An improved approach to explore spatial district aggregation

**Abstract**: Districts redistribution problem is an important research subject in census geography in terms of its potential ability to facilitate the organization and management of census while providing customized area for the user to use [1]. In many situations, it is often useful to group a large number of spatial objects such as census tracts into smaller ones whose results will form a more comprehensive spatial extend [2]. This procedure is called district aggregation. In order to deal with district aggregation problem, both performance and computational time should be well considered. This paper uses an improved approach to solve district aggregation problem by using GAT (Give and Take) algorithm [3] combined with Merge and Combine method and Parallel computing technology. This approach, not only solves the problem that GAT’s performance cannot be further improved when reaching to a certain number of iteration, but can also generate multiple optimized results at the same time. The approach and experiment are implemented and analysis are made in this paper.

**Key words:** District Aggregation, Evolutionary Algorithm, GAT, Parallel Computing, Census Geography

**Introduction:**

**Political redistribution problem, also called zone design problem, is very common in census geography. In general, it is to find a method to aggregate n units into k zones (k<n) such that some objective function values are optimized while some spatial constraints are maintained [4]. One of the most common applications in political redistribution is to aggregate administrator units into a predefined number of regions such that each new region is internally contiguous and the sum of population in each region is as similar as possible [4]. In many situations, by solving this problem, the user is able to get customized regions which will form a more comprehensive spatial extend.**

**Redistribution problem can be considered as geographic optimization problem. In general, geographic optimization problem can be categorized as four types [5]: First, selection problem without spatial constraints. What is does it to find the subset of spatial entities to satisfy one or more goals. One typical example of this problem includes p-median problem which PJ Densham, G Rushton had done some optimized method research on this area [6].**

**The second type is selection problem with spatial constraints. In addition to the selection procedures that are similar with first type, it also need to maintain the spatial constraints. One common example of this is site selection problem. In 2000,** [**TJ Cova**](https://scholar.google.com/citations?user=bnnYw7AAAAAJ&hl=en&oi=sra)**,**[**RL Church**](https://scholar.google.com/citations?user=t4-JG2IAAAAJ&hl=en&oi=sra) **raised multiple spatial constraints for a single region site search problem [7] while in 2002, H Önal, RA Briers incorporated spatial constraints and used integer programming method to solve this problem [8]. In 2005, Takeshi Shirabe purposed a new formulation of contiguity that can be used to describe spatial constraints and be incorporated into any complex integer programming model [9]. In 2009, he himself purposed a new integer programming approach to solve site selection problem [10].**

**The third type of problem is the partition problem without spatial constraints. Each spatial object is assigned a value and we need to find out the combination value of those spatial entities to satisfy some objective. Bennett, Xiao, and Armstrong has done deep research in this area by using Evolutionary algorithm [11].**

**The last type of this problem which we will also address specifically in this paper refers to partition problems with spatial constraints. In addition to the process of the third problem, spatial constraints must be satisfied. Political redistribution problem is exactly such a problem which space needs to be rearranged while spatial constraints must be maintained as well.**

**Solving this problem is challenge for the following reasons: first, it is hard to optimize the objective function value while at the same time maintains spatial constraints like contiguity. Second, these problems are often computational expensive because the number of feasible solutions will increase exponentially with the input size [5]. In this case, exact method may not be suitable for solving this problem although it can reach to global optimized result. Instead, a compromised method, called heuristic method is used to produce results in an acceptable amount of time while achieving near optimal solutions.**

**Many methods have been raised to solve this problem so far. Back in 1991, S Openshaw and L Rao tried to solve Redistribution problem by using AZP algorithm [12] and applied this method in the field of social economic units. This method though achieves good result, is computational expensive. Improvement to AZP lead to zone design system in 1995 and in 2002, Alvanides used simulated annealing to develop an alternative implementation of AZP** [13]. In 2004, [F Bacao](https://scholar.google.com/citations?user=sFoh1WAAAAAJ&hl=en&oi=sra), [V Lobo](https://scholar.google.com/citations?user=z3xr3NQAAAAJ&hl=en&oi=sra), [M Painho](https://scholar.google.com/citations?user=L1PHpI8AAAAJ&hl=en&oi=sra) used generic algorithm to zone design while in 2005 RM Assunção, MC Neves etc. used Minimum Spanning Tree (MST) to regionalize social economic units in a more efficient way. **In 2008, N, Xiao proposed a unified framework which solves geographic optimization problem by using graph theory and evolutionary algorithm. After that, in 2011, Myung Jin Kim utilized Xiao’s thinking and raises a new and efficient algorithm called Give and Take (GAT). This algorithm performs quite effectively and efficiently in terms of its ability to solve population equality problem.**

**However, this algorithm does has some limitations. First, its final result cannot be further improved when the total number of iteration reaches to a certain point. Second, this algorithm can only generate one good result at a time. If user wants to have more than one result, he/she should run this algorithm multiple times. This is not a big issue when dealing small dataset, but the factors of saving computational time becomes more critical when the input dataset becomes larger. This paper solves these two limitations by utilizing the thinking of evolutionary algorithm and the parallel computing technology. Methods will be elaborated and experiment and results will be shown in the following chapters.**

**The paper is organized in this way: Chapter one is introduction which has an overview of this topic’s research condition. Chapter two gives detailed explanation of theory and methods. Chapter three is the experiment, result and relevant analysis. Chapter four is the conclusion and the future work that needs to be done.**

**Methods:**

**This paper implements GAT (Give and Take Algorithm), one of the heuristic method to solve political redistribution problem while utilizing Evolutionary Algorithm and parallel computing technique to solve the two limitations I have discussed of GAT algorithm. This chapter will give a detailed description and explanation of the methods that are used. First, the paper will talk about graph theory and how it is used to represent spatial unities. And then it describe details of GAT. Thirdly it describes the recombining methods which utilizes the thinking of Evolutionary Algorithm to solve the first limitation that has been mentioned in the introduction. Finally, it introduces parallel computing technology to generate multiple good solutions at a time.**

