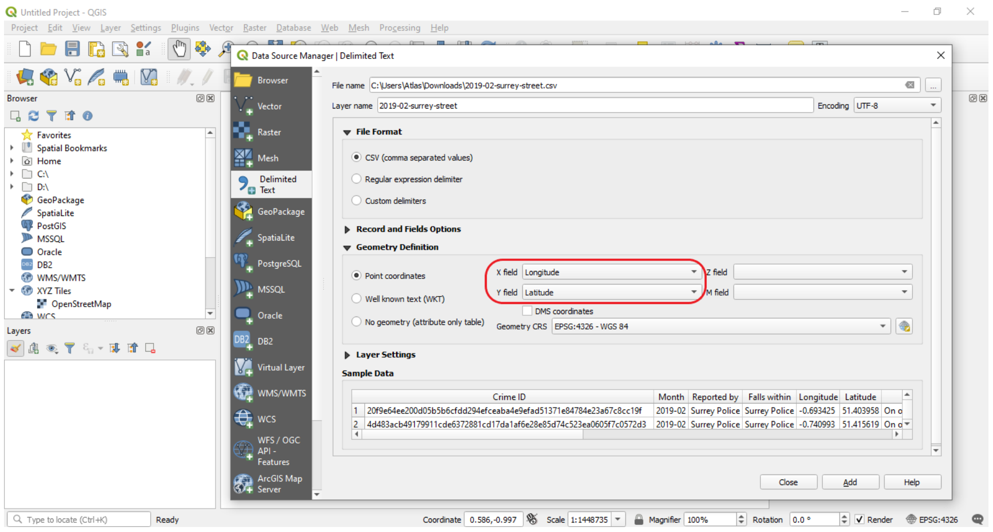


Figure 13 – Load the default map


Data Source Manager | Delimited Text

File name: C:\Users\Atlas\Downloads\2019-02-surrey-street.csv
Layer name: 2019-02-surrey-street   Encoding: UTF-8

File Format
- [✓ CSV (comma separated values)]
- Regular expression delimiter
- Custom delimiters

Record and Fields Options

Geometry Definition
- [✓ Point coordinates] X field: Longitude Y field: Latitude Z field: M field:
- Well known text (WKT) DMS coordinates
- No geometry (attribute only table) Geometry CRS: EPSG:4326 - WGS 84

Layer Settings

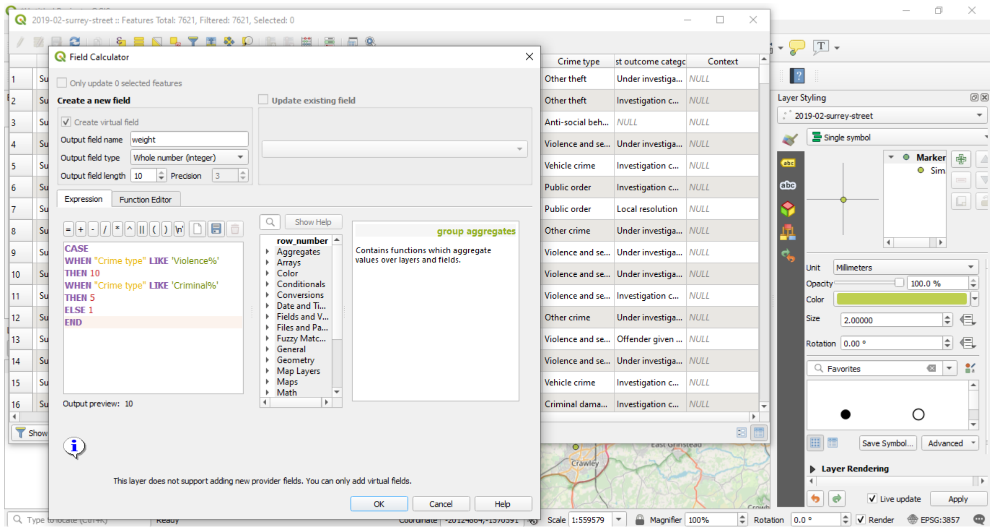
Sample Data

	Crime ID	Month	Reported by	Falls within	Longitude	Latitude	
1	20f9e64ee200d05b5b6cfdd294efceaba4e9efad51371e84784e23a67c8cc19f	2019-02	Surrey Police	Surrey Police	-0.693425	51.403958	On o
2	4d483acb49179911cde6372881cd17da1af6e28e85d74c523ea0605f7c0572d3	2019-02	Surrey Police	Surrey Police	-0.740993	51.415619	On o

Coordinate 0.586,-0.997 Scale 1:1448735 Magnifier 100% Rotation 0.0 ° Render EPSG:4326

Figure 14 – Load the data

Then, we can select a variable to be the weight and set how each of the data in this variable contributes. In this demo, the dataset is about criminal acts, so we can set the different weight levels base on the type of crime like Figure 15 shows. After this, the system will create a new column indicating the weight of each piece of data.