**Graph Theory and its representation**

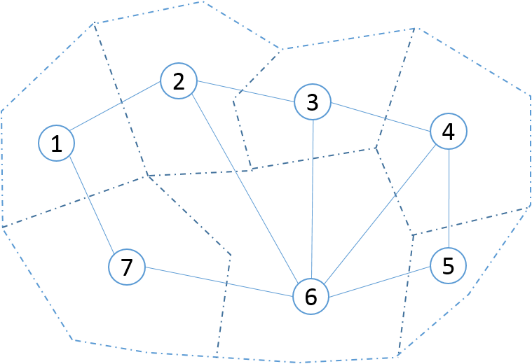
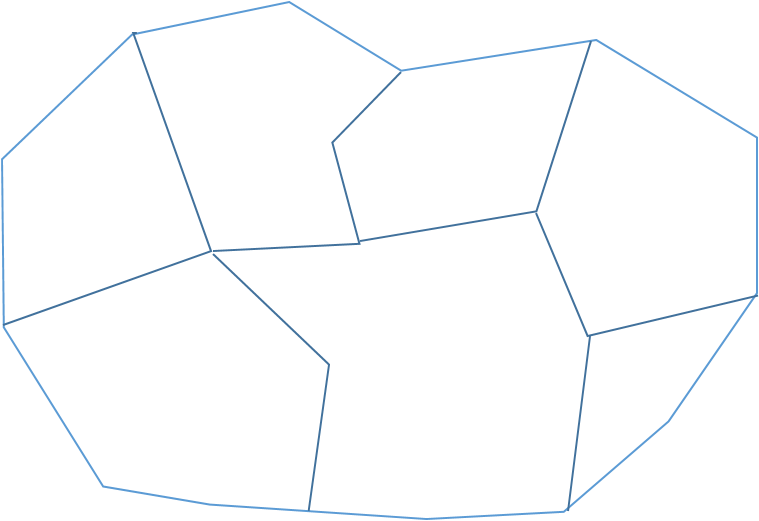
**A computer graph is defined as follows: where is called vertex or nodes and is a set of all edges that may connect one vertex to another. For example, can be represented an edge between vertex and vertex, thus.**

**Graph can be categorized as directed graph and undirected graph. Directed graph has direction attribute for each edge which uses an arrow to represent while undirected graph does not have such attribute. Figure 1a shows the example of this difference.**



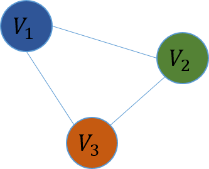
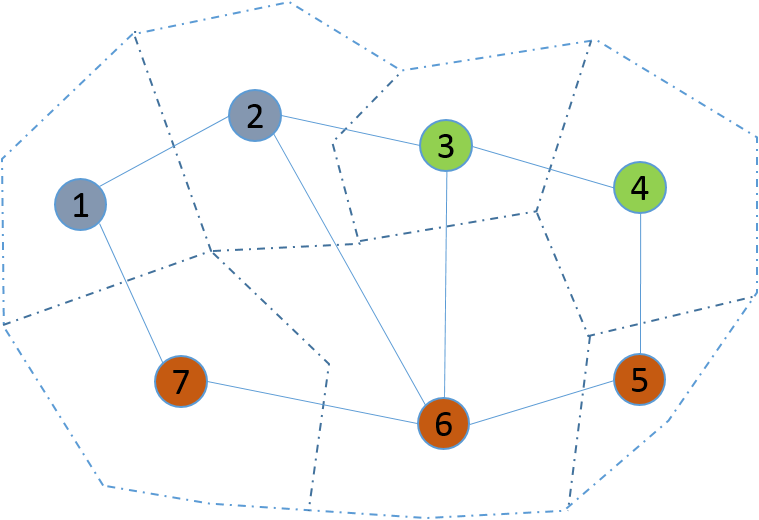
***Figure 1a: Left part is directed graph which edge has arrow to represent direction. Right part is undirected graph which edge does not have arrow.***

**In this paper, computer graph is used to represent spatial. Each census tract or block in a map is represented as a vertex or node in a graph. Since edge between each vertex in a graph is used to represent the contiguity between each region and no direction attribute is used, will use undirected graph. The relationship between geographic map and its corresponding graph representation is shown in figure 1b.**



***Figure 1b: Left part is geographic map in real world and right part is its corresponding graph representation. Here V={1,2,3,4,5,6,7} and E= {(1,2),(1,7),(2,3),(2,6),(3,4),(3.6),(4,5),(4,6)}.***

**Suppose we use some algorithms to aggregate the graph above, the result will have several new groups (say three). For each new group, it can be considered as a new ‘super node’ which contains more than one ‘sub node’ in original graph and I can use ‘super edge’ to represent the contiguity between each group. Then in this case, I can create a new super graph which simply the graph structure while maintains the core aggregation results and the spatial relationship between each other. One example of the result using graph representation is shown in figure 1c.**



***Figure 1c: Left part is one partition result example. The graph is divided into three groups. The right part figure is the simplified super graph which only maintains the grouping result and the contiguity relationship between each group. Here, V= {}, E = {(), (), ()} while.***

**This concept is very important in this method and will be used in solution recombination. More details will be covered in that part later.**

**GAT Algorithm**

**This algorithm is first developed by Kim, Myung Jin in her PhD dissertation. It is a new heuristic method as well as greedy algorithm that can be used effectively in district redistribution problem.**

Suppose we want to aggregate census tract data to generate new regions such that each region has almost same population. The GAT algorithm will be explained based on this goal.

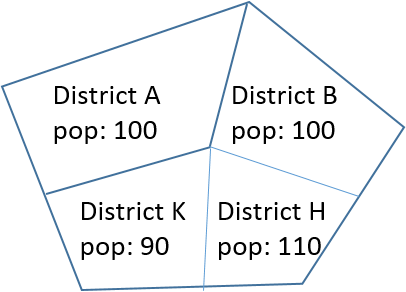
The algorithm consists of two parts: Initialization and Optimization.

* **Initialization**

The first step is to roughly generate a solution by randomly assigning census tracts to its adjacent group. We keep doing this until there is no census tract left. Since it just randomly assign census tract to its nearby group, it is hard to guarantee population equality based on initialization. Figure 2a displays one example result of initialization.

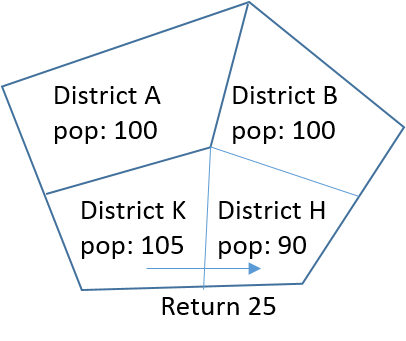
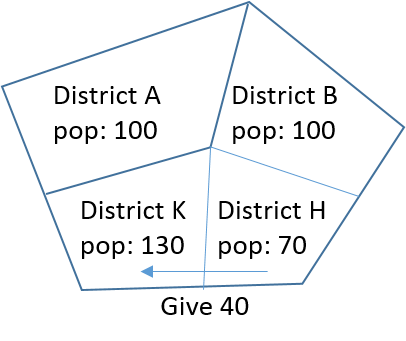
* **Optimization**

After having initial solution, Give and Take is applied to optimize this result by swapping census tracts that belong to different districts but are adjacent with each. First, it randomly picks two districts whose total population are not equal with each other. I assume we pick district c and district d and calculate the population difference between each other. Then we randomly pick a group of census units from high regions (here is H) and move these units into low regions (here is K). The total population of these selected regions should be larger than the population difference such that after moving, the population of region H now is less than the population of region K (see figure 2b). Then we repeated take single census unit back from K to H to make complement of that population gap until the total population that returned is greater than initial population difference (see figure 2c).



***Figure 2a: Initial result with District K and District H having different population.***

***Differences= pop (H)-pop (K)***



***Figure 2b: give 40 from H to K. Figure 2c: return 25 from K to H***

From this example, we can see that the population gap between district K and H has decreased. In GAT, it iterates this process until no more improvement can be made or it reaches the maximum number of iterations that user sets for it.

After exchanging units between districts, GAT will check whether this swapping violates the spatial contiguity constraints. According to **Kim, Myung Jin, it follows three steps:**

1. Randomly select a unit among those swapping units as a seed.
2. Based on this seed, keep adding adjacent units into this seed lists until no units can be added.
3. Calculate the total number of units that has been included into that seed lists (including the seed itself). If its number equals the total number units in that district, then this seed can be swapped.

Finally, objective function is used to evaluate the effectiveness of this algorithm. The objective function used in this paper is as follows:

Where P is the total population, n is the number of district that user want to generate, is census unit’s population and is the ideal population which can be calculated by *P* divided by *n*. Obviously the optimized process is to make the value of objective function as small as possible.

.

The detailed pseudo-code of this GAT algorithm is as follows:

**Input:** A Graph which represent the real world census map; number of regions that need to be aggregated; number of iterations that needs to be run.

**Output:** Aggregated results and objective function value.

Initialization ()

For each iterated solution :( # of iterations):

For each district,put all the district whose pop< target into the list\_low:

Randomly pick one from the list\_low, called LOW\_D:

Put all the districts which adjacent to LOW\_D and whose pop>target into the lists\_high

If there is no such district, return

Else

Randomly pick one from the lists,called HIGH\_D

Calculate the High pop from HIGH\_D

Calculate the Low pop from LOW\_D

Calculate the diff( called diff1)

Find the edges on border(units can be swapped)

uH=Find the border nodes belong to HIGH\_D

uL=Find the border nodes belong to LOW\_D

Swap=FALSE

While (! swap OR uH is not empty) do:

uH\_selected=randomly pick many from the uH

Remove the picked from uH

Calculate the pop from uH\_selected(actuall pop to be added)

While (returned pop<differ1 OR uL is not empty) do:

u=randomly pick one from uL

Remove the picked one from uL

Calculate the gap(gap=returned pop-differ1)

If gap>0:

Add the u into the uL\_selected

If gap<0:

Swap=DONE

If swap==DONE:

Add nodes uL\_selected to uH

Add nodes uH\_selected to uL

Remove nodes uL\_selected to uL

Remove nodes uH\_selected to uH

Reset graph

Valid\_solution()

If validated:

return result

Else

return NONE

***Figure 3d:pseudo-code of GAT algorithm***

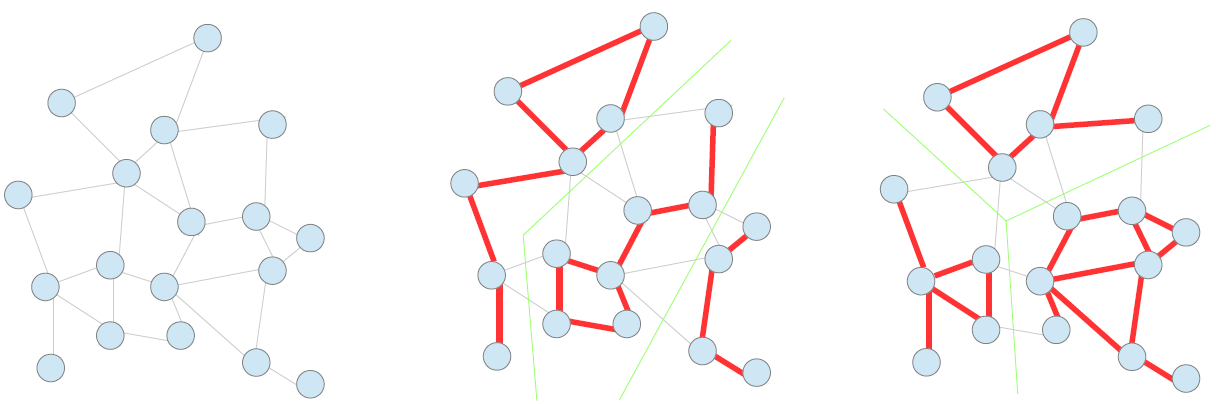
**Solution Recombination**

AlthoughGAT algorithm has achieved relatively good results in dealing with redistribution problem, it also has its limitations. Normally, with the number of iteration increasing, the value of objective function will decreases. But it only happens within a certain small range of iterations. When the total number of iteration reaches a certain level, the final result cannot be further improved no matter how many iterations will be added. Sometimes, due to the randomly selection process, the result may even be worse by constantly increasing its number of iteration. That’s one of its limitation. Solution recombination method is designed to solve this problem.

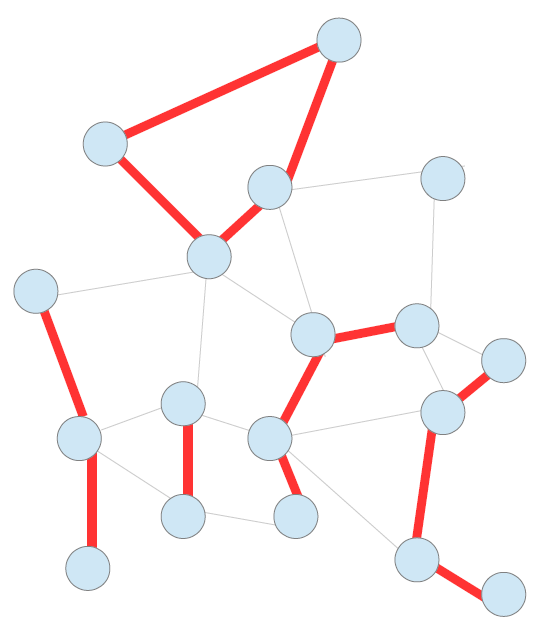
The main reason why the result cannot be further improved after its iteration reaching a certain level is that the overall structure of that aggregation has already reached its local optimal point and there are not much useful units to be swapped between each district. In other words, after reaching to a certain amount of iteration, the district structure has already be stable and any additional iterations won’t change much of its structure.

Just like gene evolution, it cannot be evolved into even better gene without incorporating different types of other gene, this aggregated structure cannot be further improved without taking advantages of different other optimal solutions. Solution recombination is such method which simulates evolutionary process and try to combine two already existing solutions into a new one which is better than by purely increasing the number of iterations.

Suppose we have two solutions for the same census map (see figure 3a) and we want to combine these two solutions into one and apply GAT again to produce an even better solution which GAT itself cannot do. What I did is to find common edges of these two solutions and these edges are kept while the rest of them are discarded. After this, I have a new graph whose number of subgraphs is greater than the number of districts we want to aggregate (see figure 3b). To construct a valid solution, we always merge the subgraph with the smallest total population to its neighbouring subgraph with the smallest population. After this procedure, the new graph has incorporated useful information both from itself and outside graph. Then GAT is applied to this merged solution to produce an even better solution. Detailed experiment and analysis will be elaborated in chapter 3.