2019-02-surrey-street :: Features Total: 7621, Filtered: 7621, Selected: 0

Field Calculator

Only update 0 selected features

Create a new field
- [✓] Create virtual field
- Output field name: weight
- Output field type: Whole number (integer)
- Output field length: 10 Precision: 3

Update existing field

Expression

```
CASE
WHEN "Crime type" LIKE 'Violence%'
THEN 10
WHEN "Crime type" LIKE 'Criminal%'
THEN 5
ELSE 1
END
```

Output preview: 10

group aggregates

Contains functions which aggregate values over layers and fields.

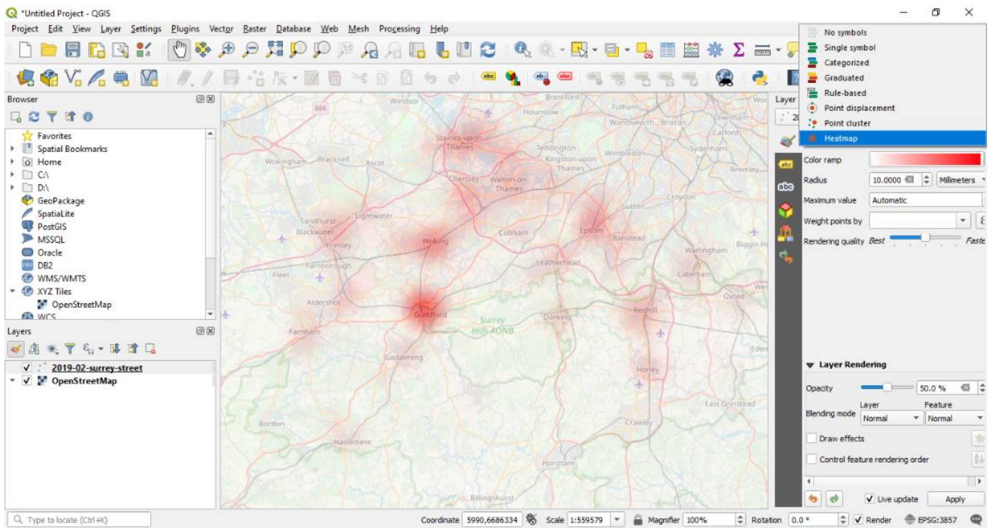
This layer does not support adding new provider fields. You can only add virtual fields.

	Crime type	st outcome categc	Context
1	Other theft	Under investiga...	NULL
2	Other theft	Investigation c...	NULL
3	Anti-social beh...	NULL	NULL
4	Violence and se...	Under investiga...	NULL
5	Vehicle crime	Investigation c...	NULL
6	Public order	Investigation c...	NULL
7	Public order	Local resolution	NULL
8	Other crime	Under investiga...	NULL
9	Violence and se...	Under investiga...	NULL
10	Violence and se...	Under investiga...	NULL
11	Violence and se...	Investigation c...	NULL
12	Other crime	Under investiga...	NULL
13	Violence and se...	Offender given ...	NULL
14	Violence and se...	Under investiga...	NULL
15	Vehicle crime	Investigation c...	NULL
16	Criminal dama...	Investigation c...	NULL

Layer Styling: 2019-02-surrey-street, Single symbol, Unit Millimeters, Opacity 100.0 %, Size 2.00000, Rotation 0.00 °

Figure 15 – Set the weight

Next, we can set the visualization way to be the ‘heatmap’ and change the color and the opacity to view the result like Figure 16.



Color ramp
Radius 10.0000 Millimeters
Maximum value Automatic
Weight points by
Rendering quality Best Faste

Layer Rendering
Opacity 50.0 %
Blending mode Layer Normal Feature Normal
Draw effects
Control feature rendering order
Live update Apply

Coordinate 5990,6686334 Scale 1:559579 Magnifier 100% Rotation 0.0 ° Render EPSG:3857

Figure 16 – Showing the heat map