***Figure 3a: Initial graph and its two different solutions, both of which have three subgraph. Each of consecutive part is considered as one subgraph.***



***Figure 3b: Subgraph that after finding common edges. Now it has five subgraph which is larger than three. Need to repeatedly find two subgraphs with least total population to merge until the total number of subgraph is equal to three. Then GAT can be applied to this valid merge solution.***

**Multi-solutions and Parallel Computing Technology**

The methods used in this part is to solve another limitation of GAT that it can only produce one good solution at a time. In order to produce many other good solutions, user has to run the program multiple times. This is not a big issue when dealing with small dataset, but the negative effect of high computation time becomes more obvious when dealing with large dataset. Multi-solution combined with parallel computing technology can be used to generate many good solutions at a time which can greatly reduce computation time.

Before digging into details of this idea, I will first introduce some prerequisites. First, a concept of Pool is introduced. Pool is a container that includes many solutions (can either be initial solutions or optimal solutions) at the same time. The GAT algorithm as well as solution recombination will be applied directly to this pool. Second, in order to let parallel computing technology work, the program has to be run in a multicore computer.

There are a few advantages to use pool to hold multiple solutions and let program to deal with pool directly instead of dealing with a single solution like traditional GAT. First, since pool can hold many solutions at the same time, when applying GAT algorithm combined with parallel computing to optimize it, user can get a pool of good results instead of just one good result. Besides, since we use parallel technique, many solutions can be generated and optimized in a parallel way such that the total amount of time it takes to generate many results is almost equal to the previously time that generates one result. Of course, time will sometimes be longer due to parallel computing communication and the limitation on number of cores a computer has. Thirdly, since we have a pool of solutions, information on different solutions is abundant. When applying to solution recombination, there are a lot of new candidate solutions can be selected to recombine, thus getting more chance of generating even better result. Of course, this also take the risk of generating bad result but in this case we can discard them. The later sensitive experiment will show that occurrence of good result occupies 60 to 80 percent of all results compared with bad ones which we don’t need to discard them and recalculate. Thus overall, this procedure still produce optimized pool of result while saving much computational time.

In summary, this process is a good simulation of evolution process: each solution is considered as single unity of the same species. The quality of them can either be good or bad. But a variety of them with different kind provide abundant source to generate all kinds of offspring which good quality remains while bad quality unity will be eliminated. In this case, after a certain period of time, only species with good quality remains.

The detailed workflow for combining all these three methods is described as follows: First, I generate a pool of initial random solutions using parallel computing technique. Then solutions in this pool are randomly chosen to be recombined and then GAT is used to improve the recombined solutions. Solutions in the pool will be replaced by better solutions which are generated through these process. Note that parallel computing is also used in this part which means multiple recombined process as well as GAT improvement can be done at the same time. The figure 4a demonstrate the whole process.



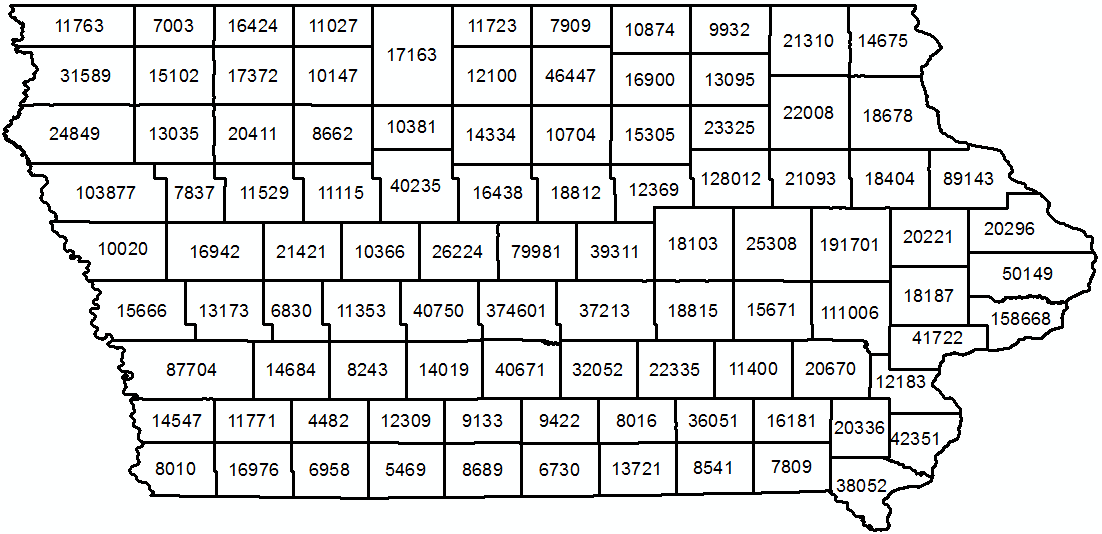
***Figure 4a: Overall process from beginning to end.***

**Experiment and Analysis**

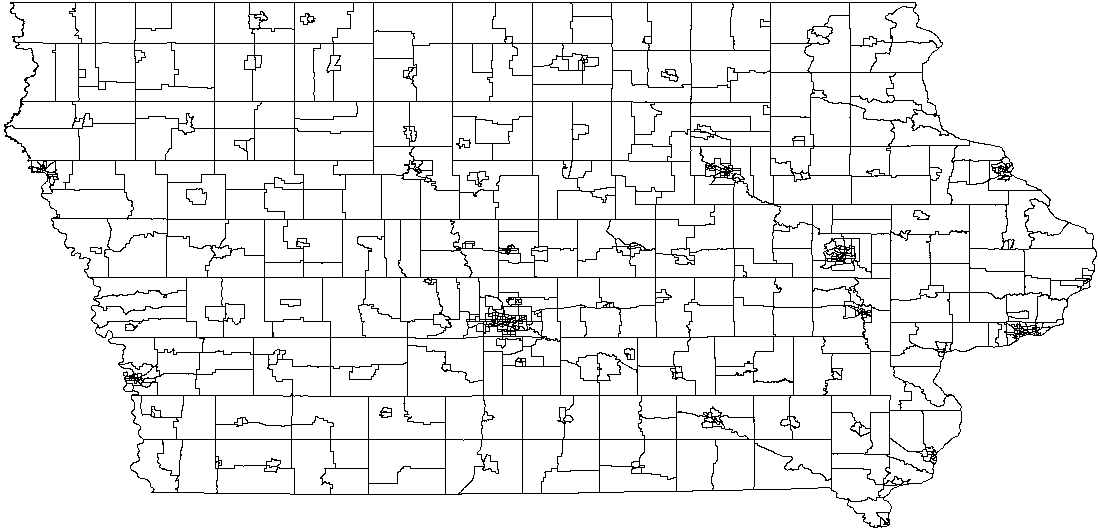
**Prerequisite**

* **Dataset**

The region I choose to conduct experiment in this paper is Iowa State, USA. Two levels of the data are used: Iowa census tract data and Iowa county level data (see figure 5a and 5b).These data can be downloaded from United State Census Bureau’s website (<http://www.census.gov/geo/maps-data/data/tiger.html>). Besides, US demographic data (<http://factfinder.census.gov/faces/nav/jsf/pages/index.xhtml>) which also comes from United State Census Bureaus is used to acquire population information.



***Figure 5a: Census county data, 99 regions with pop labelled on each region.***



***Figure 5b: Census tract data, with total of 825 sub regions. Due to too many regions, pop is not labelled on the map for better visualization.***

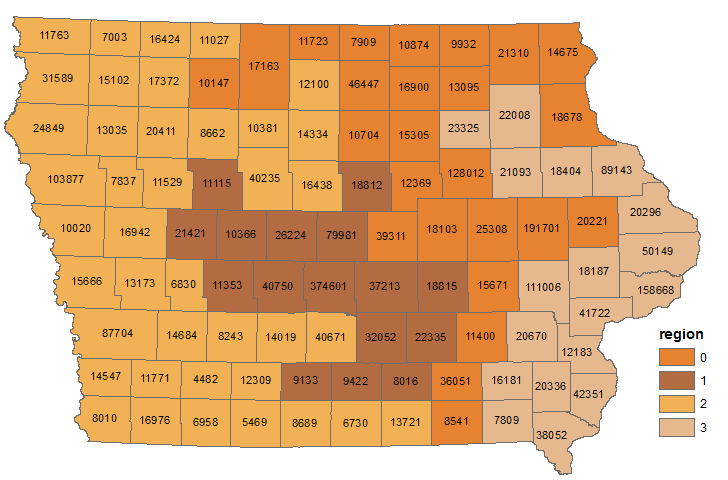
* **Programming environment**

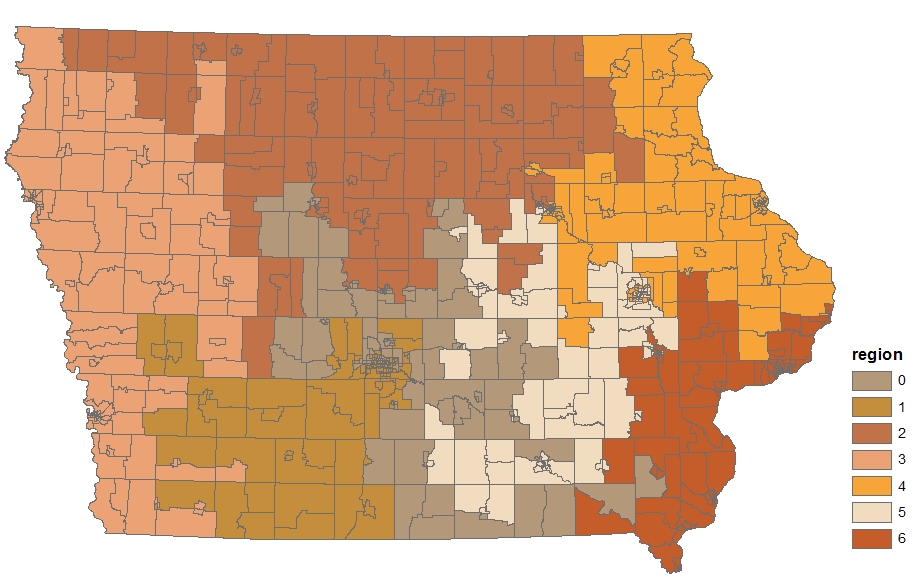
The whole program is written in python version 2.7.2. The type of CPU is Intel(R) Core(TM) i7-4700MQ and it has 4 cores with 8 logical processors. It is a 64-bit Operation system.

**Aggregation Result**

The final aggregation result will depend on the following parameters: number of new regions user wants to aggregate, total number of iterations, the size of the pool, the number of recombined solutions and the number of units that need to be aggregated. Objective function value as well as computational time varies according to different parameter inputs. How these parameter will have effect on the final performance will be systematically analysis in the next part. Experiments including aggregation, recombination will fix some of the parameters and only concentrates on the key part which corresponding to that experiment. More general sensitive evaluation will be done after this.

Figure 6a shows the result from both Iowa census tract and Iowa county data. Figure 6b is the corresponding population table.





***Figure 6-1-a: Upper part, aggregate census county data into 4 parts with 1000 iteration. ; Lower part, aggregate census tract data into 7 parts with 1000 iterations.***

|  |  |  |
| --- | --- | --- |
| Region | Count\_region | Population |
| 0 | 25 | 731550 |
| 1 | 16 | 731609 |
| 2 | 40 | 731582 |
| 3 | 18 | 731583 |

|  |  |  |
| --- | --- | --- |
| Region | Count\_region | Population |
| 0 | 109 | 433128 |
| 1 | 117 | 433183 |
| 2 | 133 | 433182 |
| 3 | 129 | 433186 |
| 4 | 119 | 433183 |
| 5 | 106 | 433181 |
| 6 | 112 | 433171 |

***Figure 6-1-b: left table, Census County population aggregation result with objective function value 0.0021. Right part, census tract population aggregation result with objective function value 0.00103.***

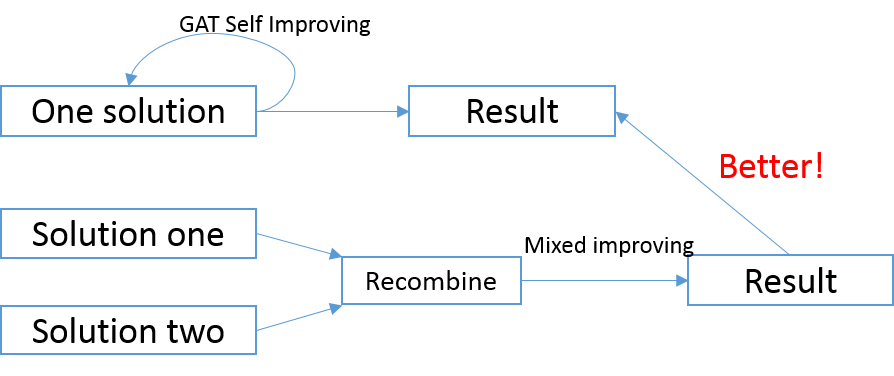
From the results we can show that, for each region, the GAT tries its best to equalize population equality while maintains spatial contiguity. When processing the data and determining whether other units are adjacent with a certain unit, what we do is to find all the spatial units whose boundaries has more than two points overlapping with the current unit. Since the swapping process is random, one of the negative effect of GAT is that it cannot guarantee a more reasonable new region shape despite its good performance. But multi-solution generation and processing in this paper mitigates such a negative effect because there are many alternatives for user to choose from pool.

Another limitation is that its final objective function cannot be further improved when the total number of iteration has reached to a certain level. Figure 6-1-c shows the tread between number of iteration and objective function value.

***Figure 6-1-c: before the number of iteration reaches to 1000, GAT works fine. However, after that, the improvement becomes slow. After 1100 iteration, the objective function value remains the same and cannot be further improved.***

**Solution Recombination Experiment**

Since only by increasing the number of iteration cannot further improve the final result when that number of iteration has reached to a certain level, more mixed information or graph structure needs to be used to gain more chance to generating new and better results. Solution recombination is designed for this purpose. In order to show that this method combined with GAT does show better performance compared with pure GAT, I design the experiment as follows: First, in each individual solution recombination experiment, other parameters such as number of iterations, number of regions to be aggregated, pool size are fixed. Second, generating two copies of the initial results from the pool. One group of this is used to run pure GAT while the other group is run both recombination and GAT. In order to compare them at the same level, the whole comparison process guarantees that the total number of iterations that pure GAT has is equal to the total number of recombination and applying GAT times. (See figure 6-2-a for workflow.) Finally, running the program to see and compare results. For evaluation and comparison, performance of both individual solution and a pool of solutions are tested.



***Figure 6-2-a: Experiment workflow***

* **Single Solution Evaluation**

10 groups of single solution are tested on Iowa county data with 800 iteration and 4 aggregated regions fixed. For each group, there are two initial solutions. Pure GAT is used to give additional iterations to see whether it can further improve the performance. Recombined method is used to combine two initial solutions and use GAT to improve that combined solution. Figure 6-2-b shows the result.

|  |  |  |  |
| --- | --- | --- | --- |
| Sol One | Sol Two | Add Iteration | Combined Add |
| 0.044151 | 5.765185 | 0.447456 | 0.007245 |
| 0.029320 | 0.134845 | 0.051054 | 0.035608 |
| 0.051122 | 0.145985 | 0.143115 | 0.068413 |
| 0.051669 | 0.068619 | 0.039093 | 0.036633 |
| 0.104978 | 0.261420 | 0.05618 | 0.008201 |
| 0.177561 | 3.287127 | 0.177561 | 0.125618 |
| 0.111061 | 5.294287 | 0.111061 | 0.102859 |
| 0.064791 | 12.618288 | 9.599757 | 0.031712 |
| 0.047021 | 2.523302 | 0.237363 | 0.019547 |

***Figure 6-2-b: single solution evaluation. For two initial solutions, Pure GAT is applied to add iterations for that worse solution. By comparing two methods, we find that pure GAT cannot further improve the objective function value or cannot improve that much compared with combined solution method.***

* **Multi-Solutions Evaluation**

Now it is time to evaluate the overall pool performance instead of single group solution by using pure GAT and Recombined method. Since there are multiple solutions in one pool (which means there are multiple objective function value), I use mean, median and standard deviation value to demonstrate the overall performance of that pool. In this experiment, I set pool size to be 10, recombine times to be 50 and number of iteration to be 800. The initial pool, only iteration pool and recombining pool is show in figure 6-2-c. The mean, median and standard deviation value is shown in figure 6-2-d.

From the value comparison (see figure 6-2-e), we can see that although both pure GAT and recombined method have made the overall result of that pool better, recombined one has less mean and median and standard deviation value which means the overall pool performance is better when applied recombined method compared with only GAT method. That’s one of the improvement this paper has made.

|  |  |  |
| --- | --- | --- |
| Initial Pool | Pure GAT | Recombined |
| 0.509308 | 0.002119 | 0.003827 |
| 0.609707 | 0.003212 | 0.005468 |
| 1.131522 | 0.003759 | 0.005741 |
| 1.418298 | 0.008885 | 0.006219 |
| 5.70757 | 0.0095 | 0.006493 |
| 6.549104 | 0.017086 | 0.006766 |
| 13.27078 | 0.021529 | 0.0095 |
| 17.01548 | 0.023306 | 0.011892 |
| 22.87143 | 0.028637 | 0.013396 |
| 64.85276 | 0.492085 | 0.019752 |

***Figure 6-2-c: Pool value from Initialization, Only iteration and combined method.***

|  |  |  |  |
| --- | --- | --- | --- |
|  | Mean | Median | Std |
| Initial | 13.3936 | 6.128337 | 18.655105 |
| Pure GAT | 0.061012 | 0.013293 | 0.143958 |
| Recombined | 0.008905 | 0.006629 | 0.004608 |

***Figure 6-2-d: Mean, Median and Standard deviation from Initial, Pure GAT, and recombined method.***

***Figure 6-2-e: Recombined Method’s mean, median and standard deviation value is less than Pure GAT method in terms of the overall performance in a pool of solutions.***

* **Sensitive Analysis**

GAT algorithm, due to its internal randomly swapping procedures, is not stable and cannot be guaranteed to generate optimal result every time. This side effect will also be brought into recombined method which makes recombining and applying GAT process cannot generate a better result every time. For the multi-solution generation, if the generated combined result is not better than either of those two selected solutions, the program will discard it and regenerate that again. In this case, this part may bring in some extra computation time. Sensitive analysis here is to determine how frequent this would happen. In this experiment, 20 times testing recombination process are run, and 11 of them generate better results at first time. This is not that good but it won’t have much effect on our final result of objective function value because the program will automatically discard that and do the recombine process again. As mentioned, the only side effect is that it will to some degree increase some extra computational time. Since we use parallel computing technology (which will be analysed later), overall, it still saves much time and make improved results compared with original pure GAT method.

**Parallel Computing Evaluation**

Parallel computing is used to solve another limitation of GAT that it can only generate one solution at a time. Since multi-solution generates multiple solutions from pool, by using parallel computing techniques, it can be run at the same time, thus saving a lot of time. This part will demonstrate how different number of thread will have effect on the computation time and CPU usage. Similar as before, this part only focus on time computation while keeps other parameters fixed.

Parallel computing technique is used in two parts in this paper. The first part is to generate multiple initial solutions. By using parallel computing technique, multiple initial solutions can be generated at same time, otherwise, it needs to loop each process many times. Taking Iowa census county data for example, set iteration to 1000 and pool size to be 10, figure 6-3-a shows the time they consume.

|  |  |
| --- | --- |
| Type | Time(s) |
| Without parallel | 10.5433 |
| With parallel | 1.58217 |

***Figure 6-3-a: Initial 10 solutions generation time comparison between program without and with parallel computing technique. The time has almost been reduced to one tenth.***

Similar to initial generation time, recombined process time can also be reduced by using parallel computing but the process is a little bit more complicated. It needs to take care of the shared memory and communication between multiple threads.

The time that can be reduced by parallel computing not only depends on the number of threads that are running on the program, but also on that computer processor’s limitation. For example, the computer that conducts this experiment has 8 logical processor which means that the operation system can at most distribute resources to let 8 threads to compute at the same time. Whenever one thread finishes its work, the operation system will dynamically distribute the resources and allow other thread to continue to work. In addition, the program only has one pool but each thread is run separated and writes results back when finishes. Since multiple process is run at the same time, there should be a parallel manager to manage all threads and combine these results together into that pool. In this paper, shared memory is used to store all the results. In order to maintain data consistence, which means read after write (RAW) and write after write (WAW) hazard won’t happen [14], each finished thread has to wait other running thread before results are written into shared memory. This waiting and communication process also takes some extra time but it is essential to get results right.

This experiment uses 2, 4, 5, 10, 20 threads to run the recombined and GAT program to see the time spent and its CPU usage. Figure 6-3-b shows the time and figure 6-3-c shows CPU usage. The experiment is run on a computer with 8 processors.

|  |  |
| --- | --- |
| Thread | Time(s) |
| 2 | 13.3114 |
| 4 | 8.600157 |
| 5 | 7.01485 |
| 10 | 6.215379 |
| 20 | 5.761735 |

***Figure 6-3-b: Time consumption varies with the number of threads.***

From the table and graph, we can see that when the number of threads is less than 8,with thread increasing, corresponding time decreases. But the time decreasing becomes less obvious when thread number is greater than 8 due to processor limitation. The reason why time reduction is not exactly matched with threat increasing when thread number is less than 8 (for example, thread number is doubled from 2 to 4 but time reduction is less than half) is because parallel waiting and communication takes some extra time to guarantee data consistence.

***Figure 6-3-c: CPU usage based on different number of threads. When the number of threads is greater than 8, the CPU usage is always 100%.***

**Algorithm Performance Evaluation**

How the algorithm parameters (including number of iterations, number of aggregated regions, recombine times, the size of a pool) have effect on the final performance will be systematically analysed in this part. As discussed in previous part, for a specific data, final results depends on a bunch of parameters which can be written as a set (see 6-4-a). Figure 6-4-b shows the parameter value in this experiment and the experiment will be combination of all these parameters. In other words, for each value in one type of parameter, every parameter in other types from the set has to be tested.

***Figure 6-4-a: Parameter set where RP is region part, RT is recombine time, PS is pool size and IT is iteration.***

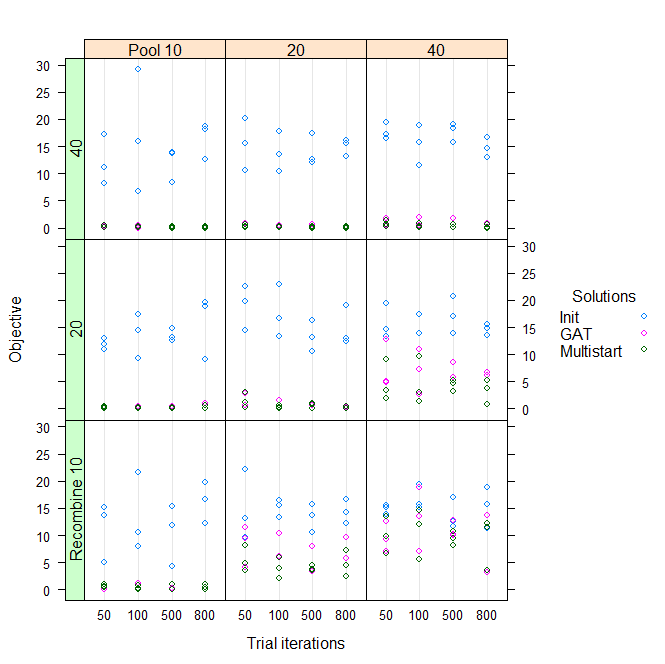
|  |  |  |  |
| --- | --- | --- | --- |
| RP | RT | PS | IT |
| 4 | 10 | 10 | 50 |
| 5 | 20 | 20 | 100 |
| 7 | 40 | 40 | 500 |
|  |  |  | 800 |
| RP | **RT** | **PS** | **IT** |
| 5 | 10 | 10 | 50 |
| 10 | 20 | 20 | 100 |
| 20 | 40 | 40 | 500 |
|  |  |  | 800 |

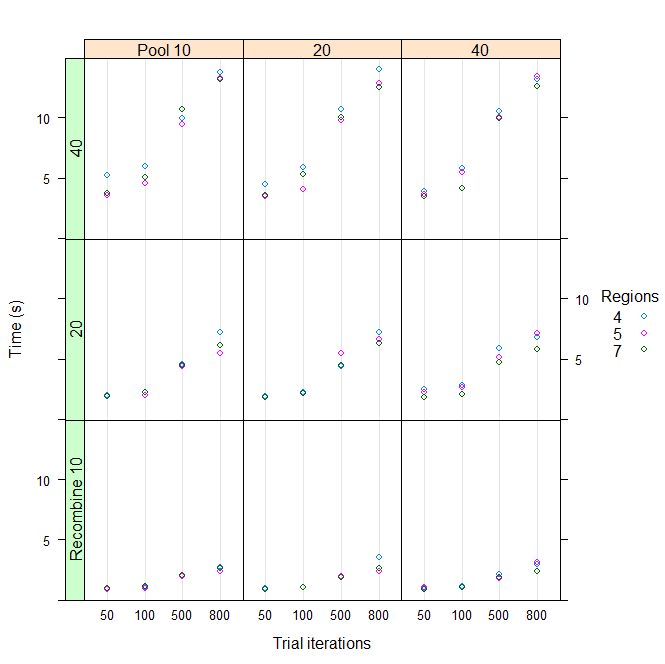
***Figure 6-4-b: Values for all the parameters that have effect on final performances. The experiments will based on all combination of these values. Upper table is census county parameters while lower table is census tract parameters.***

Figure 6-4-b shows the overall performance of census county data and census tract data respectively. Figure 6-4-c-1 and figure 6-4-c-2 shows the corresponding computational time. Here the performance is based on the median objective function value from that pool.

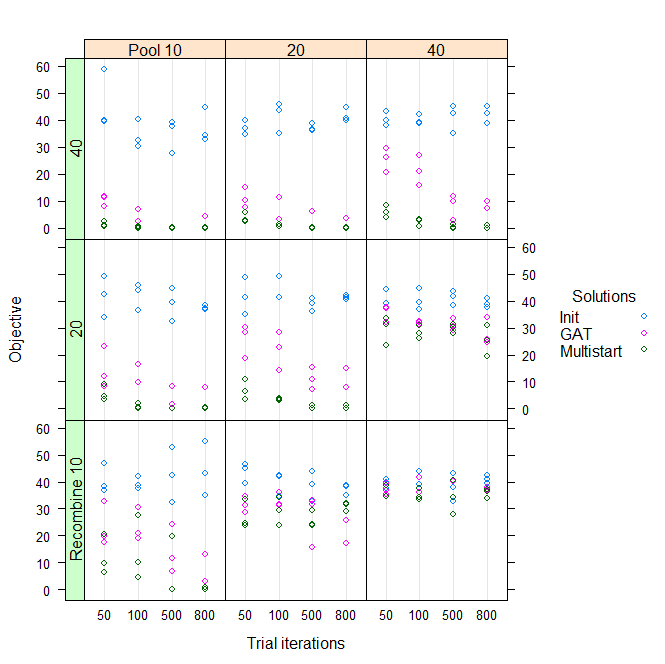
When the total number of iteration is low, the advantages of recombined method over pure GAT is not that obvious. That’s because the graph structure is not stable and still has much swapping space for further improvement. This can be seen from the 50 iterations group where some pure GAT results are not always below recombined method results. Things are getting better when the total number of iteration increases, especially 800 iterations, when pure GAT cannot further improve its own result but recombined method can still improve the final result by taking advantages of mixture information from other solutions. Another issue is about the pool size and recombined time. When the pool size is relatively small, say 10, recombine times need not to be very high to ensure good performance. When pool size becomes larger, low recombine times cannot guarantee much low median value from the pool and sometimes it is even closer to the initial solutions. This becomes even obvious when the total number of iterations is also very low. This evidence can be seen from the graph when recombine times is only 10 and iteration is 50 but pool size is 40. This makes sense for the following reasons: first, low iterations does not make much improvement towards to results. Second, for each recombined process, two solutions are randomly picked from the pool to do the combined and GAT method. When the pool size becomes larger, low recombine time is not enough for many initial solutions to be picked and combined. In this case, there are some solutions that not picked to be improved by our method which will make median value of that pool relatively high, or to some extreme, close to initial value since that value is the one that has not been improved. But this situation will be mitigated and ultimately disappear with increasing value of both recombine time and total number of iterations. From the graph, this means that pure GAT dots as well as recombined method dots become denser and they both have a large gap away from initial solution dots. Besides, dots from recombined method lays below dots from pure GAT.

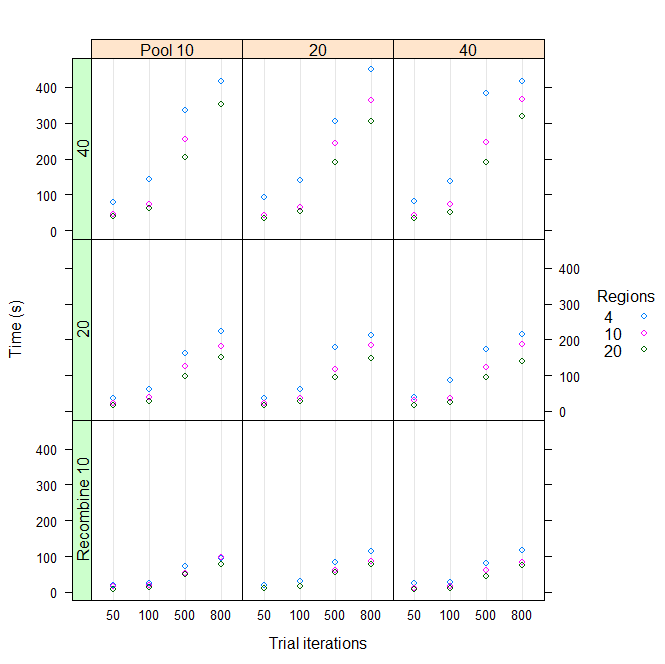
In terms of computational time, it is easy to see that more iterations, more recombine times lead to more computational time. But the number of new aggregated regions and pool size does not have much relationship with computational time. They only have effect on the final objective value.





***Figure 6-4-b:*** *performance and time of Census County*





***Figure 6-4-b:*** *performance and time of Census Tract*

**Summary and Discussion**

Zone redistribution problem can be considered as a geographic optimization problem and how to use computational approach efficiently to solve this problem becomes critical in this area. Openshaw [15], Fotheringham [16], Armstrong [17]’s research has laid solid foundation in this area whose specification is what we called quantitative geography or using computational and quantitative methods to solve geographic problem. In order to increase efficiency, improving computational methods is much better than increasing computer speed. Among all the methods that have been purposed so far, heuristic methods are considered as effective way to approximate the ultimate goal with tolerable computational time.

GAT is an efficient and effective heuristic method which can be used to solve political redistribution problem. But its result cannot be further improved when the total number of iteration has reached to a certain level. Besides it can only generate one result at a time. This paper solves these two limitations by incorporating the thinking of evolutionary algorithm to recombine two initial solutions and apply GAT after that. Parallel computing technique is also used to generate multiple optimal solutions at a time. Results and analysis show that it works and produce satisfactory results.

This research also have some limitations. First, the recombined method combined with GAT cannot always generate better result. This instability is due to the internal random swapping process of GAT which cannot guarantee optimal solution every time. In the future work, additional information may be added into the swapping process so that swapping process will not be purely random. It can save computational time and decrease the useless number of iterations. Besides, this paper only deals with population equality, but other factors like racial rate, income are also meaningful to be addressed. Whether this algorithm can be broadly extensible needs further experiment.

**Acknowledgement**

I want to show special appreciation to my advisor , Ningchuan Xiao, who has provided me much help about this paper thinking and code writing. He also gives me many valuable comments in writing this paper.

[1] Martin, David. "Geography for the 2001 Census in England and Wales."*Population Trends* 108 (2002): 7-15.

[2] Assunção, Renato M., et al. "Efficient regionalization techniques for socio‐economic geographical units using minimum spanning trees." *International Journal of Geographical Information Science* 20.7 (2006): 797-811.

[3] Kim, Myung Jin. "Optimization Approaches to Political Redistricting Problems." PhD diss., *The Ohio State University*, 2011.

[4] Bacao, Fernando, Victor Lobo, and Marco Painho. "Applying genetic algorithms to zone design." *Soft Computing* 9.5 (2005): 341-348.

[5] Xiao, Ningchuan. "A unified conceptual framework for geographical optimization using evolutionary algorithms." *Annals of the Association of American Geographers* 98.4 (2008): 795-817.

[6] Densham, Paul J., and Gerard Rushton. "A more efficient heuristic for solving largep-median problems." *Papers in Regional Science* 71.3 (1992): 307-329.

[7] Cova, Thomas J., and Richard L. Church. "Contiguity constraints for single‐region site search problems." *Geographical Analysis* 32.4 (2000): 306-329.

[8] Önal, Hayri, and Robert A. Briers. "Incorporating spatial criteria in optimum reserve network selection." *Proceedings of the Royal Society of London B: Biological Sciences* 269.1508 (2002): 2437-2441.

[9] Shirabe, Takeshi. "A model of contiguity for spatial unit allocation."*Geographical Analysis* 37.1 (2005): 2-16.

[10] Shirabe, Takeshi. "Districting modeling with exact contiguity constraints."*Environment and planning. B, Planning & design* 36.6 (2009): 1053.

[11] Bennett, David A., Ningchuan Xiao, and Marc P. Armstrong. "Exploring the geographic consequences of public policies using evolutionary algorithms."*Annals of the Association of American Geographers* 94.4 (2004): 827-847.

[12] Openshaw, Stan, and Liang Rao. "Algorithms for reengineering 1991 Census geography." *Environment and planning A* 27.3 (1995): 425-446.

[13] ALVANIDES, S., OPENSHAW, S. and REES, P., 2002, Designing your own geographies. *In The Census Data System*, P. Rees, D. Martin and P. Williamson (Eds), pp. 47–65 (Chichester, UK: Wiley).

[14] Hennessy, John L., and David A. Patterson. *Computer architecture: a quantitative* approach. Elsevier, 2012.

[15] Openshaw, Stan. "Computational human geography: Towards a research agenda." (1994): 499-505.

[16] Fotheringham, A. Stewart. "Trends in quantitative methods II: stressing the computational." Progress in Human Geography 22.2 (1998): 283-292.

[17] Bennett, David A., Ningchuan Xiao, and Marc P. Armstrong. "Exploring the geographic consequences of public policies using evolutionary algorithms."Annals of the Association of American Geographers 94.4 (2004): 827-847.